29th EACSL Annual Conference on Computer Science Logic

CSL 2021, January 25–28, 2021, Ljubljana, Slovenia (Virtual Conference)

Edited by

Christel Baier
Jean Goubault-Larrecq

LIPICS

Editors

Christel Baier
Technische Universität Dresden, Germany
christel.baier@tu-dresden.de

Jean Goubault-Larrecq
ENS Paris-Saclay, France
goubault@lsv.fr

ACM Classification 2012
Theory of computation → Logic

ISBN 978-3-95977-175-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-175-7.

Publication date
January, 2021

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 https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.CSL.2021.0

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

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the papers presented at CSL 2021, the 29th edition in the series of Computer Science Logic (CSL), the annual conference of the European Association for Computer Science Logic (EACSL).

CSL 2021 was held in Ljubljana, Slovenia, 25-28 January 2021, virtually due to the enduring SARS-Cov2 (Covid-19) outbreak.

CSL started as a series of international workshops, and became an international conference in 1992. Previous editions of CSL were held in Barcelona (2020), Birmingham (2018), Stockholm (2017), Marseille (2016), Berlin (2015), Vienna (2014), Torino (2013), Fontainebleau (2012), Bergen(2011), Brno (2010), Coimbra (2009), Bologna (2008), Lausanne (2007), Szeged (2006), Oxford (2005), Karpacz (2004), Vienna (2003), Edinburgh (2002), Paris (2001), Munich (2000), Madrid (1999), Brno (1998), Aarhus (1997), Utrecht (1996), Paderborn (1995), Kazimierz (1994), Swansea (1993) and San Miniato (1992).

CSL is an interdisciplinary conference, spanning across both basic and application-oriented research in mathematical logic and computer science. It is a forum for the presentation of research on all aspects of logic and applications, including automated deduction and interactive theorem proving, constructive mathematics and type theory, equational logic and term rewriting, automata and games, game semantics, modal and temporal logic, logical aspects of computational complexity, finite model theory, computational proof theory, logic programming and constraints, lambda calculus and combinatory logic, domain theory, categorical logic and topological semantics, database theory, specification, extraction and transformation of programs, logical aspects of quantum computing, logical foundations of programming paradigms, verification and program analysis, linear logic, higher-order logic, non-monotonic reasoning.

CSL 2021 received 82 submissions from 28 countries. The programme committee selected 34 papers for presentation at the conference. Each paper was reviewed by at least three members of the programme committee, with the help of external reviewers. The submission and reviewing process, programme committee discussion, and author notifications were all handled by the Easychair conference management system. In addition to the contributed papers, there were five invited talks, by

- Sophia Drossopoulou (Imperial College, London, UK)
- Bartek Klin (Universytet Warszawski, Warsawa, Poland)
- Assia Mahboubi (INRIA and LS2N, Université de Nantes, France, and Vrije Universiteit Amsterdam, the Netherlands)
- Sylvain Schmitz (Université de Paris, CNRS, IRIF, France and IUF, France)
- Linda Brown Westrick (Department of Mathematics, Penn State University, University Park, PA, USA).

We thank the five invited speakers for contributing to the success of the conference with their interesting talks and papers.

A special regular item in the CSL programme is the Ackermann Award presentation. This is the EACSL Outstanding Dissertation Award for Logic in Computer Science. This year, the jury decided to give the Ackermann Award for 2020 to

> Benjamin Lucien Kaminski for his PhD thesis *Advanced Weakest Precondition Calculi for Probabilistic Programs*

supervised by Joost-Pieter Katoen at the RTWH Aachen University, Germany.

The award was officially presented at the conference on January 26th, 2021. The citation of the award is included in the proceedings.

We are very grateful to all the members of the CSL 2021 programme committee and external reviewers for their careful and efficient evaluation of the papers submitted. We would like to thank also the members of the organisation committee, Alex Simpson, Andrej Bauer, and Danel Ahman, for taking care of every detail to make the conference enjoyable for all the participants, a task that was made all the more arduous because of the particular sanitary conditions of the year 2020. It was also a pleasure to work with Thomas Schwentick who, as the EACSL president, provided excellent guidance. The proceedings of CSL 2021 are published as a volume in the LIPIcs series. We thank Michael Wagner and all the Dagstuhl/LIPIcs team for their ongoing support and for the high quality preparation of these proceedings.

Christel Baier and Jean Goubault-Larrecq November 10th, 2020.

Programme Committee

Andreas Abel, University of Gothenburg, Sweden
Zena M. Ariola, University of Oregon, USA
Jeremy Avigad, Carnegie Mellon University, USA
Christel Baier, Technische Universität Dresden, Germany (co-chair)
Jasmin Blanchette, Vrije Universiteit Amsterdam, the Netherlands
Jean Goubault-Larrecq, ENS Paris-Saclay, France (co-chair)
Masahito Hasegawa, Kyoto University, Japan
Jean-Baptiste Jeannin, University of Michigan, USA
Michael Kaminski, Technion Haifa, Israel
Delia Kesner, Université de Paris, France
Laura Kovács, Vienna University of Technology, Austria
Martin Lange, University of Kassel, Germany
Sławomir Lasota, Warsaw University, Poland
Florin Manea, Georg-August Universität Göttingen, Germany
Stefan Milius, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany
Antoine Mottet, Charles University, Czech Republic
Andrzej Murawski, University of Oxford, UK
Elaine Pimentel, Universidade Federal do Rio Grande do Norte, Brasil
Sophie Pinchinat, IRISA Rennes, France
Simona Ronchi Della Rocca, Università di Torino, Italy
Krishna S, IIT Bombay, India
Peter Selinger, Dalhousie University, Canada
Sebastian Siebertz, Universität Bremen, Germany
Alex Simpson, University of Ljubljana, Slovenia (organizer)
Marie Van Den Bogaard, Université Libre de Bruxelles, Belgium
Yde Venema, Universiteit van Amsterdam, the Netherlands

Organisation Committee

Alex Simpson, University of Ljubljana (chair)
Andrej Bauer, University of Ljubljana
Danel Ahman, University of Ljubljana

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Reviewers

Andrea Aler Tubella, Matthew Anderson, Carlos Areces, Guillaume Aucher, Steve Awodey, Patrick Baillot, Chris Barrett, Christoph Berkholz, Raphaël Berthon, Dietmar Berwanger, Adrien Boiret, Johannes Borgström, Laura Bozzelli, Flavien Breuvart, James Brotherston, Guillaume Brunerie, Paola Bruscoli, Florian Bruse, Antonio Bucciarelli, Andrei Bulatov, David Cerna, Dmitry Chistikov, Jules Chouquet, Corina Cirstea, Ranald Clouston, Carmen Constantin, Gianluca Curzi, Luc Dartois, Ankush Das, Anupam Das, Rocco De Nicola, Ugo de'Liguoro, Antonin Delpeuch, Martin Desharnais, Andrej Dudenhefner, Arnaud Durand, Harley Eades III, Gabriel Ebner, Thomas Ehrhard, Kord Eickmeyer, Sebastian Enqvist, Martín Escardó, Santiago Escobar, Léo Exibard, Marie Fortin, Nissim Francez, Marco Gaboardi, Blaise Genest, Stéphane Graham-Lengrand, Stefan Göller, Daniel Hausmann, Willem Heijltjes, Léo Henry, Hugo Herbelin, Dirk Hofmann, Naohiko Hoshino, Mathieu Hoyrup, Pierre Hyvernat, Ismaël Jecker, Shin-Ya Katsumata, Kohei Kishida, Bartek Klin, Orna Kupferman, Clemens Kupke, Dietrich Kuske, Temur Kutsia, Ori Lahav, Engel Lefaucheux, Anela Lolic, Fosco Loregian, Michele Loreti, Etienne Lozes, Christof Löding, Florent Madelaine, Johann Makowsky, Giulio Manzonetto, Sonia Marin, Johannes Marti, Barnaby Martin, Andrea Masini, Ralph Matthes, Bastien Maubert, Klaus Meer, Arne Meier, Paul-André Melliès, George Metcalfe, Lukasz Mikulski, Fabio Mogavero, Georg Moser, Max New, Linh Anh Nguyen, Lê Thành Dũng Nguyễn, Andrey Nikolaev, Fredrik Nordvall Forsberg, Luca Paolini, Luc Pellissier, Mati Pentus, Guillermo Perez, Anton Pirogov, Alberto Policriti, Andrei Popescu, Amaury Pouly, Thomas Powell, Alexander Rabinovich, Roman Rabinovich, Revantha Ramanayake, Steven Ramsay, Luca Reggio, Giselle Reis, Christophe Ringeissen, Simon Robillard, Benjamin Rossman, Luca Roversi, Reuben Rowe, Jakub Rydval, David Sands, Gabriel Scherer, Joshua Schneider, Lutz Schröder, Francois Schwarzentruber, Alexandra Silva, Michał Skrzypczak, Sam Staton, Frank Stephan, Jonathan Sterling, Sorin Stratulat, Lutz Straßburger, Tony Tan, Szymon Toruńczyk, Patrick Totzke, Sophie Tourret, Riccardo Treglia, Henning Urbat, Tarmo Uustalu, Steffen van Bakel, Benno van den Berg, Peter Van Emde Boas, Femke Van Raamsdonk, Daniel Ventura, Oleg Verbitsky, Alexandre Vigny, Petar Vukmirović, Uwe Waldmann, Armin Weiss, Gregory Wilsenach, Anna Zamansky, Vladimir Zamdzhiev, Dmitriy Zhuk, Florian Zuleger.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Ackermann Award 2020

By Mikołaj Bojańczyk, Prakash Panangaden and Thomas Schwentick,
for the Jury of the EACSL Ackermann Award

The sixteenth Ackermann Award is presented at CSL'21 in Ljubljana, Slovenia (virtually). The 2020 Ackermann Award was open to any PhD dissertation on any topic represented at the annual CSL and LICS conferences that were formally accepted by a degree-granting institution in fulfillment of the PhD degree between 1 January 2018 and 31 December 2019. The Jury received eight nominations for the 2020 Award. The candidates came from a number of different countries around the world. The institutions at which the nominees obtained their doctorates represent six different countries in Europe and North America.

Again this year, EACSL Ackermann Award is generously sponsored by the association Alumni der Informatik Dortmund e.V.[1]

The topics covered a wide range of areas in Logic and Computer Science as represented by the LICS and CSL conferences. All submissions were of a very high quality and contained significant contributions to their particular fields. The jury wish to extend their congratulations to all the nominated candidates for their outstanding work.

The wide range of excellent candidates presented the jury with a difficult task. After an extensive discussion, one candidate stood out and the jury unanimously decided to award the **2020 Ackermann Award** to:

Benjamin Lucien Kaminski from Germany, for his thesis
Advanced Weakest Precondition Calculi for Probabilistic Programs
approved by *RWTH Aachen* in 2019.

Citation

Benjamin Lucien Kaminski receives the *2020 Ackerman Award* of the European Association of Computer Science Login (EACSL) for his thesis

Advanced Weakest Precondition Calculi for Probabilistic Programs.

The major contribution of the thesis is calculi – in the style of weakest precondition calculus – for tasks such as: proving bounds on expected running time (e.g. finite expected running time), proving almost sure termination, or computing conditional expected values. Due to the subtle nature of probabilistic programs, these are results which require extraordinary skill. At the same time, the thesis is expected to make – in fact, it already has made – an important impact due to the promising and wide-ranging applications. Finally, the quality of exposition is exemplary. With almost 400 pages of well chosen examples and lucid explanations, the thesis can serve as a textbook for newcomers in the field.

Background of the Thesis

Kaminski has made substantial advances in the analysis of probabilistic programs, a topic which – apart from its traditional importance in the study of programming languages – has

[1] www.cs.tu-dortmund.de/nps/en/Alumni/index.html

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

also recently assumed a central role in the field of machine learning. Program analysis is a major topic in programming languages since the seminal work of the Cousot's in the 1970's. In the case of probabilistic programs the issues are very subtle, since one cannot simply assert that a certain property holds. One must incorporate probabilistic reasoning at the very heart of the analysis. Following Dexter Kozen's seminal contributions to probabilistic program semantics and probabilistic dynamic logic in the 1980s, many workers have developed this subject.

There was a flowering of ideas connected with reasoning about probability since Kozen's work. These include Joost-Pieter Katoen, Kaminski's advisor, Christel Baier, Nathalie Bertrand and Marta Kwiatkowska among others on probabilistic model checking, Kim Larsen, Arne Skou, Josée Desharnais, Radha Jagadeesan, Vineet Gupta and Prakash Panangaden on probabilistic bisimulation and related metrics, Catuscia Palamidessi and her many collaborators on topics related to privacy and security, just to give a sample. In more recent times probabilistic programming languages like Church and Anglican have emerged from the machine learning community with fascinating results from, for example, Nate Ackerman, Cameron Freer and Dan Roy. It is clear that the topic of probabilistic programming languages and logics is not a niche topic, indeed an Ackerman Award had previously been given to Matteo Mio in 2013 for research on probabilistic logic.

Contributions of the Thesis

Kaminski's thesis extends the expectation calculus of McIver and Morgan which is the probabilistic analogue of Dijkstra's weakest precondition calculus in a significant way, by allowing signed measures instead of probability measures. His thesis contains a striking new compositional analysis of expected running times, which goes far beyond the older approach which could only infer almost-sure termination. Also for almost-sure termination, the thesis contains progress in the shape of a new proof rule, which can be applied, for example, to easily prove almost-sure termination of the 1-dimensional random walk. Apart from expected running times and almost-sure termination, Kaminski also develops a new calculus for reasoning about conditioning in probabilistic programs; which is of particular importance in the context of machine learning.

Finally, apart from proof calculi, the thesis also includes analysis of the computational complexity of deciding termination for probabilistic programs. A highlight of this analysis is the result that the questions "does a program terminate almost surely on a given input?" and "does a program terminate almost surely on all inputs?" occupy the same place in the arithmetical hierarchy; despite the seeming greater difficulty of the second question.

The numerous innovations in this thesis will have a long lasting impact on the theory and practice of probabilistic programming.

Biographical Sketch

Benjamin Kaminski carried out his PhD (under the supervision of Joost-Pieter Katoen) at RWTH in Aachen Germany, which is also where he completed his undergraduate studies. His work on probabilistic programs has received the Best Paper award at ETAPS 2016. He is currently a Lecturer at the University College of London.

Jury

The jury for the **Ackermann Award 2020** consisted of eight members, two of them *ex officio*, namely, the president and the vice-president of EACSL. In addition, the jury also included a representative of SIGLOG (the ACM Special Interest Group on Logic and Computation).

The members of the jury were:

- Christel Baier (TU Dresden),
- Michael Benedikt (University of Oxford),
- Mikołaj Bojańczyk (University of Warsaw),
- Jean Goubault-Larrecq (ENS Paris-Saclay),
- Prakash Panangaden (McGill University),
- Simona Ronchi Della Rocca (University of Torino), the vice-president of EACSL,
- Thomas Schwentick (TU Dortmund University), the president of EACSL,
- Alexandra Silva, (University College London), ACM SigLog representative.

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.

2015, Berlin:
Hugo Férée from France, and
Mickael Randour from Belgium.

2016, Marseille:
Nicolai Kraus from Germany

2017, Stockholm:
Amaury Pouly from France.
2018, Birmingham:
Amina Doumane from France.
2019, Barcelona (conference in 2020):
Antoine Mottet from France.

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

μ-Calculi with Atoms

Bartek Klin
University of Warsaw, Poland

Abstract

Modal μ-calculus is a well-known formalism for describing properties of state-based transition systems. It can define properties such as "[in the current state] p holds, and there is a path where is holds again at some point in the future", where p comes from some fixed vocabulary of basic predicates.

A formula of the classical μ-calculus refers only to finitely many basic predicates, which may sometimes seem restrictive. Real systems routinely operate on data coming from potentially infinite domains, such as numbers or character strings. Basic properties of such systems may reasonably include ones like "the number n was input", for every number n. It is then not clear how to say that "there exists a transition path where the currently input number is input again some time in the future" as a formula.

Various modal formalisms have been proposed to model temporal properties of systems that refer to data coming from infinite domains. Here I focus on the modal μ-calculus with atoms, which is an extension of the classical calculus with features of nominal sets. There, basic predicates, formulas and models rely on *atoms* that come from some fixed infinite domain and can be tested for equality (or, in an extended variant, for some fixed order).

I present a few variants of the modal μ-calculus with atoms, and describe their properties. As an example application, I show how to formulate the security property of the cryptographic Needham-Schroeder protocol, which relies on generating atomic nonces and comparing them for equality, and which famously fails due to a man-in-the-middle attack.

Much of the material presented in this talk is drawn from [1, 2, 3].

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking

Keywords and phrases modal μ-calculus, sets with atoms

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.1

Category Invited Talk

References

1 C. Eberhart and B. Klin. History-dependent nominal μ-calculus. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2019.

2 B. Klin and M. Łełyk. Scalar and Vectorial μ-calculus with Atoms. *Logical Methods in Computer Science*, Volume 15, Issue 4, 2019.

3 B. Klin and M. Łełyk. Modal μ-Calculus with Atoms. In *Procs. CSL 2017*, volume 82 of *LIPIcs*, pages 30:1–30:21, 2017.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 1; pp. 1:1–1:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Mathematical Structures in Dependent Type Theory

Assia Mahboubi
Inria, Nantes, France
LS2N, Université de Nantes, France
Vrije Universiteit Amsterdam, The Netherlands
`http://people.rennes.inria.fr/Assia.Mahboubi/`
Assia.Mahboubi@inria.fr

Abstract

In this talk, we discuss the role and the implementation of mathematical structures in libraries of formalised mathematics in dependent type theory.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Logic and verification

Keywords and phrases Mathematical structures, formalized mathematics, dependent type theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.2

Category Invited Talk

1 Summary

Since the early writings of the Nicolas Bourbaki group [3], *mathematical structures* are used in all fields of mathematics to structure the mathematical language, its vocabulary and its notational apparatus. An instance of a given structure is a carrier set equipped with some identified elements, with some operations on the carrier, and with some properties – called the axioms of the structure. Put in good use, these abstractions clarify the mathematical discourse for a knowledgeable audience, while emphasising correspondences between seemingly unrelated mathematical objects. Classical model theory provides a mathematical formalisation of the notion of *structure* [12], of which algebraic structures are an instance.

The past decade has seen the advent of several large-scale libraries of formalised mathematics [6, 2, 9, 17], most of which framed by a hierarchy of formal algebraic structures [10, 13, 8, 17]. The latter hierarchies can be seen as a formal-proof-engineering device, which organises inheritance and sharing in a similar way as the design patterns of object-oriented programming [7, 4]. The implementation and the features of these hierarchies depend both on the flavour of foundations the proof assistant is based on, and on the implementation in the prover of enhanced type inference procedures [15, 11, 16, 1, 14]. The central idea is to take benefit of some form of type inference in order to compute automatically the missing information in the user input, so as to achieve concision in the statement of formal sentences, while still providing a well-formed term to the prover's checker.

This talk will focus more specifically on the case of formalisations, and proof assistants, based on variants of dependent type theory. This setting allows in particular a first-class representation of structures using dependent tuples (also called *telescopes* [5]). It will discuss the recent techniques proposed to design these hierarchies, their pitfalls, the corresponding achievements, and their limitations.

References

1 Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. Hints in unification. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel,

editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2009. doi:10.1007/978-3-642-03359-9_8.

2 Grzegorz Bancerek, Czeslaw Bylinski, Adam Grabowski, Artur Kornilowicz, Roman Matuszewski, Adam Naumowicz, and Karol Pak. The role of the mizar mathematical library for interactive proof development in mizar. *J. Autom. Reason.*, 61(1-4):9–32, 2018. doi:10.1007/s10817-017-9440-6.

3 Nicholas Bourbaki. The Architecture of Mathematics. *The American Mathematical Monthly*, 57(4), 1950.

4 Cyril Cohen, Kazuhiko Sakaguchi, and Enrico Tassi. Hierarchy builder: Algebraic hierarchies made easy in coq with elpi (system description). In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 34:1–34:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.34.

5 N. G. de Bruijn. Telescopic mappings in typed lambda calculus. *Information and Computation*, 91(2):189–204, 1991.

6 Manuel Eberl, Gerwin Klein, Tobias Nipkow, Larry Paulson, and René Thiemann. Archive of Formal Proofs. https://www.isa-afp.org/about.html.

7 Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, 1st edition, 1994.

8 François Garillot, Georges Gonthier, Assia Mahboubi, and Laurence Rideau. Packaging mathematical structures. In *Theorem Proving in Higher-Order Logics (TPHOL 2009)*, volume 5674 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2009.

9 Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the odd order theorem. In *4th International Conference on Interactive Theorem Proving (ITP 2013), Rennes, France, July 22–26, 2013*, volume 7998 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 2013.

10 Adam Grabowski, Artur Kornilowicz, and Christoph Schwarzweller. On algebraic hierarchies in mathematical repository of Mizar. In Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki, editors, *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016*, volume 8 of *Annals of Computer Science and Information Systems*, pages 363–371. IEEE, 2016. doi:10.15439/2016F520.

11 Florian Haftmann and Makarius Wenzel. Constructive type classes in isabelle. In Thorsten Altenkirch and Conor McBride, editors, *Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, volume 4502 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 2006. doi:10.1007/978-3-540-74464-1_11.

12 Wilfrid Hodges. *Model Theory.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1993. doi:10.1017/CBO9780511551574.

13 Johannes Hölzl, Fabian Immler, and Brian Huffman. Type classes and filters for mathematical analysis in Isabelle/HOL. In *4th International Conference on Interactive Theorem Proving (ITP 2013), Rennes, France, July 22–26, 2013*, volume 7998 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2013.

14 Assia Mahboubi and Enrico Tassi. Canonical structures for the working Coq user. In *4th International Conference on Interactive Theorem Proving (ITP 2013), Rennes, France, July 22–26, 2013*, volume 7998 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2013.

15 Amokrane Saïbi. Typing algorithm in type theory with inheritance. In Peter Lee, Fritz Henglein, and Neil D. Jones, editors, *Conference Record of POPL'97: The 24th ACM SIGPLAN-*

SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, Paris, France, 15-17 January 1997, pages 292–301. ACM Press, 1997. doi:10.1145/263699.263742.

16 Matthieu Sozeau and Nicolas Oury. First-Class Type Classes. In *21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2008), Montréal, Québec, Canada, August 18–21, 2008*, volume 5170 of *Lecture Notes in Computer Science*, pages 278–293. Springer, 2008.

17 The mathlib Community. The Lean mathematical library. In *9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020), New Orleans, LA, USA, January 20–21, 2020*, pages 367–381. ACM, 2020.

Branching in Well-Structured Transition Systems

Sylvain Schmitz
Université de Paris, CNRS, IRIF, France
IUF, Paris, France
schmitz@irif.fr

Abstract

The framework of well-structured transition systems has been highly successful in providing generic algorithms to show the decidability of verification problems for infinite-state systems. In some of these applications, the executions in the system at hand are actually trees, and need to be "lifted" to executions over sets of configurations in order to fit in the framework. The downside of this approach is that we might lose precision when analysing the computational complexity of the algorithms, compared to reasoning over branching executions.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography; Theory of computation → Program reasoning; Theory of computation → Verification by model checking

Keywords and phrases fast-growing complexity, well-structured transition system

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.3

Category Invited Talk

Funding ANR-17-CE40-0028 BraVAS

Acknowledgements The presentation is based on joint work with Ranko Lazić.

1 Outline

In this talk, I intend to present a few ideas developed jointly with Ranko Lazić in [18] and investigate how to adapt the framework of *well-structured transition systems* (WSTS), due chiefly to Abdulla, Čerāns, Jonsson, and Tsay [1] and Finkel and Schnoebelen [10], in order to handle tree computations. The WSTS framework supplies generic algorithms for model-checking infinite-state systems, where the algorithms' termination relies on a *well-quasi-ordering* [16] of the configurations compatible with the transition relation.

Lifting Branching Systems. Well-structured transitions systems have found numerous applications since their inception in the 1990's, and these already encompass applications for infinite-state systems with branching executions rather than linear ones. In relation to logic in computer science, some of my favourite examples include provability in substructural logics like the conjunctive-implicational fragment of relevance logic [20, 25] or propositional linear logic with either contraction or weakening [17], and satisfiability for fragments of XPath over data trees [14, 6, 9].

Indeed, one can *lift* a branching transition relation to reason instead over linear executions over sets of configurations. Depending on the exact setup, the well-quasi-ordering on configurations is similarly lifted using either the *Smyth* quasi-ordering – also known as the *minoring* quasi-ordering – , or the *Hoare* quasi-ordering – also known as the *majoring* quasi-ordering. In the applications to substructural or data logics mentioned above, the configurations are essentially vectors of natural numbers in $\mathbb{N}^d$ for some d (ordered componentwise), and in those cases the two quasi-orderings over sets of configurations are well [13, 19] and compatible with the lifted transition relations, thereby defining a WSTS.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 3; pp. 3:1–3:3
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Algorithmic Complexity. While this lifting approach is successful for establishing decidability results, it is less so when trying to prove complexity upper bounds. In most algorithmic uses of well-quasi-orderings, one can rely on generic combinatorial analyses to extract upper bounds [7, 24, 21, 23, etc.]. The obtained bounds are typically non primitive-recursive, and depend primarily on the quasi-ordering. This approach has been applied to several classes of WSTS, and in many cases these gigantic worst-case complexity upper bounds are really a testament to the expressiveness of the corresponding classes of WSTS, as they are matched with tight lower bounds [12, 15, 11, 4, 21, 22, etc.].

In the case of the Smyth and Hoare quasi-orderings over subsets of $\mathbb{N}^d$ however, the complexity bounds on the lifted WSTS typically do not match the lower bounds. In that respect, Balasubramanian [3] recently improved the upper bounds of Abriola, Figueira, and Senno [2] and his hyper-Ackermannian bounds for the Hoare quasi-ordering over finite subsets of $\mathbb{N}^d$ are tight. But those lower bounds might not be realisable through the lifting of a branching transition system, and so far the known complexity lower bounds for all the mentioned applications [25, 5, 8, 17] are Ackermannian or lower.

References

1 Parosh A. Abdulla, Karlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1–2):109–127, 2000. doi:10.1006/inco.1999.2843.

2 Sergio Abriola, Santiago Figueira, and Gabriel Senno. Linearizing well-quasi orders and bounding the length of bad sequences. *Theoretical Computer Science*, 603:3–22, 2015. doi:10.1016/j.tcs.2015.07.012.

3 A. R. Balasubramanian. Complexity of controlled bad sequences over finite sets of $\mathbb{N}^d$. In *Proceedings of LICS 2020*. ACM, 2020. doi:10.1145/3373718.3394753.

4 Normann Decker and Daniel Thoma. On freeze LTL with ordered attributes. In *Proceedings of FoSSaCS 2016*, volume 9634 of *Lecture Notes in Computer Science*, pages 269–284. Springer, 2016. doi:10.1007/978-3-662-49630-5_16.

5 Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3):16, 2009. doi:10.1145/1507244.1507246.

6 Diego Figueira. Alternating register automata on finite words and trees. *Logical Methods in Computer Science*, 8(1), 2012. doi:10.2168/LMCS-8(1:22)2012.

7 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's Lemma. In *Proceedings of LICS 2011*, pages 269–278. IEEE, 2011. doi:10.1109/LICS.2011.39.

8 Diego Figueira and Luc Segoufin. Future-looking logics on data words and trees. In *Proceedings of MFCS '09*, volume 5734 of *Lecture Notes in Computer Science*, pages 331–343. Springer, 2009. doi:10.1007/978-3-642-03816-7_29.

9 Diego Figueira and Luc Segoufin. Bottom-up automata on data trees and vertical XPath. *Logical Methods in Computer Science*, 13(4), 2017. doi:10.23638/LMCS-13(4:5)2017.

10 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.

11 Christoph Haase, Sylvain Schmitz, and Philippe Schnoebelen. The power of priority channel systems. *Logical Methods in Computer Science*, 10(4):4:1–4:39, December 2014. doi:10.2168/LMCS-10(4:4)2014.

12 Serge Haddad, Sylvain Schmitz, and Philippe Schnoebelen. The ordinal recursive complexity of timed-arc Petri nets, data nets, and other enriched nets. In *Proceedings of LICS 2012*, pages 355–364. IEEE Press, 2012. doi:10.1109/LICS.2012.46.

13 Petr Jančar. A note on well quasi-orderings for powersets. *Information Processing Letters*, 72(5–6):155–160, 1999. doi:10.1016/S0020-0190(99)00149-0.

14 Marcin Jurdziński and Ranko Lazić. Alternating automata on data trees and XPath satisfiability. *ACM Transactions on Computational Logic*, 12(3), 2011. doi:10.1145/1929954.1929956.

15 Prateek Karandikar and Sylvain Schmitz. The parametric ordinal-recursive complexity of Post embedding problems. In *Proceedings of FoSSaCS 2013*, volume 7794 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2013. doi:10.1007/978-3-642-37075-5_18.

16 Joseph B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *Journal of Combinatorial Theory, Series A*, 13(3):297–305, 1972. doi:10.1016/0097-3165(72)90063-5.

17 Ranko Lazić and Sylvain Schmitz. Non-elementary complexities for branching VASS, MELL, and extensions. *ACM Transactions on Computational Logic*, 16(3):20:1–20:30, 2015. doi:10.1145/2733375.

18 Ranko Lazić and Sylvain Schmitz. The ideal view on Rackoff's coverability technique. *Information and Computation*, 2020. In press. doi:10.1016/j.ic.2020.104582.

19 Alberto Marcone. Fine analysis of the quasi-orderings on the power set. *Order*, 18(4):339–347, 2001. doi:10.1023/A:1013952225669.

20 Jacques Riche and Robert K. Meyer. Kripke, Belnap, Urquhart and relevant decidability & complexity. In *Proceedings of CSL 1998*, volume 1584 of *Lecture Notes in Computer Science*, pages 224–240. Springer, 1999. doi:10.1007/10703163_16.

21 Fernando Rosa-Velardo. Ordinal recursive complexity of unordered data nets. *Information and Computation*, 254(1):41–58, 2017. doi:10.1016/j.ic.2017.02.002.

22 Sylvain Schmitz. *Algorithmic Complexity of Well-Quasi-Orders*. Habilitation thesis, École Normale Supérieure Paris-Saclay, 2017. URL: http://tel.archives-ouvertes.fr/tel-01663266.

23 Sylvain Schmitz. The parametric complexity of lossy counter machines. In *Proceedings of ICALP 2019*, volume 132 of *Leibniz International Proceedings in Informatics*, pages 129:1–129:15. LZI, 2019. doi:10.4230/LIPIcs.ICALP.2019.129.

24 Sylvain Schmitz and Philippe Schnoebelen. Multiply-recursive upper bounds with Higman's Lemma. In *Proceedings of ICALP 2011*, volume 6756 of *Lecture Notes in Computer Science*, pages 441–452. Springer, 2011. doi:10.1007/978-3-642-22012-8_35.

25 Alasdair Urquhart. The complexity of decision procedures in relevance logic II. *Journal of Symbolic Logic*, 64(4):1774–1802, 1999. doi:10.2307/2586811.

Borel Sets in Reverse Mathematics

Linda Westrick
The Pennsylvania State University, University Park, PA, USA
https://personal.psu.edu/lzw299/
westrick@psu.edu

Abstract

We present what is known about the reverse mathematical strength of weak theorems involving Borel sets.

2012 ACM Subject Classification Theory of computation → Constructive mathematics

Keywords and phrases Borel sets, reverse mathematics, measure, category

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.4

Category Invited Talk

Funding *Linda Westrick*: This work was partially supported by NSF grant DMS-1854107.

Acknowledgements This talk includes joint work with many co-authors: Eric Astor, Damir Dzhafarov, Stephen Flood, Antonio Montalbán, Reed Solomon, Henry Towsner and Rose Weisshaar.

Theorems about Borel sets are often proved using arguments which appeal to some property of Borel sets, rather than recursing on the Borel structure of the set directly. For example, the statement "there is no Borel well-ordering of the reals" can be proved using either a measure or category argument. More generally, suppose we are given a theorem about Borel sets and a proof based on measure theory. Could the same theorem also be proved with a category argument? In principle, when the answer is "no", reverse mathematics provides a framework for proving this negative answer. However, early treatments of Borel sets in reverse mathematics (see [3]) used arithmetic transfinite recursion (ATR_0) as a base theory, and thus were not able to distinguish between a measure argument, a category argument, and a direct recursion.

Given a code for a Borel set B and an element x which may or may not be in B, a direct recursion on the structure of B is generally required to determine whether $x \in B$ or $x \notin B$. Therefore, in models of second order arithmetic which do not satisfy ATR_0, there are codes B and elements x for which the model is not powerful enough to see either $x \in B$ or $x \notin B$ [2]. To avoid giving artificial strength to theorems which simply assert that a Borel set has an element, in [1] we defined the notion of a *completely determined Borel set*. Roughly speaking, inside a given model, B is completely determined if for all x either $x \in B$ or $x \notin B$. This allows a separation between direct recursion on the one hand, and measure and category arguments on the other.

▶ **Theorem 1.** *The following principles are strictly weaker than* ATR_0*:*

1. *Every completely determined Borel set has the property of Baire. [1]*
2. *Every completely determined Borel set is measurable. [5]*

Furthermore, if $\mathcal{M}$ is an ω-model of (1), then for every $Z \in \mathcal{M}$ there is a $G \in \mathcal{M}$ that is $\Delta^1_1(Z)$-generic [1]. And if $\mathcal{M}$ is an ω-model of (2), then for every $Z \in \mathcal{M}$ there is an $R \in \mathcal{M}$ that is $\Delta^1_1(Z)$-random [5].

The hyperarithmetic sets, HYP, give the weakest ω-model of second order arithmetic in which the completely determined Borel sets are at all well behaved, being closed under countable unions and intersections. However, in HYP, one can construct some "Borel" sets by choice arguments.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 4; pp. 4:1–4:2
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Theorem 2** ([4]). *In HYP:*

1. *There is a completely determined Borel well-ordering of the reals.*
2. *Every completely determined Borel n-regular acyclic graph has a completely determined Borel 2-coloring.*
3. *The prisoners have a completely determined Borel winning strategy in the infinite prisoner hat game.*

We present what is known about the reverse mathematical strength of these and other weak theorems involving Borel sets.

References

1 Eric P. Astor, Damir Dzhafarov, Antonio Montalbán, Reed Solomon, and Linda Brown Westrick. The determined property of Baire in reverse math. *J. Symb. Log.*, 85(1):166–198, 2020. doi:10.1017/jsl.2019.64.

2 Damir Dzhafarov, Stephen Flood, Reed Solomon, and Linda Brown Westrick. Effectiveness for the Dual Ramsey Theorem. *CoRR*, Submitted 2017. arXiv:1710.00070.

3 Stephen G. Simpson. *Subsystems of second order arithmetic.* Perspectives in Logic. Cambridge University Press, Cambridge; Association for Symbolic Logic, Poughkeepsie, NY, second edition, 2009. doi:10.1017/CBO9780511581007.

4 Henry Towsner, Rose Weisshaar, and Linda Westrick. Borel combinatorics fail in HYP. *CoRR*, In preparation.

5 Linda Westrick. Completely determined Borel sets and measurability. *CoRR*, Submitted 2020. arXiv:2001.01881.

The Logic of Contextuality

Samson Abramsky
Department of Computer Science, University of Oxford, UK
`https://www.cs.ox.ac.uk/people/samson.abramsky/`
samson.abramsky@cs.ox.ac.uk

Rui Soares Barbosa[1]
INL – International Iberian Nanotechnology Laboratory, Braga, Portugal
`https://www.cs.ox.ac.uk/people/rui.soaresbarbosa/rsb`
rui.soaresbarbosa@inl.int

Abstract

Contextuality is a key signature of quantum non-classicality, which has been shown to play a central role in enabling quantum advantage for a wide range of information-processing and computational tasks. We study the logic of contextuality from a structural point of view, in the setting of partial Boolean algebras introduced by Kochen and Specker in their seminal work. These contrast with traditional quantum logic à la Birkhoff and von Neumann in that operations such as conjunction and disjunction are partial, only being defined in the domain where they are physically meaningful.

We study how this setting relates to current work on contextuality such as the sheaf-theoretic and graph-theoretic approaches. We introduce a general free construction extending the commeasurability relation on a partial Boolean algebra, i.e. the domain of definition of the binary logical operations. This construction has a surprisingly broad range of uses. We apply it in the study of a number of issues, including:

- establishing the connection between the abstract measurement scenarios studied in the contextuality literature and the setting of partial Boolean algebras;
- formulating various contextuality properties in this setting, including probabilistic contextuality as well as the strong, state-independent notion of contextuality given by Kochen–Specker paradoxes, which are logically contradictory statements validated by partial Boolean algebras, specifically those arising from quantum mechanics;
- investigating a Logical Exclusivity Principle, and its relation to the Probabilistic Exclusivity Principle widely studied in recent work on contextuality as a step towards closing in on the set of quantum-realisable correlations;
- developing some work towards a logical presentation of the Hilbert space tensor product, using logical exclusivity to capture some of its salient quantum features.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Quantum computation theory; Mathematics of computing

Keywords and phrases partial Boolean algebras, contextuality, exclusivity principles, Kochen–Specker paradoxes, tensor product

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.5

Funding *Samson Abramsky*: This author acknowledges support from EPSRC – Engineering and Physical Sciences Research Council, EP/T00696X/1, *Resources and Coresources: a junction between categorical semantics and descriptive complexity.*
Rui Soares Barbosa: This author acknowledges support from EPSRC – Engineering and Physical Sciences Research Council, EP/R044759/1, *Combining Viewpoints in Quantum Theory (Ext.)*, and from FCT – Fundação para a Ciência e a Tecnologia, CEECINST/00062/2018.

Acknowledgements The authors would like to thank Chris Heunen for helpful discussions.

[1] This work was carried out in part while RSB was at the School of Informatics, the University of Edinburgh.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 5; pp. 5:1–5:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Kochen and Specker's seminal work on quantum contextuality used the formalism of partial Boolean algebras [21]. In contrast to quantum logic in the sense of Birkhoff and von Neumann [7], partial Boolean algebras only admit physically meaningful operations. In the key example of $\mathsf{P}(\mathcal{H})$, the projectors on a Hilbert space $\mathcal{H}$, the operation of conjunction, i.e. product of projectors, becomes a partial one, only defined on **commuting** projectors.

In more recent work [20], Kochen developed a large part of the foundations of quantum theory in terms of partial Boolean algebras. Heunen and van den Berg [25] showed that every partial Boolean algebra is the colimit of its (total) Boolean subalgebras. Thus the topos approach to quantum theory [17] can be seen as a refinement, in explicitly categorical language, of the partial Boolean algebra approach. In this paper, we relate partial Boolean algebras to current frameworks for contextuality, in particular the sheaf-theoretic [5] and graph-theoretic [9] approaches.

A major role in our technical development is played by a general universal construction for partial Boolean algebras, which freely generates a new partial Boolean algebra from a given one and extra commeasurability constraints (Section 2.2, Theorem 1). This result is proved constructively, using an inductive presentation by generators and relations. It is used throughout the paper as it provides a flexible tool, subsuming a number of other constructions: some previously known, and some new.

We describe a construction of partial Boolean algebras from graphical measurement scenarios, i.e. scenarios whose measurement compatibility structure is given by a binary compatibility relation, or graph. Empirical models, i.e. correlations satisfying the no-signalling or no-disturbance principle, on these scenarios correspond bijectively to probability valuations, or states, on the corresponding partial Boolean algebras (Section 3.3).

We then turn our attention to contextuality properties. We discuss how probabilistic contextuality is formulated in the setting of partial Boolean algebras (Section 4.2), and show that the strong, state-independent form of contextuality considered by Kochen and Specker can be neatly captured using the universal construction mentioned above (Section 4.1, Theorem 16).

We also consider questions of quantum realisability, i.e. aiming to characterise the logical structure of partial Boolean algebras of projectors on a Hilbert space, and probability models that admit a Hilbert space realisation. This motivates us to propose a **Logical Exclusivity Principle** (LEP), which is always satisfied by partial Boolean algebras of the form $\mathsf{P}(\mathcal{H})$ (Section 5.1). We use a variant of our universal construction to show that there is a reflection between partial Boolean algebras and those satisfying LEP (Section 5.4, Theorem 26). We relate this Logical Exclusivity Principle to Specker's Exclusivity Principle for probabilistic models [8]. We show that if a partial Boolean algebra satisfies LEP, then all its states satisfy the Probabilistic Exclusivity Principle (PEP) (Section 5.3, Proposition 24). Moreover, we show that a state on a partial Boolean algebra satisfies PEP if it extends to one on its logically exclusive reflection, i.e. the freely generated partial Boolean algebra satisfying LEP (Section 5.3, Theorem 25).

In a similar vein, we consider the extent to which the tensor product operation on Hilbert spaces can be "tracked" by a corresponding operation on partial Boolean algebras. We first consider the tensor product described in [25, 20], which can be put in generator and relations form using our free construction (Section 6.1). It is easily seen that it fails to capture all the relations holding in the partial Boolean algebra of projectors on the Hilbert space tensor product. We then show that there is a natural monoidal structure on partial Boolean algebras satisfying LEP (Section 6.2). This contrasts with the situation for standard contextuality models satisfying Specker's Exclusivity Principle, which are not closed under tensor product. Both tensor product constructions above work by freely extending commeasurability starting from the coproduct of partial Boolean algebras. We show that such an operation never gives rise to Kochen–Specker paradoxes (Section 6.3). This can be seen as a limitative result for using such an approach to fully capture the Hilbert space tensor product in logical form, in terms of partial Boolean algebras.

We conclude with a discussion of some natural questions that arise from our results (Section 7).

2 Partial Boolean algebras

2.1 Basic definitions

A partial Boolean algebra A is given by a set (also written A), a reflexive, symmetric binary relation $\odot$ on A, read as "commeasurability" or "compatibility", constants 0 and 1, a total unary operation $\neg$, and partial binary operations $\wedge$ and $\vee$ with domain $\odot$. These must satisfy the following property: every set S of pairwise-commeasurable elements must be contained in a set T of pairwise-commeasurable elements which forms a (total) Boolean algebra[2] under the restrictions of the given operations.

Morphisms of partial Boolean algebras are maps preserving commeasurability, and the operations wherever defined. This gives a category $\mathbf{pBA}$.

Heunen and van den Berg show in [25, Theorem 4] that every partial Boolean algebra is the colimit, in $\mathbf{pBA}$, of the diagram $\mathcal{C}(A)$ consisting of its Boolean subalgebras and the inclusions between them.

2.2 Colimits and free extensions of commeasurability

In [25], the category $\mathbf{pBA}$ is shown to be cocomplete. Coproducts have a simple direct description. The coproduct $A \oplus B$ of partial Boolean algebras A, B is their disjoint union with 0_A identified with 0_B, and 1_A identified with 1_B. Other than these identifications, no commeasurability holds between elements of A and elements of B. By contrast, coequalisers, and general colimits, are shown to exist in [25] by an appeal to the Adjoint Functor Theorem. One of our technical contributions is to give a direct construction of the needed colimits, by an inductive presentation.[3]

More generally, we use this approach to prove the following result, which freely generates from a given partial Boolean algebra a new one where prescribed additional commeasurability relations are enforced between its elements.

▶ **Theorem 1.** *Given a partial Boolean algebra A and a binary relation $\circledcirc$ on A, there is a partial Boolean algebra $A[\circledcirc]$ such that:*

- *there is a $\mathbf{pBA}$-morphism $\eta \colon A \longrightarrow A[\circledcirc]$ satisfying $a \circledcirc b \;\Rightarrow\; \eta(a) \odot_{A[\circledcirc]} \eta(b)$;*
- *for every partial Boolean algebra B and $\mathbf{pBA}$-morphism $h \colon A \longrightarrow B$ satisfying $a \circledcirc b \Rightarrow h(a) \odot_B h(b)$, there is a unique $\mathbf{pBA}$-morphism $\hat{h} \colon A[\circledcirc] \longrightarrow B$ such that $h = \hat{h} \circ \eta$, i.e. such that the following diagram commutes*

$$\begin{tikzcd} A \arrow[r, "\eta"] \arrow[dr, "h"'] & A[\circledcirc] \arrow[d, "\hat{h}"] \\ & B \end{tikzcd}$$

[2] From now on, whenever we say just "Boolean algebra", we mean total Boolean algebra.

[3] For a well-known discussion of the advantages of an explicit construction over an appeal to the Adjoint Functor Theorem, see [18, p. *xvii*].

Table 1 Rules for free partial Boolean algebra with extended compatibility relation.

$$\frac{a \in A}{\imath(a)\downarrow} \qquad \frac{a \odot_A b}{\imath(a) \odot \imath(b)} \qquad \frac{a \circledcirc b}{\imath(a) \odot \imath(b)}$$

$$\frac{}{0 \equiv \imath(0_A),\ 1 \equiv \imath(1_A)} \qquad \frac{a \odot_A b}{\imath(a) \wedge \imath(b) \equiv \imath(a \wedge_A b),\ \imath(a) \vee \imath(b) \equiv \imath(a \vee_A b)} \qquad \frac{}{\neg\imath(a) \equiv \imath(\neg_A a)}$$

$$\frac{}{0\downarrow,\ 1\downarrow} \qquad \frac{t \odot u}{t \wedge u\downarrow,\ t \vee u\downarrow} \qquad \frac{t\downarrow}{\neg t\downarrow}$$

$$\frac{t\downarrow}{t \odot t,\ t \odot 0,\ t \odot 1} \qquad \frac{t \odot u}{u \odot t} \qquad \frac{t \odot u,\ t \odot v,\ u \odot v}{t \wedge u \odot v,\ t \vee u \odot v} \qquad \frac{t \odot u}{\neg t \odot u}$$

$$\frac{t\downarrow}{t \equiv t} \qquad \frac{t \equiv u}{u \equiv t} \qquad \frac{t \equiv u,\ u \equiv v}{t \equiv v} \qquad \frac{t \equiv u,\ u \odot v}{t \odot v}$$

$$\frac{\varphi(\vec{x}) \equiv_{\mathsf{Bool}} \psi(\vec{x}),\ \bigwedge_{i,j} u_i \odot u_j}{\varphi(\vec{u}) \equiv \psi(\vec{u})} \qquad \frac{t \equiv t',\ u \equiv u',\ t \odot u}{t \wedge u \equiv t' \wedge u',\ t \vee u \equiv t' \vee u'} \qquad \frac{t \equiv u}{\neg t \equiv \neg u}$$

We do not require that the relation $\circledcirc$ include the commeasurability relation $\odot_A$ already defined on A. Of course, it is the case that $A[\circledcirc] \cong A[\odot_A \cup \circledcirc]$ for any $\circledcirc$, but it will be notationally convenient to allow an arbitrary relation $\circledcirc$ in this construction. In particular, note that $A[\varnothing] \cong A[\odot_A] \cong A$.

As already mentioned, this result is proved constructively, by giving proof rules for commeasurability and equivalence relations over a set of syntactic terms generated from A. In fact, we start with a set of "pre-terms" and also give rules for definedness.

We define the set of **pre-terms** P inductively, to be the closure of the set of generators $G := \{\imath(a) \mid a \in A\}$ under the Boolean operations and constants. The standard theory of Boolean algebras gives us an equational theory $\equiv_{\mathsf{Bool}}$ for terms in the Boolean signature $\{0, 1, \wedge, \vee, \neg\}$ over variables $x, y, \ldots$ We have the usual notion of substitution of pre-terms for variables: if $\varphi(\vec{x})$ is a Boolean term in the variables $x_1, \ldots, x_n$, and if $u_1, \ldots, u_n$ are pre-terms, then $\varphi(\vec{u})$ is the pre-term which results from replacing x_i by u_i for all $i \in \{1, \ldots, n\}$.

We now define a predicate $\downarrow$ (definedness or existence), and binary relations $\odot$ and $\equiv$ on P, by the set of rules in Table 1. To illustrate the first rule on the last line, consider the distributivity axiom: $x \wedge (y \vee z) \equiv_{\mathsf{Bool}} (x \wedge y) \vee (x \wedge z)$. Under the assumptions $t \odot u$, $u \odot v$, $t \odot v$, we can infer $t \wedge (u \vee v) \equiv (t \wedge u) \vee (t \wedge v)$. Note that in this rule $\varphi(\vec{x})$ and $\psi(\vec{x})$ are pure Boolean terms, i.e. they do not contain generators.

One can show the following by structural induction on derivations, where $\vdash$ means derivability of an assertion from the rules.

▶ **Lemma 2.** *For all pre-terms t and u,*

1. $\vdash t \odot u$ *implies* $\vdash t\downarrow$ *and* $\vdash u\downarrow$*;*
2. $\vdash t \equiv u$ *implies* $\vdash t\downarrow$ *and* $\vdash u\downarrow$ *and* $\vdash t \odot u$*.*

We define the set of terms $T := \{t \in P \mid t\downarrow\}$. The relation $\equiv$ is an equivalence relation on T, by the rules in the fifth line. We define a structure $A[\circledcirc]$ as follows. The carrier is $T/{\equiv}$. The relation $\odot$ is defined by: $[t] \odot [u] \ :\Leftrightarrow\ \vdash t \odot u$. This is well defined due to the last rule on the fifth line. The operations are defined by representatives: if $[t] \odot [u]$, then $[t] \wedge [u] := [t \wedge u]$, etc. These are shown to be well defined using the congruence rules on the last line. The first rule on the last line now suffices to show that any set of pairwise-commeasurable elements of $A[\circledcirc]$ extends to a Boolean algebra, establishing the following proposition.

▶ **Proposition 3.** *$A[\circledcirc]$ is a partial Boolean algebra.*

There is a map $\eta\colon A \longrightarrow A[\circledcirc]$ sending a to $[\imath(a)]$. By the rules on the first two lines, this is a **pBA**-morphism which moreover satisfies $a \circledcirc b \;\Rightarrow\; \eta(a) \odot \eta(b)$.

Now, given a partial Boolean algebra B and a morphism $h\colon A \longrightarrow B$ such that $a \circledcirc b \;\Rightarrow\; h(a) \odot_B h(b)$, we shall show that there is a unique partial Boolean algebra morphism $\hat{h}\colon A[\circledcirc] \longrightarrow B$ such that $h = \hat{h} \circ \eta$.

We define a partial map $\gamma\colon P \rightharpoonup B$ by structural recursion on pre-terms:

$$\gamma(\imath(a)) := h(a) \qquad\qquad \gamma(t \wedge u) := \gamma(t) \wedge_B \gamma(u)$$
$$\gamma(\neg t) := \neg_B \gamma(t) \qquad\qquad \gamma(t \vee u) := \gamma(t) \vee_B \gamma(u)$$

Note that this map is partial because the operations $\wedge_B$ and $\vee_B$ are.

▶ **Proposition 4.** *For all pre-terms t and u, the following conditions hold:*

1. *$\vdash t\!\downarrow$ implies $\gamma(t)$ is defined;*
2. *$\vdash t \odot u$ implies $\gamma(t) \odot_B \gamma(u)$;*
3. *$\vdash t \equiv u$ implies $\gamma(t) = \gamma(u)$.*

Proof. The proof goes by structural induction on derivations from the rules. It suffices to verify that each rule is a valid statement about the partial Boolean algebra B when assertions about t, u are replaced by the corresponding assertions about $\gamma(t)$, $\gamma(u)$. Note that $\gamma(t) \odot_B \gamma(u)$ and $\gamma(t) = \gamma(u)$ are taken to imply, in particular, that $\gamma(t)$ and $\gamma(u)$ are well-defined elements of B.

For example, the third rule on the fifth line (transitivity of $\equiv$) gets translated to

$$\frac{\gamma(t) = \gamma(u),\;\; \gamma(u) = \gamma(v)}{\gamma(t) = \gamma(v)}$$

which simply expresses transitivity of equality. Most other cases are similar.

The first rule on the last line is the least straightforward. The induction hypothesis gives $\gamma(u_i) \odot_B \gamma(u_j)$ for all i and j, i.e. $\{\gamma(u_1), \ldots, \gamma(u_n)\}$ is a set of pairwise-commeasurable elements in B. It can therefore be extended to a Boolean subalgebra of B. This implies that for any pure Boolean term $\varphi(\vec{x})$, $\gamma(\varphi(u_1, \ldots, u_n)) = \varphi^B(\gamma(u_1), \ldots, \gamma(u_n))$ is well defined in B, and moreover that $\gamma(\varphi(\vec{u})) = \gamma(\psi(\vec{u}))$ whenever $\varphi(\vec{x}) \equiv_{\mathsf{Bool}} \psi(\vec{x})$, as required. ◀

Proof of Theorem 1. We can now establish the required universal property. We define $\hat{h}([t]) := \gamma(t)$. It follows straightforwardly from the definition of γ together with the previous proposition that this is well defined and has the required properties. ◀

This result will prove to be very useful in what follows.

Coequalisers and colimits

A variation of this construction is also useful, where instead of just forcing commeasurability, one forces equality. Given a partial Boolean algebra A and a relation $\circledcirc$ as before, we write $A[\circledcirc, \equiv]$ for the algebra generated by the above inductive construction, with one additional rule:

$$\frac{a \circledcirc a'}{\imath(a) \equiv \imath(a')}$$

We can define a $\mathbf{pBA}$-morphism $\eta\colon A \longrightarrow A[\circledcirc, \equiv]$ by $\eta(a) := [\imath(a)]$. Clearly this satisfies $a \circledcirc a' \;\Rightarrow\; \eta(a) = \eta(a')$. A simple adaptation of the proof of Theorem 1 establishes the following universal property of this construction.

▶ **Theorem 5.** *Let $h\colon A \longrightarrow B$ be a $\mathbf{pBA}$-morphism such that $a \circledcirc a' \;\Rightarrow\; h(a) = h(a')$. Then there is a unique $\mathbf{pBA}$-morphism $\hat{h}\colon A[\circledcirc, \equiv] \longrightarrow B$ such that $h = \hat{h} \circ \eta$.*

This result can be used to give an explicit construction of coequalisers, and hence general colimits, in $\mathbf{pBA}$. Given a diagram

$$\begin{tikzcd} A \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & B \end{tikzcd}$$

in $\mathbf{pBA}$, we define a relation $\circledcirc$ on B by $b \circledcirc b' := \exists a \in A.\, f(a) = b \;\wedge\; g(a) = b'$. Then, $\eta\colon B \longrightarrow B[\circledcirc, \equiv]$ is the coequaliser of f and g.

2.3 States on partial Boolean algebras

▶ **Definition 6.** *A **state** or **probability valuation** on a partial Boolean algebra A is a map $\nu\colon A \longrightarrow [0,1]$ such that:*

1. $\nu(0) = 0$*;*
2. $\nu(\neg x) = 1 - \nu(x)$*;*
3. *for all $x, y \in A$ with $x \odot y$, $\nu(x \vee y) + \nu(x \wedge y) = \nu(x) + \nu(y)$.*

▶ **Proposition 7.** *A map $\nu\colon A \longrightarrow [0,1]$ is a state iff for every Boolean subalgebra B of A, the restriction of ν to B is a finitely additive probability measure on B.*

▶ **Lemma 8.** *Let A be a partial Boolean algebra. There is a one-to-one correspondence between:*

- *states on A;*
- *families $(\nu_S)_{S \in \mathcal{C}(A)}$ indexed by the Boolean subalgebras S of A, where ν_S is a finitely additive probability measure on S and $\nu_S = \nu_T \circ \iota_{S,T}$ whenever $S \subseteq T$.*

▶ **Lemma 9.** *Let A be a finite Boolean algebra. There is a one-to-one correspondence between states on A and probability distributions on its set of atoms.*

Proof. Write X for the set of atoms of A. If $\nu\colon A \longrightarrow [0,1]$ is a state on A, then

$$\sum_{x \in X} \nu(x) = \nu\left(\bigvee X\right)$$

can be shown by induction on the size of X, using Definition 6–1 for the base case, and using Definition 6–3,1 and the fact that $x \wedge y = 0$ when x and y are distinct atoms for the induction step. Since $\bigvee X = 1$, we conclude that $\sum_{x \in X} \nu(x) = 1$ and so $\nu|_X\colon X \longrightarrow [0,1]$ is a probability distribution on X.

Conversely, if $d\colon X \longrightarrow [0,1]$ is a probability distribution, we extend it to the whole Boolean algebra using the fact that any element is uniquely written as the join of a set of atoms, as follows: for any $S \subseteq X$,

$$\nu\left(\bigvee S\right) := \sum_{x \in S} d(x)\ .$$ ◀

3 Graphical measurement scenarios and partial Boolean algebras

3.1 Measurement scenarios and (no-signalling) empirical models

We consider the basic framework of the sheaf-theoretic approach introduced in [5] to provide a unified perspective on non-locality and contextuality. Our focus here will not be solely on the question of contextuality, but also on principles that approximate the set of quantum-realisable behaviours.

Measurement scenarios provide an abstract notion of an experimental setup. They model a situation where there is a set of measurements, or queries, one can perform on a system, but not all of which may be performed simultaneously.

In this paper, we focus on what we term "graphical" scenarios, where a subset of measurements is compatible (i.e. can be performed together) if its elements are pairwise compatible. Hence, compatibility is specified simply by a binary relation. A paradigmatic example is quantum theory, where compatibility is given by commutativity: a set of measurements (observables) can be performed together if and only if its elements commute pairwise.

Note that, in contrast to [5], we do not require that the set of measurements be finite. We do, however, consider only measurements with a finite set of outcomes. This allows us to include within the scope of our discussion the scenario formed by all the quantum-mechanical observables on a system described by a finite-dimensional Hilbert space.

▶ **Definition 10.** *A **graphical measurement scenario** is a triple $\langle X, \frown, O\rangle$ consisting of:*

- *a set X of measurements,*
- *a reflexive, symmetric relation $\frown$ on X, indicating compatibility of measurements.*
- *a family $(O_x)_{x\in X}$ assigning a finite set O_x of outcomes to each measurement $x \in X$.*

*A **context** is a subset of measurements $\sigma \subseteq X$ that are pairwise compatible, i.e. a clique of the relation $\frown$. We write $\mathsf{Kl}(\frown)$ for the set of contexts.*

A particular case of interest is that of measurement scenarios where every measurement is dichotomic, i.e. has two possible outcomes.

Given a measurement scenario, an empirical model specifies particular probabilistic observable behaviour that may be displayed by a physical system.

▶ **Definition 11.** *Let $\langle X, \frown, O\rangle$ be a measurement scenario. A (no-signalling) **empirical model** is a family $(e_\sigma)_{\sigma\in\mathsf{Kl}(\frown)}$ where for each context $\sigma \in \mathsf{Kl}(\frown)$, e_σ is a probability distribution on the set $\mathcal{E}(\sigma) := \prod_{x\in\sigma} O_x$ of joint assignments of outcomes to the measurements in σ, and such that $e_\sigma = e_\tau|_\sigma$ whenever σ and τ are contexts with $\sigma \subseteq \tau$, where $e_\tau|_\sigma$ is marginalisation of distributions given as follows: for any $\mathbf{s} \in \mathcal{E}(\sigma)$,*

$$e_\tau|_\sigma(\mathbf{s}) := \sum_{\mathbf{t}\in\mathcal{E}(\tau), \mathbf{t}|_\sigma=\mathbf{s}} e_\tau(\mathbf{t}) \ .$$

*Such an empirical model is said to be **non-contextual** if there is a (global) probability distribution d on the set $\mathcal{E}(X) = \prod_{x\in X} O_x$ that marginalises to the empirical probabilities, i.e. such that $d|_\sigma = e_\sigma$ for all contexts $\sigma \in \mathsf{Kl}(\frown)$.*

The marginalisation condition in the definition of empirical models ($e_\sigma = e_\tau|_\sigma$ for contexts $\sigma \subseteq \tau$) ensures that the probabilistic outcome of a compatible subset of measurements is independent of which other compatible measurements are performed alongside these. This is sometimes referred to as the **no-disturbance condition** [24], or **no-signalling condition** [23] in the special case of Bell scenarios. This is a local compatibility condition, whereas non-contextuality can be seen as global compatibility: this justifies the slogan that contextuality arises from empirical data which is **locally consistent but globally inconsistent** [3, 2].

No-disturbance is satisfied by any empirical probabilities that can be realised in quantum mechanics [5]. However, this condition is much weaker than quantum realisability. Empirical models allow for behaviours that may be considered super-quantum, exemplified by the Popescu–Rohrlich (PR) box [23]. A lot of effort has gone into trying to characterise the set of quantum behaviours by imposing some additional, physically motivated conditions on empirical models, leading to various approximations from above to this quantum set.

3.2 Exclusivity principle on empirical models

One candidate for a property that is distinctive for the quantum case has appeared in various formulations as Local Orthogonality [11], Consistent Exclusivity [15], or Specker's Exclusivity Principle [8]. We shall refer to it as the Probabilistic Exclusivity Principle (PEP), since it is expressed as a constraint on probability assignments.

Informally, it says that if we have a family of pairwise exclusive events, then their probabilities must sum to at most 1. Of course, if all the events live on a single sample space, this would just be a basic property of probability measures. What gives the condition its force is that, in general, these events live on different, **incompatible** contexts. Thus, it reaches beyond the usual view of contexts as different classical "windows" on a quantum system, in which incompatible contexts are regarded as incommensurable.

We can give a precise formulation of PEP in terms of empirical models as follows. First, we say that events $s \in \mathcal{E}(\sigma)$ and $t \in \mathcal{E}(\tau)$ are **exclusive** if for some $x \in \sigma \cap \tau$, $s(x) \neq t(x)$. The principle holds for an empirical model $(e_\sigma)_{\sigma \in \mathsf{Kl}(\frown)}$ if for any family $\{s_i \in \mathcal{E}(\sigma_i)\}_{i \in I}$ of pairwise-exclusive events, then

$$\sum_{i \in I} e_{\sigma_i}(s_i) \; \leq \; 1.$$

This principle is valid in quantum-realisable empirical models, in which measurements correspond to observables, because incompatible (non-commuting) observables can share projectors, and exclusivity of outcomes with respect to common projectors implies **orthogonality**.

Although we know that PEP does not fully characterise the quantum-realisable empirical models, it stands as an important and fruitful principle [15, 6]. We wish to study this principle from the perspective of partial Boolean algebras.

3.3 From graphical measurement scenarios to partial Boolean algebras

To any graphical measurement scenario, we can associate a partial Boolean algebra whose states correspond to empirical models.

▶ **Definition 12.** *Let $\mathbf{X} = \langle X, \frown, O \rangle$ be a graphical measurement scenario. The partial Boolean algebra $A_{\mathbf{X}}$ is defined as follows:*

- *For each measurement $x \in X$, take B_x to be the finite Boolean algebra with atoms corresponding to the elements of O_x. We write $[x = o]$ for the atom of B_x corresponding to the outcome $o \in O_x$.*
- *Consider the partial Boolean algebra $A := \bigoplus_{x \in X} B_x$, the coproduct of all the Boolean algebras B_x taken in the category* **pBA**. *Note that all its elements are of the form $\imath_x(a)$ for a unique $x \in X$ and $a \in B_x$, except for the constants* 0 *and* 1.

- *Define the following relation* $\odot$ *on the elements of* A*:*

$$\imath_x(a) \odot \imath_y(b) \qquad \textit{iff} \qquad x \frown y \;\; \textit{or} \;\; a \in \{0,1\} \;\; \textit{or} \;\; b \in \{0,1\} \; .$$

- *Take* $A_{\mathbf{X}} := A[\odot]$*, the extension of* A *by the relation* $\odot$*, as given by Theorem 1.*

We can give an alternative description using colimits.

▶ **Definition 13.** *Let* $\mathbf{X} = \langle X, \frown, O \rangle$ *be a graphical measurement scenario. The partial Boolean algebra* $B_{\mathbf{X}}$ *is defined as follows:*

- *For each measurement* $x \in X$*, let* B_x *be as in Definition 12.*
- *For each context* $\sigma \in \mathsf{Kl}(\frown)$*, let* $B_\sigma := \sum_{x\in\sigma} B_x$*, the coproduct of all the* B_x *with* $x \in \sigma$*, taken in the category* **BA** *of Boolean algebras.*[4]
- *Given contexts* $\sigma, \tau \in \mathsf{Kl}(\frown)$ *with* $\sigma \subseteq \tau$*, there is a Boolean algebra homomorphism* $\iota_\sigma^\tau \colon B_\sigma \longrightarrow B_\tau$ *given by the obvious injection.*
- *Take* $B_{\mathbf{X}}$ *to be the colimit in the category* **pBA** *of the diagram consisting of the Boolean algebras* $(B_\sigma)_{\sigma\in\mathsf{Kl}(\frown)}$ *and the inclusions* $(\iota_\sigma^\tau)_{\sigma\subseteq\tau\in\mathsf{Kl}(\frown)}$*.*

Note that the colimit in this instance can be given explicitly in a closed form, as it is that of a diagram of Boolean algebras and inclusions satisfying the conditions of Kalmbach's "bundle lemma" [19, 1.4.22]. The carrier set of $B_{\mathbf{X}}$ is the union of all the B_σ modulo the identifications along inclusions ι_σ^τ. The Boolean subalgebras of $B_{\mathbf{X}}$ are exactly those in the family $(B_\sigma)_{\sigma\in\mathsf{Kl}(\frown)}$.

▶ **Proposition 14.** *The two descriptions coincide: for any* $\mathbf{X}$*,* $A_{\mathbf{X}} \cong B_{\mathbf{X}}$*.*

▶ **Proposition 15.** *For any graphical measurement scenario* $\mathbf{X}$*, there is a one-to-one correspondence between states on* $A_{\mathbf{X}}$ *and empirical models on* $\mathbf{X}$*.*

Proof. This follows from the fact that the Boolean subalgebras of $A_{\mathbf{X}}$ are the family $(B_\sigma)_{\sigma\in\mathsf{Kl}(\frown)}$, by applying Lemmas 8 and 9, and noting that the condition $\nu_S = \nu_T \circ \iota_{S,T}$ for $S \subseteq T$ on states on Boolean subalgebras translates under the correspondence in Lemma 9 to marginalisation of probability distributions. ◀

4 Partial Boolean algebras and contextuality

We consider some aspects of contextuality formulated in the framework of partial Boolean algebras, and relate them to the free construction from Theorem 1.

4.1 The Kochen–Specker property

The Kochen–Specker theorem, as originally stated [21], is that there are partial Boolean algebras of Hilbert space projectors with no **pBA**-morphisms to **2**, the two-element Boolean algebra. Since every (non-trivial[5]) Boolean algebra has a homomorphism to **2**, this implies that such a partial Boolean algebra A has no morphism to **any** (non-trivial) Boolean algebra.

Now, **BA** is a full subcategory of **pBA**. We know from [25] that A is the colimit in **pBA** of the diagram $\mathcal{C}(A)$ consisting of its Boolean subalgebras and inclusions between them. Let B be the colimit in **BA** of the same diagram $\mathcal{C}(A)$. Then, the cone from $\mathcal{C}(A)$ to B is also a cone in **pBA**, hence there is a mediating **pBA**-morphism from A to B.

[4] Note that the set of atoms of such a coproduct Boolean algebra is the cartesian product of the sets of atoms of each of the summands. Hence an atom of B_σ corresponds to an assignment of an outcome in O_x to each measurement $x \in \sigma$.

[5] See the following discussion.

To resolve the apparent contradiction, note that **BA** is an equational variety of algebras over **Set**. As such, it is complete and cocomplete, but it also admits the one-element Boolean algebra $\mathbf{1}$, in which $0 = 1$. Note that the trivial Boolean algebra $\mathbf{1}$ does **not** have a homomorphism to $\mathbf{2}$.

We can conclude from the discussion above that a partial Boolean algebra satisfies the Kochen–Specker property of not having a morphism to $\mathbf{2}$ if and only if the colimit in **BA** of its diagram of Boolean subalgebras is $\mathbf{1}$. In fact, we could formulate this property directly for diagrams of Boolean algebras, without referring to partial Boolean algebras at all: a diagram in **BA** is K–S if its colimit in **BA** is $\mathbf{1}$. We could say that such a diagram is "implicitly contradictory" since in trying to combine all the information in a colimit we obtain the manifestly contradictory $\mathbf{1}$.

Finally, this property admits a neat formulation in terms of the free extension of partial Boolean algebras by a relation, reminiscent of the definition of a perfect group.

▶ **Theorem 16.** *Let A be a partial Boolean algebra. The following are equivalent:*

1. *A has the K–S property, i.e. it has no morphism to $\mathbf{2}$.*
2. *The diagram $\mathcal{C}(A)$ of Boolean subalgebras of A is K–S, i.e. its colimit in* **BA** *is $\mathbf{1}$.*
3. *$A[A^2] = \mathbf{1}$.*

Proof. The equivalence between the first two statements follows from the discussion above. Now, all elements are commeasurable in $A[A^2]$, so it is a Boolean algebra. There is a morphism $A \longrightarrow \mathbf{2}$ if and only if there is a morphism $A[A^2] \longrightarrow \mathbf{2}$, by the universal property of $A[A^2]$ (in the $\Rightarrow$ direction) or composition with $\eta \colon A \longrightarrow A[A^2]$ (in the $\Leftarrow$ direction). Since $A[A^2]$ is a Boolean algebra, this is in turn equivalent to $A[A^2]$ being non-trivial. In other words, there is no morphism $A \longrightarrow \mathbf{2}$ if and only if $A[A^2] = \mathbf{1}$. ◀

4.2 Probabilistic contextuality

The notion of contextuality for states also admits a formulation in this setting.

▶ **Definition 17.** *A state $\nu \colon A \longrightarrow [0,1]$ on a partial Boolean algebra A is said to be* ***non-contextual*** *if it extends to $A[A^2]$, i.e. if there is a state $\hat{\nu} \colon A[A^2] \longrightarrow [0,1]$ such that $\nu = \hat{\nu} \circ \eta$.*

By the universal property of $A[A^2]$, this is equivalent to requiring that there be some Boolean algebra B, a morphism $h \colon A \longrightarrow B$, and state $\hat{\nu} \colon B \longrightarrow [0,1]$ such that $\nu = \hat{\nu} \circ \eta$.

▶ **Proposition 18.** *Let $\mathbf{X}$ be a graphical measurement scenario. A state on $A_{\mathbf{X}}$ is contextual in the sense of Definition 17 if and only if the corresponding empirical model under the correspondence of Proposition 15 is contextual in the sense of Definition 11.*

Note that if A has the Kochen–Specker property, then $A[A^2] = \mathbf{1}$, and since there is no state on $\mathbf{1}$, every state of A is necessarily contextual. An advantage of partial Boolean algebras is that the K–S property provides an intrinsic, logical approach to defining **state-independent contextuality**.

5 Exclusivity principles for partial Boolean algebras

We now consider exclusivity principles from the partial Boolean algebra perspective. This will subsume the previous discussion on PEP for empirical models in graphical measurement scenarios.

We introduce two exclusivity principles: one acts at the "logical" level, i.e. the level of events or elements of a partial Boolean algebra, whereas the other acts at the "probabilistic" level, applying to states of a partial Boolean algebra.

5.1 Logical exclusivity principle (LEP)

The basic ingredient is a notion of exclusivity between events (or elements) of a partial Boolean algebra. Given a partial Boolean algebra A and elements $a, b \in A$, we write $a \leq b$ to mean that $a \odot b$ and $a \wedge b = a$. Note that the restriction of this relation $\leq$ to any Boolean subalgebra of A coincides with the partial order underlying that Boolean algebra.

▶ **Definition 19.** *Let A be a partial Boolean algebra. Two elements $a, b \in A$ are said to be* ***exclusive****, written $a \perp b$, if there is an element $c \in A$ such that $a \leq c$ and $b \leq \neg c$.*

Note that $a \perp b$ is a weaker requirement than $a \wedge b = 0$, although the two would be equivalent in a Boolean algebra. The point is that in a general partial Boolean algebra one might have exclusive events that are not commeasurable (and for which, therefore, the $\wedge$ operation is not even defined).

▶ **Definition 20.** *A partial Boolean algebra is said to satisfy the* ***logical exclusivity principle (LEP)*** *if any two elements that are exclusive are also commeasurable, i.e. if $\perp \subseteq \odot$.*

We write **epBA** *for the full subcategory of* **pBA** *whose objects are partial Boolean algebras satisfying LEP.*

Logical exclusivity and transitivity

The logical exclusivity principle turns out to be equivalent to the following notion of transitivity [22, 16].

▶ **Definition 21.** *A partial Boolean algebra is said to be* ***transitive*** *if for all elements a, b, c, $a \leq b$ and $b \leq c$ implies $a \leq c$.*

Transitivity can fail in general for a partial Boolean algebra, since one need not have $a \odot c$ under the stated hypotheses. Note that the relation $\leq$ on a partial Boolean algebra is always reflexive and anti-symmetric, so this condition is equivalent to $\leq$ being a partial order (globally) on A. A partial Boolean algebra of the form $\mathsf{P}(\mathcal{H})$ is always transitive.

▶ **Proposition 22.** *Let A be a partial Boolean algebra. Then it satisfies LEP if and only if it is transitive.*

Proof. Suppose that A satisfies LEP, $a \leq b$, and $b \leq c$. Then $\neg c \leq \neg b$. Hence, by LEP, $a \odot \neg c$, and so $a \odot \neg\neg c = c$. Now, $a \wedge c = (a \wedge b) \wedge c = a \wedge (b \wedge c) = a \wedge b = a$, showing that $a \leq c$.

Conversely, suppose that A is transitive, $a \leq c$, and $b \leq \neg c$. Then, $c = \neg\neg c \leq \neg b$, hence $a \leq \neg b$ by transitivity. In particular, $a \odot \neg b$, and so $a \odot \neg\neg b = b$. ◀

As an immediate consequence, any $\mathsf{P}(\mathcal{H})$ satisfies LEP.

It is shown in [14] that a partial Boolean algebra is transitive if and only if it is an orthomodular poset.

5.2 Probabilistic exclusivity principle (PEP)

We now consider an analogous principle applying at the probabilistic level, i.e. at the level on states of a partial Boolean algebra.

▶ **Definition 23.** *Let A be a partial Boolean algebra. A state $\nu\colon A \longrightarrow [0,1]$ on A is said to satisfy the* ***probabilistic exclusivity principle (PEP)*** *if for any set $S \subseteq A$ of pairwise-exclusive elements, i.e. such that $a \perp b$ for any distinct $a, b \in S$, we have $\sum_{a\in S} \nu(a) \leq 1$.*

A partial Boolean algebra is said to satisfy PEP if all of its states satisfy PEP.

Note that the condition $\sum_{a\in S} \nu(a) \leq 1$ is true of any set S of elements in a Boolean algebra satisfying $a \wedge b = 0$ for distinct $a, b \in S$.

Note that this subsumes the discussion of the PEP at the level of empirical models. If $\mathbf{X}$ is a measurement scenario, the correspondence in Proposition 15 between empirical models on $\mathbf{X}$ and states of $A_{\mathbf{X}}$ restricts to a bijection between empirical models and states satisfying the probabilistic exclusivity principle.

5.3 LEP *vs* PEP

The following result follows immediately from the definitions of partial Boolean algebras and states.

▶ **Proposition 24.** *Let A be a partial Boolean algebra satisfying the logical exclusivity principle. Then, any state on A satisfies the probabilistic exclusivity principle.*

In a general partial Boolean algebra A, however, not all states need satisfy the PEP. A well-known example is the state on the partial Boolean algebra corresponding to a $(4,2,2)$ Bell scenario[6] which corresponds to two (independent) copies of the PR box [11].

However, using the construction from Theorem 1, we can construct from A a new partial Boolean algebra, namely $A[\perp]$, whose states yield states of A that satisfy PEP.

▶ **Theorem 25.** *Let A be a partial Boolean algebra. Then a state $\nu\colon A \longrightarrow [0,1]$ satisfies PEP if there is a state $\hat{\nu}$ of $A[\perp]$ such that*

$$\begin{tikzcd} A \arrow[r, "\eta"] \arrow[dr, "\nu"'] & A[\perp] \arrow[d, "\hat{\nu}"] \\ & {[0,1]} \end{tikzcd}$$

commutes.

Proof. Let $\nu\colon A \longrightarrow [0,1]$ be a state, and suppose it factorises through a state $\hat{\nu}$ of $A[\perp]$. Let $S \subseteq A$ be a set of pairwise exclusive events in A. Then $\{\eta(a) \mid a \in S\}$ is a commeasurable subset of $A[\perp]$, hence it is contained in a Boolean subalgebra B of $A[\perp]$. Since $\hat{\nu}$ must restrict to a finitely-additive probability measure on B, and since $\eta(a) \wedge_{A[\perp]} \eta(b) = 0$ for all distinct $a, b \in S$, we have that

$$\sum_{a\in S} \nu(a) = \sum_{a\in S} \hat{\nu}(\eta(a)) \leq 1 \ . \qquad ◀$$

[6] This stands for a scenario in which there are 4 parties, each of which can choose to perform one of 2 measurements with 2 possible outcomes.

5.4 A reflective adjunction for logical exclusivity

It is not clear whether the partial Boolean algebra $A[\perp]$ necessarily satisfies LEP. While the principle holds for all its elements in the image of $\eta\colon A \longrightarrow A[\perp]$, it may fail to hold for other elements in $A[\perp]$.

However, we can adapt the construction of Theorem 1 to show that one can freely generate, from any given partial Boolean algebra, a new partial Boolean algebra satisfying LEP. This LEP-isation is analogous to e.g. the way one can "abelianise" any group, or use Stone–Čech compactification to form a compact Hausdorff space from any topological space.

▶ **Theorem 26.** *The category* $\mathbf{epBA}$ *is a reflective subcategory of* $\mathbf{pBA}$*, i.e. the inclusion functor* $I\colon \mathbf{epBA} \longrightarrow \mathbf{pBA}$ *has a left adjoint* $X\colon \mathbf{pBA} \longrightarrow \mathbf{epBA}$*. Concretely, for any partial Boolean algebra* A*, there is a partial Boolean algebra* $X(A) = A[\perp]^*$ *which satisfies LEP such that:*

- *there is a* $\mathbf{pBA}$*-morphism* $\eta\colon A \longrightarrow A[\perp]^*$*;*
- *for any* $\mathbf{pBA}$*-morphism* $h\colon A \longrightarrow B$ *where* B *is a partial Boolean algebra* B *satisfying LEP, there is a unique* $\mathbf{pBA}$*-morphism* $\hat{h}\colon A[\perp]^* \longrightarrow B$ *such that* $h = \hat{h} \circ \eta$*, i.e. such that the following diagram commutes:*

$$\begin{tikzcd}
A \arrow[r, "\eta"] \arrow[dr, "h"'] & A[\perp]^* \arrow[d, "\hat{h}"] \\
 & B
\end{tikzcd}$$

The proof of this result follows from a simple adaptation of the proof of Theorem 1, namely adding the following rule to the inductive system presented in Table 1:

$$\frac{u \wedge t \equiv u,\ v \wedge \neg t \equiv v}{u \odot v}$$

This rule will enforce the logical exclusivity principle, and the universal property is proved in a manner similar to the proof of Theorem 1.

6 Tensor products of partial Boolean algebras

6.1 A (first) tensor product by generators and relations

In [25], it is shown that $\mathbf{pBA}$ has a monoidal structure, with $A \otimes B$ given by the colimit of the family of Boolean algebras $C + D$, as C ranges over Boolean subalgebras of A, D ranges over Boolean subalgebras of B, and $+$ denotes the coproduct of Boolean algebras.

The tensor product in [25] is not constructed explicitly: it relies on the existence of coequalisers in $\mathbf{pBA}$, which is proved by an appeal to the Adjoint Functor Theorem.

Our Theorem 1 allows us to give an explicit description of this construction using generators and relations.

▶ **Proposition 27.** *Let* A *and* B *be partial Boolean algebras. Then*

$$A \otimes B \;\cong\; (A \oplus B)[\obar] \ ,$$

where $\obar$ *is the relation on the carrier set of* $A \oplus B$ *given by* $\imath(a) \obar \jmath(b)$ *for all* $a \in A$ *and* $b \in B$*.*

This can be verified by comparing the universal property from Theorem 1 with [25, Proposition 30].

6.2 A more expressive tensor product

There is a lax monoidal functor $\mathsf{P}\colon \mathbf{Hilb} \longrightarrow \mathbf{pBA}$, which takes a Hilbert space to its projectors, viewed as constituting a partial Boolean algebra. The coherence morphisms $\mathsf{P}(\mathcal{H}) \otimes \mathsf{P}(\mathcal{K}) \longrightarrow \mathsf{P}(\mathcal{H} \otimes \mathcal{K})$ are induced by the evident embeddings of $\mathsf{P}(\mathcal{H})$ and $\mathsf{P}(\mathcal{K})$ into $\mathsf{P}(\mathcal{H} \otimes \mathcal{K})$, given by $p \longmapsto p \otimes 1$, $q \longmapsto 1 \otimes q$.

It is easy to see that such morphisms are far from being isomorphisms. For example, if $\mathcal{H} = \mathcal{K} = \mathbb{C}^2$, then there are (many) morphisms from $A = \mathsf{P}(\mathbb{C}^2)$ to $\mathbf{2}$, which lift to morphisms from $A \otimes A$ to $\mathbf{2}$. However, by the Kochen–Specker theorem, there is no such morphism from $\mathsf{P}(\mathbb{C}^4) = \mathsf{P}(\mathbb{C}^2 \otimes \mathbb{C}^2)$.

Interestingly, in [20] it is shown that the images of $\mathsf{P}(\mathcal{H})$ and $\mathsf{P}(\mathcal{K})$, for any finite-dimensional $\mathcal{H}$ and $\mathcal{K}$, generate $\mathsf{P}(\mathcal{H}\otimes\mathcal{K})$. This is used in [20] to justify the claim contradicted by the previous paragraph. The gap in the argument is that more relations hold in $\mathsf{P}(\mathcal{H} \otimes \mathcal{K})$ than in $\mathsf{P}(\mathcal{H}) \otimes \mathsf{P}(\mathcal{K})$. Nevertheless, this result is very suggestive. In standard Boolean algebra theory, these images would satisfy the criteria for $\mathsf{P}(\mathcal{H} \otimes \mathcal{K})$ being the "internal sum" of $\mathsf{P}(\mathcal{H})$ and $\mathsf{P}(\mathcal{K})$ [12]. Evidently, for partial Boolean algebras, these criteria are no longer sufficient. This poses the challenge of finding stronger criteria, and a stronger notion of tensor product to match.

An important property satisfied by the rules in Table 1 as applied in constructing $A \otimes B$ is that, if $t\!\downarrow$ can be derived, then $u\!\downarrow$ can be derived for every subterm u of t. This appears to be too strong a constraint to capture the full logic of the Hilbert space tensor product.

To see why this is an issue, consider projectors $p_1 \otimes p_2$ and $q_1 \otimes q_2$. To ensure in general that they commute, we need the conjunctive requirement that p_1 commutes with q_1 **and** p_2 commutes with q_2. However, to show that they are **orthogonal**, we have a disjunctive requirement: $p_1 \perp q_1$ **or** $p_2 \perp q_2$. If we establish orthogonality in this way, we are entitled to conclude that $p_1 \otimes p_2$ and $q_1 \otimes q_2$ are commeasurable, even though (say) p_2 and q_2 are not. Indeed, the idea that propositions can be defined on quantum systems even though subexpressions are not is emphasised in [20].

This leads us to define a stronger tensor product by forcing logical exclusivity to hold in the tensor product from [25]. This amounts to composing with the reflection to $\mathbf{epBA}$; $\boxtimes := X \circ \otimes$. Explicitly, we define the logical exclusivity tensor product by

$$A \boxtimes B = (A \otimes B)[\perp]^* = (A \oplus B)[\odot][\perp]^*.$$

This is sound for the Hilbert space model. More precisely, P is still a lax monoidal functor with respect to this tensor product. It remains to be seen how close it gets us to the full Hilbert space tensor product.

6.3 Commeasurability extensions, Kochen–Specker, and Hilbert space tensor product

We can ask generally if extending commeasurability by some relation R can induce the Kochen–Specker property in $A[R]$ when it did not hold in A. In fact, it is easily seen that this can never happen.

▶ **Theorem 28** (K–S faithfulness of extensions). *Let A be a partial Boolean algebra, and $R \subseteq A^2$ a relation on A. Then A has the K–S property if and only if $A[R]$ does.*

Proof. If A does not have the K–S property, it has a morphism to a non-trivial Boolean algebra B. By the universal property of $A[R]$, there is a morphism $\hat{h}\colon A[R] \longrightarrow B$. Thus, $A[R]$ does not have the K–S property. Conversely, if there is a morphism $k\colon A[R] \longrightarrow B$ to a non-trivial Boolean algebra B, then $k \circ \eta\colon A \longrightarrow B$, so A does not have the K–S property. ◀

We can apply this in particular to the tensor product.

▶ **Corollary 29.** *If A and B do not have the K–S property, then neither does $(A \otimes B)[\perp]^k$.*

Proof. If A and B do not have the K–S property, they have morphisms to $\mathbf{2}$, and hence so does $A \oplus B$. Applying Theorem 28 inductively $k+1$ times, one concludes that $(A \otimes B)[\perp]^k = (A \oplus B)[\oplus][\perp]^k$ does not have the K–S property. ◀

Under the conjecture that $A[\perp]^*$ coincides with iterating $A[\perp]$ to a fixpoint, this would show that the logical exclusivity tensor product $A \boxtimes B$ never induces a Kochen–Specker paradox if none was already present in A or B.

This can be seen as a limitative result, in the following sense. One of the key points at which non-classicality emerges in quantum theory is the passage from $\mathsf{P}(\mathbb{C}^2)$, which does not have the K–S property, to $\mathsf{P}(\mathbb{C}^4) = \mathsf{P}(\mathbb{C}^2 \otimes \mathbb{C}^2)$, which does.[7] By contrast, it would follow from Corollary 29 that $\mathsf{P}(\mathbb{C}^2) \boxtimes \mathsf{P}(\mathbb{C}^2)$ does not have the K–S property. Therefore, we need a stronger tensor product to track this emergent complexity in the quantum case.

7 Discussion

A number of questions arise from the ideas developed in this paper.

- First, we have shown that LEP implies PEP; that is, if a partial Boolean algebra satisfies Logical Exclusivity, then all its states satisfy Probabilistic Exclusivity. We conjecture that the converse holds.

  ▶ **Conjecture 30.** *PEP $\Rightarrow$ LEP.*
- Similarly, we conjecture the converse to Theorem 25.

  ▶ **Conjecture 31.** *If state ν of a partial Boolean algebra A satisfies PEP, then there is a state $\hat{\nu}$ of $A[\perp]$ such that $\nu = \hat{\nu} \circ \eta$.*

  This would amount to generalising the universality of $A[\perp]$ from **pBA**-morphisms to states. It would yield a one-to-one correspondence between states of A satisfying PEP and states of $A[\perp]$.
- Proving the conjecture above would involve extending a state on a partial Boolean algebra A to a state on $A[\circledcirc]$. A similar operation was achieved for partial Boolean algebras arising from measurement scenarios in Proposition 15, because in that case Definition 13 provided a simple description of the Boolean subalgebras of $A[\circledcirc]$. Is an analogous description possible for the general case considered in Theorem 1, or at least for the particular case of $A[\perp]$?
- A classic result by Greechie [13] constructs a class of orthomodular lattices which admit no states. Since orthomodular lattices are transitive partial Boolean algebras (see e.g. [25]), this means that there are examples of partial Boolean algebras satisfying LEP which admit no states. Is there a partial Boolean algebra not satisfying LEP which admits no states? This would provide a counter-example to Conjecture 30.
- There are some technical questions relating to the $A[\perp]^*$ construction:
  - Is it a completion (i.e. is the reflector a faithful functor)?
  - Is it the same as iterating the $A[\perp]$ construction to a fixpoint?

[7] Note that $\mathsf{P}(\mathbb{C}^2) \cong \bigoplus_{i \in I} \mathbf{4}_i$, where I is a set of the power of the continuum, and each $\mathbf{4}_i$ is the four-element Boolean algebra.

  - Is the relation of $A[\bot]^*$ to $A[\bot]$ an instance of a more general relationship between iterating an inductive construction, and adding a rule to the inductive construction itself?
- Our discussion of tensor products led us to introduce a strong tensor product of partial Boolean algebras, $A \boxtimes B$. This brings us closer to an answer to the following particularly interesting question:

  ▶ **Question 32.** Is there a monoidal structure $\circledast$ on the category **pBA** such that the functor $\mathsf{P}\colon \mathbf{Hilb} \longrightarrow \mathbf{pBA}$ is **strong monoidal** with respect to this structure, i.e. such that $\mathsf{P}(\mathcal{H}) \circledast \mathsf{P}(\mathcal{K}) \cong \mathsf{P}(\mathcal{H} \otimes \mathcal{K})$?

  A positive answer to this question would offer a complete logical characterisation of the Hilbert space tensor product, and provide an important step towards giving logical foundations for quantum theory in a form useful for quantum information and computation.
- We recall the following quotation from Ernst Specker given in [8]:

  > Do you know what, according to me, is the fundamental theorem of quantum mechanics? ... That is, if you have several questions and you can answer any two of them, then you can also answer all three of them. This seems to me very fundamental.

  This refers to the **binarity** of compatibility in quantum mechanics. A set of observables is compatible if they are pairwise so. This is built into the definition of partial Boolean algebras, and it is why we only considered graphical measurement scenarios in this paper. However, in the general theory of contextuality, as developed e.g. in [5], more general forms of compatibility are considered, represented by simplicial complexes. The notion of partial Boolean algebras in a broader sense introduced in [10] seems suitable to deal with this more general format. How much of the theory carries over?
- Partial Boolean algebras capture logical structure. We have seen how this logical structure can be used to enforce strong constraints on the probabilistic behaviour of states. This is somewhat analogous to the role of possibilistic empirical models in [5]. Can we lift the concepts and results relating to possibilistic empirical models in [5, 4, 1] to the level of partial Boolean algebras?
- There is much more to be said regarding contextuality in this setting. In current work in progress, we are considering the following topics:
  - A hierarchy of logical contextuality properties generalising those studied in [5].
  - A systematic treatment of "Kochen–Specker paradoxes", i.e. contradictory statements which can be validated in partial Boolean algebras.
  - Constructions that transform state-dependent to state-independent forms of contextuality.

References

1 Samson Abramsky. Relational hidden variables and non-locality. *Studia Logica*, 101(2):411–452, 2013. In Juha Kontinen, Jouko Väänänen, and Dag Westerståhl, editors, special issue on *Dependence and Independence in Logic*. doi:10.1007/s11225-013-9477-4.

2 Samson Abramsky. Contextuality: At the borders of paradox. In Elaine Landry, editor, *Categories for the Working Philosopher*. Oxford University Press, 2017. doi:10.1093/oso/9780198748991.003.0011.

3 Samson Abramsky, Rui Soares Barbosa, Kohei Kishida, Raymond Lal, and Shane Mansfield. Contextuality, cohomology and paradox. In Stephan Kreutzer, editor, *Proceedings of 24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *Leibniz*

International Proceedings in Informatics (LIPIcs), pages 211–228. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.211.

4 Samson Abramsky, Rui Soares Barbosa, Kohei Kishida, Raymond Lal, and Shane Mansfield. Possibilities determine the combinatorial structure of probability polytopes. *Journal of Mathematical Psychology*, 74:58–65, 2016. In Ehtibar N. Dzhafarov, Janne V. Kujala, and Reinhard Suck, editors, special issue on *Foundations of Probability Theory in Psychology and Beyond*. doi:10.1016/j.jmp.2016.03.006.

5 Samson Abramsky and Adam Brandenburger. The sheaf-theoretic structure of non-locality and contextuality. *New Journal of Physics*, 13(11):113036, 2011. doi:10.1088/1367-2630/13/11/113036.

6 Barbara Amaral, Marcelo Terra Cunha, and Adán Cabello. Exclusivity principle forbids sets of correlations larger than the quantum set. *Physical Review A*, 89(3):030101, 2014. doi:10.1103/PhysRevA.89.030101.

7 Garrett Birkhoff and John von Neumann. The logic of quantum mechanics. *Annals of Mathematics*, 37(4):823–843, 1936. doi:10.2307/1968621.

8 Adán Cabello. Specker's fundamental principle of quantum mechanics. arXiv preprint arXiv:1212.1756 [quant-ph], 2012.

9 Adán Cabello, Simone Severini, and Andreas Winter. Graph-theoretic approach to quantum correlations. *Physical Review Letters*, 112(4):040401, 2014. doi:10.1103/PhysRevLett.112.040401.

10 Janusz Czelakowski. Partial boolean algebras in a broader sense. *Studia Logica*, 38(1):1–16, 1979. doi:10.1007/BF00493669.

11 Tobias Fritz, Ana Belén Sainz, Remigiusz Augusiak, Jonatan Bohr Brask, Rafael Chaves, Anthony Leverrier, and Antonio Acín. Local orthogonality as a multipartite principle for quantum correlations. *Nature Communications*, 4:2263, 2013. doi:10.1038/ncomms3263.

12 Steven Givant and Paul Halmos. *Introduction to Boolean algebras*. Undergraduate Texts in Mathematics. Springer-Verlag New York, 2009. doi:10.1007/978-0-387-68436-9.

13 Richard J. Greechie. Orthomodular lattices admitting no states. *Journal of Combinatorial Theory, Series A*, 10(2):119–132, 1971. doi:10.1016/0097-3165(71)90015-X.

14 Stanley P. Gudder. Partial algebraic structures associated with orthomodular posets. *Pacific Journal of Mathematics*, 41(3):717–730, 1972. doi:10.2140/pjm.1972.41.717.

15 Joe Henson. Quantum contextuality from a simple principle? arXiV preprint arXiv:1210.5978 [quant-ph], 2012.

16 Richard I. G. Hughes. *The structure and interpretation of quantum mechanics*. Harvard University Press, 1989.

17 Chris J. Isham and Jeremy Butterfield. Topos perspective on the Kochen–Specker theorem: I. Quantum states as generalized valuations. *International Journal of Theoretical Physics*, 37(11):2669–2733, 1998.

18 Peter T. Johnstone. *Topos theory*, volume 10 of *London Mathematical Society Monographs*. Academic Press, 1997.

19 Gudrun Kalmbach. *Orthomodular lattices*, volume 18 of *London Mathematical Society Monographs*. Academic Press, 1983.

20 Simon Kochen. A reconstruction of quantum mechanics. *Foundations of Physics*, 45(5):557–590, 2015. doi:10.1007/s10701-015-9886-5.

21 Simon Kochen and Ernst P. Specker. The problem of hidden variables in quantum mechanics. *Journal of Mathematics and Mechanics*, 17(1):59–87, 1967.

22 Patricia F. Lock and Gary M. Hardegree. Connections among quantum logics. Part 1. Quantum propositional logics. *International Journal of Theoretical Physics*, 24(1):43–53, 1985. doi:10.1007/BF00670072.

23 Sandu Popescu and Daniel Rohrlich. Quantum nonlocality as an axiom. *Foundations of Physics*, 24(3):379–385, 1994. doi:10.1007/BF02058098.

24 Ravishankar Ramanathan, Akihito Soeda, Paweł Kurzyński, and Dagomir Kaszlikowski. Generalized monogamy of contextual inequalities from the no-disturbance principle. *Physical Review Letters*, 109(5):050404, 2012. `doi:10.1103/PhysRevLett.109.050404`.

25 Benno van den Berg and Chris Heunen. Noncommutativity as a colimit. *Applied Categorical Structures*, 20(4):393–414, 2012. `doi:10.1007/s10485-011-9246-3`.

Factorize Factorization

Beniamino Accattoli
Inria & LIX, École Polytechnique, UMR 7161, Palaiseau, France

Claudia Faggian
Université de Paris, IRIF, CNRS, F-75013 Paris, France

Giulio Guerrieri
University of Bath, Department of Computer Science, Bath, UK

Abstract

We present a new technique for proving factorization theorems for compound rewriting systems in a modular way, which is inspired by the Hindley-Rosen technique for confluence. Specifically, our approach is well adapted to deal with extensions of the call-by-name and call-by-value λ-calculi.

The technique is first developed abstractly. We isolate a sufficient condition (called linear swap) for lifting factorization from components to the compound system, and which is compatible with β-reduction. We then closely analyze some common factorization schemas for the λ-calculus.

Concretely, we apply our technique to diverse extensions of the λ-calculus, among which de' Liguoro and Piperno's non-deterministic λ-calculus and – for call-by-value – Carraro and Guerrieri's shuffling calculus. For both calculi the literature contains factorization theorems. In both cases, we give a new proof which is neat, simpler than the original, and strikingly shorter.

2012 ACM Subject Classification Theory of computation → Rewrite systems; Theory of computation → Lambda calculus; Theory of computation → Logic

Keywords and phrases Lambda Calculus, Rewriting, Reduction Strategies, Factorization

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.6

Funding This work is partially supported by ANR JCJC grant "COCA HOLA" (ANR-16-CE40-004-01), ANR PRC project PPS (ANR-19-CE48-0014), and EPSRC Project EP/R029121/1 *Typed lambda-calculi with sharing and unsharing.*

1 Introduction

The λ-calculus underlies functional programming languages and, more generally, the paradigm of higher-order computation. Through the years, more and more advanced features have enriched this paradigm, including control, non-determinism, states, probabilistic or quantum features. The well established way to proceed is to extend the λ-calculus with new operators. Every time, good operational properties, such as confluence, normalization, or termination, need to be proved. It is evident that the more complex and advanced is the calculus under study, the more the ability to *modularize the analyses* of its properties is crucial.

Techniques for modular proofs are available for termination and confluence, with a rich literature which examines under which conditions these properties lift from modules to the compound system – some representative papers are [56, 55, 57, 51, 40, 41, 42, 32, 10, 20, 18, 17, 5, 12], see Gramlich [22] for a survey. Termination and confluence concern the existence and the uniqueness of normal forms, which are the results of a computation. When the focus is on *how to compute* the result, that is, on identifying reduction strategies with good properties, then only few abstract techniques are currently available (we mention [21, 38, 39], [54, Ch. 8], and [2]) – this paper proposes a new one.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 6; pp. 6:1–6:25
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Factorization. The most basic property about how to compute is *factorization*, whose paradigmatic example is the head factorization theorem of the λ-calculus (Theorem 11.4.6 in Barendregt's book [11]): every β-reduction sequence $t \to_\beta^* u$ can be re-organized/factorized so as to first reducing head redexes and then everything else – in symbols $t \underset{h}{\to}^* \cdot \underset{\neg h}{\to}^* u$.

The study of factorization in λ-calculus goes back to Rosser [50]. Factorization results are sometimes referred to as *semi-standardization* [43], or *postponement* [54], and often simply called *standardization* – standardization is however a more sophisticated property (sketched below) of which factorization is a basic instance. Here, we adopt Melliès terminology [39].

According to Melliès [39], the meaning of factorization is that the *essential* part of a computation can always be separated from its junk. Let's abstract the role of head reduction, by assuming that computations consist of steps $\underset{e}{\to}$ which are in some sense *essential*, and steps $\underset{i}{\to}$ which are not. Factorization says that every rewrite sequence $t \to^* s$ can be factorized as $t \underset{e}{\to}^* u \underset{i}{\to}^* s$, *i.e.*, as a sequence of essential steps followed by inessential ones.

Well known examples of essential reductions are head and leftmost-outermost reduction for the λ-calculus (see Barendregt [11]), or left and weak reduction for the call-by-value λ-calculus (see Plotkin [46] and Paolini and Ronchi Della Rocca [49]).

Very much as confluence, factorization for λ-calculi requires non-trivial proof techniques such as finite developments [15, 54], labeling [37, 31], or parallel reduction [53].

Uses of Factorization. Factorization is commonly used as a *building block* in proving more sophisticated properties of the *how-to-compute* kind. It is often the main ingredient in proofs of *normalization* theorems [11, 53, 29, 3], stating that a reduction strategy reaches a normal form whenever one exists. Leftmost-outermost normalization is a well known example.

Another property, *standardization*, generalizes factorization: reduction sequences can be organized with respect to an order on redexes, not just with respect to the distinction essential/inessential. It is an early result that factorization can be used to prove standardization: iterated head factorizations provides what is probably the simplest way to prove Curry and Feys' left-to-right standardization theorem, via Mitschke's argument [43].

Additionally, the independence of some computational tasks, such as garbage collection, is often modeled as a factorization theorem.

Contributions of this Paper. In this paper we propose a technique for proving in a *modular* way factorization theorems for *compound higher-order systems,* such as those obtained by extending the λ-calculus with advanced features. The approach can be seen as an analogous for factorization of the classical technique for confluence based on Hindley-Rosen lemma, which we discuss in the next paragraphs. Mimicking the use of Hindley-Rosen lemma is natural, yet to our knowledge such an approach has never been used before, at least not in the λ-calculus literature. Perhaps this is because a direct transposition of Hindley-Rosen technique does not work with β reduction, as we discuss below and in Sect. 3.

After developing a sharper technique, we apply it to various known extensions of the λ-calculus which do not fit into easily manageable categories of rewriting systems. In all our case studies, our novel proofs are neat, concise, and simpler than the originals.

Confluence via Hindley-Rosen. The simplest *modular* technique to establish confluence is based on *Hindley-Rosen lemma* [11, Prop. 3.3.5], which states that the *union* of two confluent reductions $\to_1$ and $\to_2$ is confluent if $\to_1$ and $\to_2$ satisfy a commutation property. This is the technique used in Barendregt's book for proving confluence of $\to_{\beta\eta}$ [11, Thm. 3.3.9], where it is also stressed that the proof is simpler than Curry and Feys' original one [15].

While the result is basic, Hindley-Rosen technique provides a powerful tool to prove confluence of compound systems. In the literature of the λ-calculus, we mention for instance its use in the linear-algebraic λ-calculus [7], the probabilistic λ-calculus [19], the $\Lambda\mu$-calculus [52], the shuffling calculus [14], the λ-calculus extended with lists [48] or pattern-matching [13], or with a `let` construct [6]. It is worth to spell out the gain. Confluence is often a non-trivial property to establish – when higher-order is involved, the proof of confluence requires sophisticated techniques. The difficulty *compounds* when extending the λ-calculus with new constructs. Still, the problem is often originated by β reduction itself, which encapsulates the higher-order features of the computation. By using Hindley-Rosen lemma, confluence of β is used as a *black box*: one *relies* on that – without having to prove it again – to show that the extended calculus is confluent.

Hindley-Rosen and Sufficient Conditions. There is a subtle distinction between Hindley-Rosen *lemma*, and what we refer to as Hindley-Rosen *technique*. Hindley-Rosen lemma reduces confluence of a compound system to commutation of the components – the modules. To establish commutation, however, is a non-trivial task, because it is a *global* property, that is, it quantifies over *all sequences* of steps. The success of the lemma in the λ-calculus literature stems from the existence of easy-to-check conditions which suffice to prove commutation. All the examples mentioned above indeed satisfy Hindley's *strong commutation* property [26] (Lemma 3.3.6 in [11]), where at most one reduction – but not both – may require multiple steps to close a diagram, commutation then follows by a finitary tiling argument. Strong commutation turns Hindley-Rosen lemma into an effective, concrete proof technique.

Modular Factorization, Abstractly. Here, we present a modular approach to factorization inspired by the Hindley-Rosen *technique*. A formulation of Hindley-Rosen lemma for factorization is immediate, and is indeed folklore. But exactly as for confluence, this reduces factorization of a compound system to a property that is difficult to establish, without a real gain. The crucial point is finding suitable conditions that can be used in practice. The *issue* here is that the natural adaptation of strong commutation to factorization is – in general – *not* verified by extensions of the λ-calculi, as it does not interact well with β (see Ex. 3.2 in Sect. 3). We identify an alternative condition – called *linear swap* – which is satisfied by a large variety of interesting examples, turning the approach into an effective, concrete *technique*. Testing the linear swap condition is easy and combinatorial in nature, as it is a *local* property, in the sense that only single steps (rather than sequences of steps) need to be manipulated. This holds true even when the modules are not confluent, or non-terminating. The other key point in our approach is that we *assume* the modules to be factorizing, therefore we can use their factorization – that may require non-trivial proof techniques such as parallel reductions or finite developments – as a black box.

Modular Factorization, Concretely. We then focus on our target, how to establish factorization results for extensions of the λ-calculus. Concretely, we start from β reduction, or its call-by-value counterpart β_v, and allow the calculus to be enriched with extra rules. Here we discover a further striking gain: for common factorization schemas such as head or weak factorization, verifying the required linear swap conditions reduces to checking *a single case*, together with the fact that the new rule behaves well with respect to substitution. The test for modular factorization that we obtain is a ready-to-use and easy recipe that can be applied in a variety of cases.

We illustrate our technique by providing several examples, chosen to stress the independence of the technique from other rewriting properties. In particular, we give a *new* and arguably *simpler* proof of two results from the literature. The first is head factorization for the non-deterministic λ-calculus by de' Liguoro and Piperno [16], which extends the λ-calculus with a choice operator $\oplus$. It is a *non-confluent* calculus, and it is representative of the class of λ-calculi extended with a commutative effect – such as *probabilistic* choice – of which it presents most features and all issues, see [34] for a thorough discussion.

The second is a new, simplified proof of factorization for the shuffling calculus – a refinement of the call-by-value λ-calculus due to Carraro and Guerrieri [14], whose left factorization is proved by Guerrieri, Paolini, and Ronchi della Rocca in [25]. In this case the λ-calculus is extended with extra rules but they are not associated with a new operator. The resulting calculus is subtle, as it has critical pairs.

In both cases, the new proof is neat, conceptually clear, and strikingly short. The reason why our proofs are only a few lines long, whereas the originals require several pages, is exactly that there is no need to "prove again" factorization of β or β_v. We just show that β or β_v interacts well with the new rules.

Further Applications: Probabilistic λ-calculi. The investigation in this paper was triggered by concrete needs, namely the study of strategies for probabilistic λ-calculi [19, 36]. The probabilistic structure adds complexity, and indeed makes the study of factorization painful – exposing the need for tools to make such an analysis more manageable. Our technique smoothly applies, providing new concise proofs that are significantly simpler than the originals – indeed surprisingly simple. These results are however only overviewed in this paper: we sketch the application to the call-by-value probabilistic calculus by Faggian and Ronchi della Rocca [19], leaving the technical details in Appendix B. The reason is that, while the application of our technique is simple, the *syntax* of probabilistic λ-calculi is not – because reduction is defined on (monadic) structures representing probability distributions over terms. Aiming at making the paper accessible within the space limits, we prefer to focus on examples in a syntax which is familiar to a wide audience. Indeed, once the technique is understood, its application to other settings is immediate, and in large part automatic.

A Final Remark. Like Hindley-Rosen for confluence, our technique is sufficient but not necessary to factorization. Still, its features and wide range of application make it a remarkable tool to tame the complexity that is often encountered in the analysis of advanced compound calculi. By emphasizing the benefits of a modular approach to factorization, we hope to prompt the development of even more techniques.

Related work. To our knowledge, the only result in the literature about modular techniques for factorization is Accattoli's technique for calculi with explicit substitutions [2], which relies on termination hypotheses. Our *linear swap* condition (page 8) is technically the same as his *diagonal-swap* condition. One of the insights at the inception of this work is exactly that termination in [2] is used only to establish factorization of each single module, but not when combining them. Here we *assume* modules to be factorizing, therefore avoiding termination requirements, and obtaining a more widely applicable technique.

Van Oostrom's decreasing diagrams technique [59] is a powerful and *inherently modular* tool to establish confluence and commutation. Surprisingly, it has not yet been used for factorization, but steps in this direction have been presented recently [58].

A divide-and-conquer approach is well-studied for termination. The key point is finding conditions which guarantee that the union of terminating relations is terminating. Several have been studied [10, 20]. The weakest such condition, namely $\to_2 \cdot \to_1 \subseteq \to_2 \cup (\to_1 \cdot (\to_1 \cup \to_2)^*)$, is introduced by Doornbos and von Karger [18], and then studied by Dershowitz [17], under the name of *lazy commutation*, and by van Oostrom and Zantema [61]. Interestingly, lazy commutation is similar to the linear swap condition.

Another approach to study extensions of a rewriting system is isolating syntactical conditions that induce rewriting properties – for instance orthogonality of the rewriting rules induces confluence, see Terese [54, Ch. 10.4]. Factorization and standardization are also investigated, in particular for left-to-right standardization [54, Ch. 8.5.7].

2 Preliminaries

In this section we recall some standard definitions and notations in rewriting theory (see for instance Terese [54] or Baader and Nipkow [9]), and then provide an overview of commutation, confluence, and factorization. Both *confluence* and *factorization* are forms of commutation.

Basics. An *abstract rewriting system (ARS)* is a pair $\mathcal{A} = (A, \to)$ consisting of a set A and a binary relation $\to$ on A whose pairs are written $t \to s$ and called *steps*. We denote $\to^*$ (resp. $\to^=$) the transitive-reflexive (resp. reflexive) closure of $\to$, and use $\leftarrow$ for the reverse relation of $\to$, that is, $u \leftarrow t$ if $t \to u$. If $\to_1, \to_2$ are binary relations on A then $\to_1 \cdot \to_2$ denotes their composition, *i.e.* $t \to_1 \cdot \to_2 s$ if there exists $u \in A$ such that $t \to_1 u \to_2 s$. We write $(A, \{\to_1, \to_2\})$ to denote the ARS $(A, \to)$ where $\to \; = \; \to_1 \cup \to_2$. We freely use the fact that the transitive-reflexive closure of a relation is a closure operator, that is, it satisfies

$$\to \subseteq \to^*, \qquad (\to^*)^* \; = \to^*, \qquad \to_1 \subseteq \to_2 \text{ implies } \to_1^* \subseteq \to_2^* \,. \qquad \textbf{(Closure)}$$

The following property is an immediate consequence:

$$(\to_1 \cup \to_2)^* \; = \; (\to_1^* \cup \to_2^*)^*. \qquad \textbf{(TR)}$$

Local vs Global Properties. An important distinction in rewriting theory is between local and global properties. A property of term t is *local* if it is quantified over only *one-step reductions* from t, while it is *global* if it is quantified over all *rewrite sequences* from t. Local properties are easier to test, because the analysis (usually) involves a finite number of cases.

Commutation. Two relations $\to_1$ and $\to_2$ on A *commute* if ${\leftarrow_1}^* \cdot {\to_2}^* \subseteq {\to_2}^* \cdot {\leftarrow_1}^*$.

Confluence. A relation $\to$ on A is confluent if it commutes with itself. A classic tool to modularize the proof of confluence is Hindley-Rosen lemma. Confluence of two relations $\to_1$ and $\to_2$ does not imply confluence of $\to_1 \cup \to_2$, however it does if they commute.

▶ **Lemma** (Hindley-Rosen). *Let $\to_1$ and $\to_2$ be relations on the set A. If $\to_1$ and $\to_2$ are confluent and commute with each other, then $\to_1 \cup \to_2$ is confluent.*

Easy-to-Check Conditions for Hindley Rosen. Commutation is a global condition, which is difficult to test. What turns Hindley-Rosen lemma into an effective, usable *technique*, is the availability of local, *easy-to-check* sufficient conditions. One of the simplest but most useful such conditions is Hindley's strong commutation [26]:

$$\leftarrow_1 \cdot \to_2 \subseteq \; {\to_2}^* \cdot {\leftarrow_1}^= \qquad \textbf{(Strong Commutation)}$$

▶ **Lemma 2.1** (Local test for commutation [26]). *Strong commutation implies commutation.*

All the extensions of λ-calculus we cited at pages 2–3 (namely [11, 7, 19, 52, 14, 48, 13, 6]) prove confluence by using Hindley-Rosen lemma via strong commutation (possibly in its weaker diamond-like form $\leftarrow_1 \cdot \rightarrow_2 \subseteq \rightarrow_2^= \cdot \leftarrow_1^=$).

Factorization. We now recall definitions and basic facts on the rewriting property at the center of this paper, factorization. Let $\mathcal{A} = (A, \{\underset{e}{\to}, \underset{i}{\to}\})$ be an ARS.

- The relation $\to = \underset{e}{\to} \cup \underset{i}{\to}$ satisfies **e-factorization**, written $\mathtt{Fact}(\underset{e}{\to}, \underset{i}{\to})$, if

$$\mathtt{Fact}(\underset{e}{\to}, \underset{i}{\to}): \quad (\underset{e}{\to} \cup \underset{i}{\to})^* \subseteq \underset{e}{\to}^* \cdot \underset{i}{\to}^* \qquad (\textbf{Factorization})$$

- The relation $\underset{i}{\to}$ **postpones** after $\underset{e}{\to}$, written $\mathtt{PP}(\underset{e}{\to}, \underset{i}{\to})$, if

$$\mathtt{PP}(\underset{e}{\to}, \underset{i}{\to}): \quad \underset{i}{\to}^* \cdot \underset{e}{\to}^* \subseteq \underset{e}{\to}^* \cdot \underset{i}{\to}^*. \qquad (\textbf{Postponement})$$

Postponement can be formulated in terms of commutation, and vice versa, since clearly ($\underset{i}{\to}$ postpones after $\underset{e}{\to}$) if and only if ($\underset{i}{\leftarrow}$ commutes with $\underset{e}{\to}$). Note that reversing $\underset{i}{\to}$ introduces an asymmetry between the two relations. It is an easy result that e-factorization is equivalent to postponement, which is a more convenient way to express it. The following equivalences – which we shall use freely – are all well known.

▶ **Lemma 2.2.** *For any two relations $\underset{e}{\to}, \underset{i}{\to}$ the following statements are equivalent:*

1. Semi-local postponement*: $\underset{i}{\to}^* \cdot \underset{e}{\to} \subseteq \underset{e}{\to}^* \cdot \underset{i}{\to}^*$ (and its dual $\underset{i}{\to} \cdot \underset{e}{\to}^* \subseteq \underset{e}{\to}^* \cdot \underset{i}{\to}^*$).*
2. Postponement*: $\mathtt{PP}(\underset{e}{\to}, \underset{i}{\to})$.*
3. Factorization*: $\mathtt{Fact}(\underset{e}{\to}, \underset{i}{\to})$.*

Another property that we shall use freely is the following, which is immediate by the definition of postponement and property **TR** (page 5).

▶ **Lemma 2.3.** *Given a relation $\underset{i}{\circ\!\!\!\to}$ such that $\underset{i}{\circ\!\!\!\to}^* = \underset{i}{\to}^*$, $\mathtt{PP}(\underset{e}{\to}, \underset{i}{\to})$ if and only if $\mathtt{PP}(\underset{e}{\to}, \underset{i}{\circ\!\!\!\to})$.*

A well-known use of the above is to instantiate $\underset{i}{\circ\!\!\!\to}$ with a notion of parallel reduction [53].

Easy-to-Check Sufficient Condition for Postponement. Hindley first noted that a local property implies postponement, hence factorization [26]. It is immediate to recognize that the property below is exactly the postponement analog of strong commutation in Lemma 2.1 (it is the same expression, with $\underset{i}{\to} := \leftarrow_1$ and $\underset{e}{\to} := \rightarrow_2$).

We say that $\underset{i}{\to}$ **strongly postpones** after $\underset{e}{\to}$, if

$$\mathtt{SP}(\underset{e}{\to}, \underset{i}{\to}): \quad \underset{i}{\to} \cdot \underset{e}{\to} \subseteq \underset{e}{\to}^* \cdot \underset{i}{\to}^= \qquad (\textbf{Strong Postponement})$$

▶ **Lemma 2.4** (Local test for postponement [26]). *Strong postponement implies postponement: $\mathtt{SP}(\underset{e}{\to}, \underset{i}{\to})$ implies $\mathtt{PP}(\underset{e}{\to}, \underset{i}{\to})$, and so $\mathtt{Fact}(\underset{e}{\to}, \underset{i}{\to})$.*

Strong postponement is at the heart of several factorization proofs. However (similarly to the diamond property for confluence) it can rarely be used *directly*, because most interesting relations – *e.g.* β reduction in λ-calculus – do not satisfy it. Still, its range of application hugely widens by using Lemma 2.3.

It is instructive to examine strong postponement with respect to β reduction, as it allows us to also recall why it is difficult to establish head factorization for the λ-calculus.

▶ **Example 2.5** (λ-calculus and strong postponement)**.** In view of head factorization, the β reduction is decomposed in head reduction $\underset{\mathsf{h}}{\to}_{\beta}$ and its dual $\underset{\neg\mathsf{h}}{\to}_{\beta}$, that is $\to_{\beta} = \underset{\mathsf{h}}{\to}_{\beta} \cup \underset{\neg\mathsf{h}}{\to}_{\beta}$. To prove head factorization is not trivial precisely because $\mathtt{SP}(\underset{\mathsf{h}}{\to}_{\beta}, \underset{\neg\mathsf{h}}{\to}_{\beta})$ *does not* hold.

Consider the following example: $(\lambda x.xxx)(Iz) \underset{\neg\mathsf{h}}{\to}_{\beta} (\lambda x.xxx)z \underset{\mathsf{h}}{\to}_{\beta} zzz$. This sequence $\underset{\neg\mathsf{h}}{\to} \cdot \underset{\mathsf{h}}{\to}$ can only postpone to a reduction sequence of the form $\underset{\mathsf{h}}{\to} \cdot \underset{\mathsf{h}}{\to} \cdot \underset{\neg\mathsf{h}}{\to} \cdot \underset{\neg\mathsf{h}}{\to}$:

$$(\lambda x.xxx)(Iz) \underset{\mathsf{h}}{\to}_{\beta} (Iz)(Iz)(Iz) \underset{\mathsf{h}}{\to}_{\beta} z(Iz)(Iz) \underset{\neg\mathsf{h}}{\to}_{\beta} zz(Iz) \underset{\neg\mathsf{h}}{\to}_{\beta} zzz.$$

A solution is to compress sequences of $\underset{\neg\mathsf{h}}{\to}$ by introducing an intermediate relation $\underset{\neg\mathsf{h}}{\Rightarrow}$ (*internal parallel reduction*) such that $\underset{\neg\mathsf{h}}{\Rightarrow}^{*} = \underset{\neg\mathsf{h}}{\to}_{\beta}^{*}$ and which does verify strong postponement. This is indeed the core of Takahashi's technique [53]. All the work in [53] goes into defining parallel reductions, and proving $\mathtt{SP}(\underset{\mathsf{h}}{\to}_{\beta}, \underset{\neg\mathsf{h}}{\Rightarrow})$. One indeed has $\underset{\neg\mathsf{h}}{\Rightarrow} \cdot \underset{\mathsf{h}}{\to}_{\beta} \subseteq \underset{\mathsf{h}}{\to}_{\beta} \cdot \underset{\mathsf{h}}{\to}_{\beta}^{*} \cdot \underset{\neg\mathsf{h}}{\Rightarrow}$.

3 Modularizing Factorization

All along this section, we assume to have two relations $\to_{\alpha}, \to_{\gamma}$ on the same set A, such that

$$\to_{\alpha} = \underset{\mathsf{e}}{\to}_{\alpha} \cup \underset{\mathsf{i}}{\to}_{\alpha} \quad \text{and} \quad \to_{\gamma} = \underset{\mathsf{e}}{\to}_{\gamma} \cup \underset{\mathsf{i}}{\to}_{\gamma} .$$

We define $\underset{\mathsf{i}}{\to} := (\underset{\mathsf{i}}{\to}_{\alpha} \cup \underset{\mathsf{i}}{\to}_{\gamma})$ and $\underset{\mathsf{e}}{\to} := (\underset{\mathsf{e}}{\to}_{\alpha} \cup \underset{\mathsf{e}}{\to}_{\gamma})$. Clearly $\to_{\alpha} \cup \to_{\gamma} = \underset{\mathsf{i}}{\to} \cup \underset{\mathsf{e}}{\to}$. Our goal is obtaining a technique in the style of Hindley-Rosen's for confluence, to establish that if $\to_{\alpha}, \to_{\gamma}$ are e-factorizing then their union also is, that is, $\mathtt{Fact}(\underset{\mathsf{e}}{\to}, \underset{\mathsf{i}}{\to})$ holds.

Issues. In spite of the large and fruitful use in the λ-calculus literature of Hindley-Rosen technique to simplify the analysis of confluence, we are not aware of any similar technique in the analysis of factorization. In this section we explain why a transposition of the technique is not immediate when β reduction is involved.

A direct equivalent of Hindley-Rosen lemma for commutation is folklore. An explicit proof is in [59]. Formulated in terms of postponement we obtain the following statement.

▶ **Lemma 3.1** (Hindley-Rosen transposed to factorization)**.** *Assume that* $\to_{\alpha}$ *and* $\to_{\gamma}$ *are* e*-factorizing relations. Their union* $\to_{\alpha} \cup \to_{\gamma}$ *satisfies* $\mathtt{Fact}(\underset{\mathsf{e}}{\to}, \underset{\mathsf{i}}{\to})$ *if*

$$\mathtt{PP}(\underset{\mathsf{e}}{\to}_{\gamma}, \underset{\mathsf{i}}{\to}_{\alpha}) : \underset{\mathsf{i}}{\to}_{\alpha}^{*} \cdot \underset{\mathsf{e}}{\to}_{\gamma}^{*} \subseteq \underset{\mathsf{e}}{\to}_{\gamma}^{*} \cdot \underset{\mathsf{i}}{\to}_{\alpha}^{*} \quad \textit{and} \quad \mathtt{PP}(\underset{\mathsf{e}}{\to}_{\alpha}, \underset{\mathsf{i}}{\to}_{\gamma}) : \underset{\mathsf{i}}{\to}_{\gamma}^{*} \cdot \underset{\mathsf{e}}{\to}_{\alpha}^{*} \subseteq \underset{\mathsf{e}}{\to}_{\alpha}^{*} \cdot \underset{\mathsf{i}}{\to}_{\gamma}^{*} . \ (\#)$$

Exactly as Hindley-Rosen lemma, the modularization lemma above is of no practical use by itself, as the pair of conditions (#) one has to test are as global as the original problem. What we need is to have local conditions (akin to strong commutation) to turn the lemma into a usable technique. One obvious choice is *strong postponement*:

$$\mathtt{SP}(\underset{\mathsf{e}}{\to}_{\gamma}, \underset{\mathsf{i}}{\to}_{\alpha}) : \underset{\mathsf{i}}{\to}_{\alpha} \cdot \underset{\mathsf{e}}{\to}_{\gamma} \subseteq \underset{\mathsf{e}}{\to}_{\gamma}^{*} \cdot \underset{\mathsf{i}}{\to}_{\alpha}^{=} \quad \text{and} \quad \mathtt{SP}(\underset{\mathsf{e}}{\to}_{\alpha}, \underset{\mathsf{i}}{\to}_{\gamma}) : \underset{\mathsf{i}}{\to}_{\gamma} \cdot \underset{\mathsf{e}}{\to}_{\alpha} \subseteq \underset{\mathsf{e}}{\to}_{\alpha}^{*} \cdot \underset{\mathsf{i}}{\to}_{\gamma}^{=} . \ (\#\#)$$

Clearly, ## implies # (Lemma 2.4). We may hope to have all the elements for a postponement analog of Hindley-Rosen technique, but it is not the case. Unfortunately, conditions ## usually *do not hold* in extensions of the λ-calculus. Let us illustrate the issue with an example, the non-deterministic λ-calculus, that we shall develop formally in Sect. 5.

▶ **Example 3.2** (Issues)**.** Consider the extension of the language of λ-terms with a construct $\oplus$ which models non-deterministic choice. The term $\oplus pq$ non-deterministically reduces to either p or q, that is, $\oplus pq \to_{\oplus} p$ and $\oplus pq \to_{\oplus} q$. The calculus $(\Lambda, \to_{\beta} \cup \to_{\oplus})$ has two reduction rules, $\to_{\beta}$ and $\to_{\oplus}$. For both, we define head and non-head steps as usual.

Consider the following sequence: $(\lambda x.xxx)(\oplus pq) \underset{\neg\mathsf{h}}{\to}_{\oplus} (\lambda x.xxx)p \underset{\mathsf{h}}{\to}_{\beta} ppp$. This sequence $\underset{\neg\mathsf{h}}{\to}_{\oplus} \cdot \underset{\mathsf{h}}{\to}_{\beta}$ can only postpone to a reduction sequence of the form $\underset{\mathsf{h}}{\to}_{\beta} \cdot \underset{\mathsf{h}}{\to}_{\oplus} \cdot \underset{\neg\mathsf{h}}{\to}_{\oplus} \cdot \underset{\neg\mathsf{h}}{\to}_{\oplus}$:

$$(\lambda x.xxx)(\oplus pq) \underset{\mathsf{h}}{\to}_{\beta} (\oplus pq)(\oplus pq)(\oplus pq) \underset{\mathsf{h}}{\to}_{\oplus} p(\oplus pq)(\oplus pq) \underset{\neg\mathsf{h}}{\to}_{\oplus} pp(\oplus pq) \underset{\neg\mathsf{h}}{\to}_{\oplus} ppp.$$

As the β-step duplicates the redex $\oplus pq$, the condition $\mathtt{SP}(\underset{\mathsf{h}}{\to}_{\beta}, \underset{\neg\mathsf{h}}{\to}_{\oplus})$: $\underset{\neg\mathsf{h}}{\to}_{\oplus} \cdot \underset{\mathsf{h}}{\to}_{\beta} \subseteq \underset{\mathsf{h}}{\to}_{\beta}{}^{*} \cdot \underset{\neg\mathsf{h}}{\to}_{\oplus}{}^{=}$ *does not hold.* The phenomenon is similar to Ex. 2.5, but now moving to parallel reduction is not a solution: the problem here is not just compressing steps, but the fact that by swapping $\underset{\neg\mathsf{h}}{\to}_{\oplus}$ and $\underset{\mathsf{h}}{\to}_{\beta}$, a *third* relation $\underset{\mathsf{h}}{\to}_{\oplus}$ appears.

Note that the problem above is specific to factorization, and does not appear with confluence.

A Robust Condition for Modular Factorization. Inspired by Accattoli's study of factorization for λ-calculi with explicit substitutions [2], we consider an alternative sufficient condition for modular factorization, which holds in many examples, as the next sections shall show.

We say that $\underset{\mathsf{i}}{\to}_{\alpha}$ **linearly swaps** with $\underset{\mathsf{e}}{\to}_{\gamma}$ if

$$\mathtt{lSwap}(\underset{\mathsf{i}}{\to}_{\alpha}, \underset{\mathsf{e}}{\to}_{\gamma}) : \ \underset{\mathsf{i}}{\to}_{\alpha} \cdot \underset{\mathsf{e}}{\to}_{\gamma} \subseteq \underset{\mathsf{e}}{\to}_{\gamma} \cdot \to_{\alpha}^{*} \qquad \textbf{(Linear Swap)}$$

Note that, on the right-hand side, the relation is $\to_{\alpha}^{*}$, not $\underset{\mathsf{i}}{\to}_{\alpha}$. This small change will make a big difference, and overcome the issue we have seen in Ex. 3.2 (note that there $\underset{\mathsf{i}}{\to}_{\beta}\underset{\mathsf{h}}{\to}_{\oplus} \subseteq \underset{\mathsf{h}}{\to}_{\beta} \to_{\oplus}^{*}$ holds). Perhaps surprisingly, this easy-to-check condition, which is *local* and *linear* in $\underset{\mathsf{e}}{\to}_{\gamma}$, suffices, and holds in a large variety of cases. Moreover, it holds *directly* (even with β) that is, without the mediating role of parallel reductions (as it is the instead the case of Takahashi's technique, see Ex. 2.5).

We finally obtain a modular factorization technique, via the following easy property.

▶ **Lemma 3.3.** $\to_{a} \cdot \to_{b} \subseteq \to_{b} \cdot \to_{c}^{*}$ *implies* $\to_{a}^{*} \cdot \to_{b} \subseteq \to_{b} \cdot \to_{c}^{*}$.

▶ **Theorem 3.4** (Modular factorization). *Let* $\to_{\alpha} = (\underset{\mathsf{e}}{\to}_{\alpha} \cup \underset{\mathsf{i}}{\to}_{\alpha})$ *and* $\to_{\gamma} = (\underset{\mathsf{e}}{\to}_{\gamma} \cup \underset{\mathsf{i}}{\to}_{\gamma})$ *be* e*-factorizing relations. The union* $\to_{\alpha} \cup \to_{\gamma}$ *satisfies* e*-factorization* $\mathtt{Fact}(\underset{\mathsf{e}}{\to}, \underset{\mathsf{i}}{\to})$*, for* $\underset{\mathsf{e}}{\to} := \underset{\mathsf{e}}{\to}_{\alpha} \cup \underset{\mathsf{e}}{\to}_{\gamma}$*, and* $\underset{\mathsf{i}}{\to} := \underset{\mathsf{i}}{\to}_{\alpha} \cup \underset{\mathsf{i}}{\to}_{\gamma}$*, if the following linear swaps hold:*

$$\mathtt{lSwap}(\underset{\mathsf{i}}{\to}_{\alpha}, \underset{\mathsf{e}}{\to}_{\gamma}) : \ \underset{\mathsf{i}}{\to}_{\alpha} \cdot \underset{\mathsf{e}}{\to}_{\gamma} \subseteq \underset{\mathsf{e}}{\to}_{\gamma} \cdot \to_{\alpha}^{*} \quad \textit{and} \quad \mathtt{lSwap}(\underset{\mathsf{i}}{\to}_{\gamma}, \underset{\mathsf{e}}{\to}_{\alpha}) : \ \underset{\mathsf{i}}{\to}_{\gamma} \cdot \underset{\mathsf{e}}{\to}_{\alpha} \subseteq \underset{\mathsf{e}}{\to}_{\alpha} \cdot \to_{\gamma}^{*}.$$

Proof. We prove that the assumptions imply $\mathtt{SP}(\underset{\mathsf{e}}{\to}, \underset{\mathsf{i}}{\to}_{\alpha}{}^{*} \cup \underset{\mathsf{i}}{\to}_{\gamma}{}^{*})$, hence $\mathtt{PP}(\underset{\mathsf{e}}{\to}, \underset{\mathsf{i}}{\to}_{\alpha}{}^{*} \cup \underset{\mathsf{i}}{\to}_{\gamma}{}^{*})$ (by Lemma 2.4). Therefore $\mathtt{PP}(\underset{\mathsf{e}}{\to}, \underset{\mathsf{i}}{\to})$ by Lemma 2.3 (because $(\underset{\mathsf{i}}{\to}_{\alpha} \cup \underset{\mathsf{i}}{\to}_{\gamma})^{*} = (\underset{\mathsf{i}}{\to}_{\alpha}{}^{*} \cup \underset{\mathsf{i}}{\to}_{\gamma}{}^{*})^{*}$ by property **TR**), and so $\mathtt{Fact}(\underset{\mathsf{e}}{\to}, \underset{\mathsf{i}}{\to})$ holds according to Lemma 2.2.

To verify $\mathtt{SP}(\underset{\mathsf{e}}{\to}_{\alpha} \cup \underset{\mathsf{e}}{\to}_{\gamma}, \underset{\mathsf{i}}{\to}_{\alpha}{}^{*} \cup \underset{\mathsf{i}}{\to}_{\gamma}{}^{*})$, we observe that $\underset{\mathsf{i}}{\to}_{k}{}^{*} \cdot \underset{\mathsf{e}}{\to}_{j} \subseteq (\underset{\mathsf{e}}{\to}_{j} \cup \underset{\mathsf{e}}{\to}_{k})^{*} \cdot \underset{\mathsf{i}}{\to}_{k}{}^{*}$ for all $k, j \in \{\alpha, \gamma\}$:

- **Case $j = k$.** This is immediate by e-factorization of $\to_{\alpha}$ and $\to_{\gamma}$, and by Lemma 2.2.1.
- **Case $j \neq k$.** $\mathtt{lSwap}(\underset{\mathsf{i}}{\to}_{\alpha}, \underset{\mathsf{e}}{\to}_{\gamma})$ implies $(\underset{\mathsf{i}}{\to}_{\alpha})^{*} \cdot \underset{\mathsf{e}}{\to}_{\gamma} \subseteq \underset{\mathsf{e}}{\to}_{\gamma} \cdot \to_{\alpha}^{*}$, by Lemma 3.3. Since $\to_{\alpha}$ e-factorizes, we obtain $(\underset{\mathsf{i}}{\to}_{\alpha})^{*} \cdot \underset{\mathsf{e}}{\to}_{\gamma} \subseteq \underset{\mathsf{e}}{\to}_{\gamma} \cdot \underset{\mathsf{e}}{\to}_{\alpha}{}^{*} \cdot \underset{\mathsf{i}}{\to}_{\alpha}{}^{*}$. Similarly for $\mathtt{lSwap}(\underset{\mathsf{i}}{\to}_{\gamma}, \underset{\mathsf{e}}{\to}_{\alpha})$. ◀

Note that in the proof of Theorem 3.4, the assumption that $\to_{\alpha}$ and $\to_{\gamma}$ factorize is crucial. Using that, together with Lemma 3.3, we obtain $\mathtt{SP}(\underset{\mathsf{e}}{\to}, \underset{\mathsf{i}}{\to}_{\alpha}{}^{*})$, that is, $\underset{\mathsf{i}}{\to}_{\alpha}{}^{*}$ postpones after both e-steps, (and similarly for $\underset{\mathsf{i}}{\to}_{\gamma}{}^{*}$). Note also that $\mathtt{lSwap}(\underset{\mathsf{i}}{\to}, \underset{\mathsf{e}}{\to})$ – taken alone – does not imply $\mathtt{PP}(\underset{\mathsf{e}}{\to}, \underset{\mathsf{i}}{\to})$. For instance, let's consider again Ex. 2.5. It is clear that $\mathtt{lSwap}(\underset{\neg\mathsf{h}}{\to}_{\beta}, \underset{\mathsf{h}}{\to}_{\beta})$ holds and yet it does not imply $\mathtt{Fact}(\underset{\mathsf{h}}{\to}_{\beta}, \underset{\neg\mathsf{h}}{\to}_{\beta})$. Stronger tools, such as parallel reduction or finite development are needed here – there is no magic.

The next sections apply the modularization result to various λ-calculus extensions.

Linear Postponement. We collect here two easy properties which shall simplify the proof of factorization in several of the case studies (to prove them, use Lemma 3.3 and Lemma 2.4).

▶ **Lemma 3.5** (Linear postponement).

1. $(\xrightarrow[\mathsf{i}]{} \cdot \xrightarrow[\mathsf{e}]{} \subseteq \xrightarrow[\mathsf{e}]{} \cdot \xrightarrow[\mathsf{i}]{}^{*}) \;\Rightarrow\; \mathtt{SP}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{}^{*}) \;\Rightarrow\; \mathtt{Fact}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{})$.
2. $(\xrightarrow[\mathsf{i}]{} \cdot \xrightarrow[\mathsf{e}]{} \subseteq \xrightarrow[\mathsf{e}]{} \cdot \to^{=}) \;\Rightarrow\; \mathtt{SP}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{}) \;\Rightarrow\; \mathtt{Fact}(\xrightarrow[\mathsf{e}]{}, \xrightarrow[\mathsf{i}]{})$.

Factorization vs. Confluence. Factorization and confluence are *independent* properties. In Sect. 5 we apply our modular factorization technique to a non-confluent calculus. Conversely, $\beta\eta$, which is confluent, does not verify head nor leftmost factorization, even though both β and η – separately – do.

4 Extensions of the Call-by-Name λ-Calculus: Head Factorization

We shall study factorization theorems for extensions of both of the call-by-name (shortened to CbN) and of the call-by-value (CbV) λ-calculus. The CbN λ-calculus – also simply known as λ-calculus – is the set of λ-terms Λ, equipped with the β-reduction, while the CbV λ-calculus is the set of λ-terms Λ, equipped with the β_v-reduction.

In this section, we first revise the language of the λ-calculus and then consider the case where the calculus is *enriched with new operators*, such as a non-deterministic choice or a fixpoint operator – so, together with β, we have other reduction rules. We study in this setting *head factorization*, which is by far the most important and common factorization scheme in λ-calculus. We show that here our modular technique further simplifies, providing an easy, *ready-to-use* test for head factorization of compound systems (Proposition 4.5). Indeed, verifying the two linear swap conditions of Theorem 3.4 now reduces to a single, simple test. Such a simplification only relies on β and on the properties of the contextual closure, that is, it holds independently of the specific form of the extra rule.

4.1 The (Applied) λ-Calculus

Since in the next sections we shall extend the λ-calculus with new operators, such as a non-deterministic choice $\oplus$ or a fixpoint combinator Y, we include constants in the syntax, which are meant to represent such operators. So, for instance, in Sect. 5 we shall see $\oplus$ as a constant. This way factorization results with respect to β-reduction can be seen as holding also in the λ-calculus with extended syntax – this is absolutely harmless.

Note that despite the fact that the classic Barendregt's book [11] defines the λ-calculus without constants (the calculus is pure), other classic references such as Hindley and Seldin's book [28] or Plotkin [46] do include constants in the language of terms – thus there is nothing exotic in our approach. Following Hindley and Seldin, when the set of constants is empty, the calculus is called *pure*, otherwise *applied*.

The Language. The following grammars generate λ-terms and contexts.

$$t,u,p,q,r,s ::= x \mid c \mid \lambda x.t \mid ts \quad (\textbf{terms}, \Lambda) \qquad\qquad \mathsf{C} ::= \langle\,\rangle \mid t\mathsf{C} \mid \mathsf{C}t \mid \lambda x.\mathsf{C} \quad (\textbf{contexts})$$

where x ranges over a countable set of *variables*, c over a disjoint (finite, infinite or empty) set of constants. Variables and constants are *atoms*, terms of shape pq are *applications*, and $\lambda x.p$ *abstractions*. If the constants are $c_1, ..., c_n$, the set of terms is sometimes noted as $\Lambda_{c_1 \ldots c_n}$.

The plugging $\mathsf{C}\langle t\rangle$ of a term t into a context C is the term obtained by replacing the only occurrence of the hole $\langle\,\rangle$ in C with t, possibly capturing some free variables of t.

A *reduction step* $\to_\gamma$ is defined as the contextual closure of a relation $\mapsto_\gamma$ on Λ called *root* or (γ-)*rule*. Explicitly, $t \to_\gamma s$ if $t = \mathsf{C}\langle r\rangle$ and $s = \mathsf{C}\langle r'\rangle$, for some context C with $r \mapsto_\gamma r'$; the term r is called a γ-*redex.* Given two rules $\mapsto_\alpha, \mapsto_\gamma$ on Λ, the relation $\to_{\alpha\gamma}$ is $\to_\alpha \cup \to_\gamma$, which can equivalently be defined as the contextual closure of $\mapsto_\alpha \cup \mapsto_\gamma$.

The (*CbN*) λ-*calculus* is $(\Lambda, \to_\beta)$, the set Λ of terms and β-*reduction* $\to_\beta$, defined as the contextual closure of the β-rule: $(\lambda x.p)q \mapsto_\beta p\{x{:=}q\}$ where $p\{x{:=}q\}$ denotes capture-avoiding substitution. We work up to α-equivalence; $\mathsf{fv}(t)$ is the set of free variables of t.

Properties of the Contextual Closure. Here we recall basic properties about contextual closures and substitution, preparing the ground for the simplifications studied next.

A relation $\looparrowright$ on terms is *substitutive* if

$$r \looparrowright r' \text{ implies } r\{x{:=}q\} \looparrowright r'\{x{:=}q\}. \qquad (\textbf{Substitutivity})$$

An obvious induction on the shape of terms shows the following (see Barendregt [11], p. 54).

▶ **Property 4.1** (Substitution). *Let $\to_\gamma$ be the contextual closure of $\mapsto_\gamma$.*

1. *If $\mapsto_\gamma$ is substitutive then $\to_\gamma$ is substitutive: $p \to_\gamma p'$ implies $p\{x{:=}q\} \to_\gamma p'\{x{:=}q\}$.*
2. *If $q \to_\gamma q'$ then $t\{x{:=}q\} \to_\gamma^* t\{x{:=}q'\}$.*

We recall a basic but key property of contextual closures. If a step $\to_\gamma$ is obtained by closure under *non-empty context* of a rule $\mapsto_\gamma$, then it preserves the shape of the term:

▶ **Property 4.2** (Shape preservation). *Assume $t = C\langle r\rangle \to C\langle r'\rangle = t'$ and that context C is* non-empty*. The term t' is an application (resp. an abstraction) if and only if t is.*

Since the closure of $\mapsto_\gamma$ under the *empty* context $\langle\,\rangle$ is always an *essential* step (whatever head, left, or weak), Property 4.2 implies that non-essential steps always preserve the shape of terms – we spell this out in Properties A.1 and A.2 in the Appendix. Notice that $\mapsto_\gamma$ indicates the step $\to_\gamma$ that is obtained by *empty contextual closure.*

Head Reduction. Head contexts are defined as follows:

$$\mathsf{H} ::= \lambda x_1 \ldots \lambda x_k.\langle\,\rangle t_1 \ldots t_n \qquad (\textbf{head contexts})$$

where $k \geq 0$ and $n \geq 0$. A *non-head context* is a context that is not head. A *head* step $\underset{\mathsf{h}}{\to}_\gamma$ (resp. *non-head* step $\underset{\neg\mathsf{h}}{\to}_\gamma$) is defined as the closure under head contexts (resp. non-head contexts) of the rule $\mapsto_\gamma$. Clearly, $\to_\gamma = \underset{\mathsf{h}}{\to}_\gamma \cup \underset{\neg\mathsf{h}}{\to}_\gamma$. Head steps play the role of essential steps.

Note that the *empty context* $\langle\,\rangle$ is a *head context.* Therefore $\mapsto_\gamma \subseteq \underset{\mathsf{h}}{\to}_\gamma$ holds (a fact that we shall freely use) and Property 4.2 always applies to non-head steps.

4.2 Call-by-Name: Head Factorization, Modularly

Head factorization is of great importance for the theory of the CbN λ-calculus, which is why head factorization for $\to_\beta$ is well studied. If we consider a calculus $(\Lambda, \to_\beta \cup \to_\gamma)$, where $\to_\gamma$ is a new reduction added to β, our modular technique (Theorem 3.4) states that the compound system $\to_\beta \cup \to_\gamma$ satisfies head factorization if $\to_\gamma$ does, and both $\texttt{lSwap}(\underset{\neg\mathsf{h}}{\to}_\beta, \underset{\mathsf{h}}{\to}_\gamma)$ and $\texttt{lSwap}(\underset{\neg\mathsf{h}}{\to}_\gamma, \underset{\mathsf{h}}{\to}_\beta)$ hold. We show that in the head case our technique simplifies even more, reducing to the test in Proposition 4.5.

First, we observe that in this case, *any* linear swap condition can be tested by considering for the head step only the root relation $\mapsto$, that is, only the closure of $\mapsto$ under *empty* context, which is a head step by definition. This is expressed in the following lemma, where we include also a variant that shall be useful later on.

▶ **Lemma 4.3** (Lifting root linear swaps). *Let $\mapsto_\alpha$ and $\mapsto_\gamma$ be root relations on Λ.*

1. $\underset{\neg\mathsf{h}}{\rightarrow}_\alpha \cdot \mapsto_\gamma \subseteq \underset{\mathsf{h}}{\rightarrow}_\gamma \cdot \rightarrow_\alpha^*$ *implies* $\mathtt{lSwap}(\underset{\neg\mathsf{h}}{\rightarrow}_\alpha, \underset{\mathsf{h}}{\rightarrow}_\gamma)$.
2. *Similarly,* $\underset{\neg\mathsf{h}}{\rightarrow}_\alpha \cdot \mapsto_\gamma \subseteq \underset{\mathsf{h}}{\rightarrow}_\gamma \cdot \rightarrow_\alpha^=$ *implies* $\underset{\neg\mathsf{h}}{\rightarrow}_\alpha \cdot \rightarrow_\gamma \subseteq \underset{\mathsf{h}}{\rightarrow}_\gamma \cdot \rightarrow_\alpha^=$.

Second, since we are studying $\rightarrow_\beta \cup \rightarrow_\gamma$, one of the linear swaps is $\mathtt{lSwap}(\underset{\neg\mathsf{h}}{\rightarrow}_\gamma, \underset{\mathsf{h}}{\rightarrow}_\beta)$. We show that, whatever is $\rightarrow_\gamma$, it linearly swaps with $\underset{\mathsf{h}}{\rightarrow}_\beta$ as soon as $\mapsto_\gamma$ is *substitutive*.

▶ **Lemma 4.4** (Swap with $\underset{\mathsf{h}}{\rightarrow}_\beta$). *If $\mapsto_\gamma$ is substitutive then* $\mathtt{lSwap}(\underset{\neg\mathsf{h}}{\rightarrow}_\gamma, \underset{\mathsf{h}}{\rightarrow}_\beta)$ *holds.*

The proofs of Lemma 4.3 and Lemma 4.4 are in Appendix A.1.

Summing up, since head factorization for β is known, we obtain the following test to verify that the compound system $\rightarrow_\beta \cup \rightarrow_\gamma$ satisfies head factorization $\mathtt{Fact}(\underset{\mathsf{h}}{\rightarrow}_\beta \cup \underset{\mathsf{h}}{\rightarrow}_\gamma, \underset{\neg\mathsf{h}}{\rightarrow}_\beta \cup \underset{\neg\mathsf{h}}{\rightarrow}_\gamma)$.

▶ **Proposition 4.5** (A test for modular head factorization). *Let $\rightarrow_\beta$ be β-reduction and $\rightarrow_\gamma$ be the contextual closure of a rule $\mapsto_\gamma$. Their union $\rightarrow_\beta \cup \rightarrow_\gamma$ satisfies head factorization if:*

1. Head factorization of $\rightarrow_\gamma$*:* $\mathtt{Fact}(\underset{\mathsf{h}}{\rightarrow}_\gamma, \underset{\neg\mathsf{h}}{\rightarrow}_\gamma)$.
2. Root linear swap*:* $\underset{\neg\mathsf{h}}{\rightarrow}_\beta \cdot \mapsto_\gamma \subseteq \underset{\mathsf{h}}{\rightarrow}_\gamma \cdot \rightarrow_\beta^*$.
3. Substitutivity*: $\mapsto_\gamma$ is substitutive.*

Note that none of the properties concerns $\rightarrow_\beta$ alone, as we already know that head factorization of $\rightarrow_\beta$ holds. In Sect. 5 we shall use our test (Proposition 4.5) to prove head factorization for the non-deterministic λ-calculus. The full proof is only a few lines long.

We conclude by observing that Lemma 4.3 gives either a proof that the swap conditions hold, or a counter-example. Let us give an example of this latter use.

▶ **Example 4.6** (Finding counter-examples). The test of Proposition 4.5 can also be used to provide a counter-example to head factorization when it fails. Let's instantiate $\rightarrow_\gamma$ with $\rightarrow_\eta$, that is, the contextual closure of rule $\lambda x.tx \mapsto_\eta t$ if $x \notin \mathsf{fv}(t)$. Now, consider the root linear swap: $t \coloneqq \lambda x.(II)(Ix) \underset{\neg\mathsf{h}}{\rightarrow}_\beta \lambda x.(II)x \mapsto_\eta II \eqqcolon s$, where $I \coloneqq \lambda z.z$. Note that t has no $\underset{\mathsf{h}}{\rightarrow}_\eta$ step, and so the two steps cannot be swapped. The reduction sequence above is a *counter-example to both head and leftmost factorization* for $\beta\eta$. Start with the head (and leftmost) redex II: $\lambda x.(II)(Ix) \underset{\mathsf{h}}{\rightarrow}_{\beta\eta} \lambda x.I(Ix)$. From $\lambda x.I(Ix)$, there is no way to reach s. We recall that $\beta\eta$ still satisfies leftmost normalization – the proof is non-trivial [31, 53, 60, 30].

5 The Non-Deterministic λ-Calculus $\Lambda_\oplus$

De' Liguoro and Piperno's non-deterministic λ-calculus $\Lambda_\oplus$ is defined in [16] by extending the λ-calculus with a new operator $\oplus$ whose rule models *non-deterministic choice*. Intuitively, $t \oplus p$ non-deterministically rewrites to either t or p. Notably, $\Lambda_\oplus$ is *not confluent*, hence it is a good example of the fact that confluence and factorization are independent properties.

We briefly recall $\Lambda_\oplus$ and its features, then use our technique to give a *novel and neat* proof of de' Liguoro and Piperno's *head factorization* result [16, Cor. 2.10].

Syntax. We slightly depart from the presentation in [16], as we consider $\oplus$ as a constant, and write $\oplus tp$ rather than $t \oplus p$, working as usual for the λ-calculus with constants (see *e.g.*, [28], or [11, Sec. 15.3]).[1] Terms and contexts are generated by:

$$t, p, q, r ::= x \mid \oplus \mid \lambda x.t \mid tp \quad (\textbf{terms}, \ \Lambda_\oplus) \qquad \mathsf{C} ::= \langle\,\rangle \mid t\mathsf{C} \mid \mathsf{C}t \mid \lambda x.\mathsf{C} \quad (\textbf{contexts})$$

As before, $\rightarrow_\beta$ denotes β-reduction, while the rewrite step $\rightarrow_\oplus$ is the contextual closure of the following non-deterministic rule: $\oplus tp \mapsto_\oplus t$ and $\oplus tp \mapsto_\oplus p$.

[1] Note that there is no loss with respect to the syntax in [16], where $\oplus$ comes with exactly two arguments, because in our formalism such a restriction defines a sub-system that is closed under reduction.

Subtleties. The calculus $(\Lambda_\oplus, \to_\beta \cup \to_\oplus)$ is not trivial. Clearly, $\to_\beta \cup \to_\oplus$ is not confluent. Moreover, the following examples from [16] show that permuting β and $\oplus$ steps is delicate.

- $\to_\oplus$ *creates β-redexes.* For instance, $(\oplus\, y\, (\lambda x.x))z \to_\oplus (\lambda x.x)z \to_\beta z$, hence the $\to_\oplus$-step cannot be postponed after $\to_\beta$.
- *Choice duplication.* Postponing $\to_\beta$ after $\to_\oplus$ is also problematic, because β-steps may multiply choices, introducing new results: flipping a coin and duplicating the result is not equivalent to duplicating the coin and then flipping twice. For instance, let $t = (\lambda x.xx)(\oplus pq)$. Duplicating first one may have $t \to_\beta (\oplus pq)(\oplus pq) \to_\oplus q(\oplus pq) \to_\oplus qp$ while flipping first one has $t \to_\oplus (\lambda x.xx)p \to_\beta pp$ or $t \to_\oplus (\lambda x.xx)q \to_\beta qq$ but in both cases qp cannot be reached.

These examples are significant as the same issues impact any calculus with choice effects.

Head Factorization. The head (resp. non-head)[2] rewrite steps $\underset{\mathsf{h}}{\to}_\beta$ and $\underset{\mathsf{h}}{\to}_\oplus$ (resp. $\underset{\neg\mathsf{h}}{\to}_\beta$ and $\underset{\neg\mathsf{h}}{\to}_\oplus$) are defined as the closure by head (resp. non-head) contexts of rules $\mapsto_\beta$ and $\mapsto_\oplus$, respectively. We also set $\underset{\mathsf{h}}{\to} := \underset{\mathsf{h}}{\to}_\beta \cup \underset{\mathsf{h}}{\to}_\oplus$ and $\underset{\neg\mathsf{h}}{\to} := \underset{\neg\mathsf{h}}{\to}_\beta \cup \underset{\neg\mathsf{h}}{\to}_\oplus$.

De' Liguoro and Piperno prove that despite the failure of confluence, $\Lambda_\oplus$ satisfies head factorization. They prove this result via standardization, following Klop's technique [31].

▶ **Theorem 5.1** (Head factorization, Cor. 2.10 in [16]). $\mathtt{Fact}(\underset{\mathsf{h}}{\to}, \underset{\neg\mathsf{h}}{\to})$ *holds in the non-deterministic λ-calculus $\Lambda_\oplus$.*

A New Proof, Modularly. We give a novel, strikingly simple proof of $\mathtt{Fact}(\underset{\mathsf{h}}{\to}, \underset{\neg\mathsf{h}}{\to})$, simply by proving that $\to_\beta$ and $\to_\oplus$ satisfy the hypotheses of the test for modular head factorization (Proposition 4.5). All the ingredients we need are given by the following easy lemma.

▶ **Lemma 5.2** (Root linear swaps).

1. $t \underset{\neg\mathsf{h}}{\to}_\beta p \mapsto_\oplus q$ *implies* $t \mapsto_\oplus \cdot \to_\beta^{=} q$.
2. $t \underset{\neg\mathsf{h}}{\to}_\oplus p \mapsto_\oplus q$ *implies* $t \mapsto_\oplus \cdot \to_\oplus^{=} q$.

Proof.

1. Let $p = \oplus p_1 p_2$ and assume $\oplus p_1 p_2 \mapsto_\oplus p_i = q$, with $i \in \{1, 2\}$. Since $t \underset{\neg\mathsf{h}}{\to}_\beta p$, then by Property 4.2 (as spelled out in Property A.1) t has shape $\oplus t_1 t_2$, with $\oplus t_1 t_2 \underset{\neg\mathsf{h}}{\to}_\beta \oplus p_1 p_2$. Therefore, either $t_1 \to_\beta p_1$ or $t_2 \to_\beta p_2$, from which $t = \oplus t_1 t_2 \mapsto_\oplus t_i \to_\beta^{=} p_i = q$.
2. The proof is the same as above, just replace β with $\oplus$. ◀

▶ **Theorem 5.3** (Testing head factorization). *We have* $\mathtt{Fact}(\underset{\mathsf{h}}{\to}, \underset{\neg\mathsf{h}}{\to})$ *because we have:*

1. Head factorization of $\to_\oplus$*:* $\mathtt{Fact}(\underset{\mathsf{h}}{\to}_\oplus, \underset{\neg\mathsf{h}}{\to}_\oplus)$.
2. Root linear swap*:* $\underset{\neg\mathsf{h}}{\to}_\beta \cdot \mapsto_\oplus \subseteq \underset{\mathsf{h}}{\to}_\oplus \cdot \to_\beta^{=}$.
3. Substitutivity*: $\mapsto_\oplus$ is substitutive.*

Proof. We prove the hypotheses of Proposition 4.5.

1. $\underset{\neg\mathsf{h}}{\to}_\oplus$ linearly postpones after $\underset{\mathsf{h}}{\to}_\oplus$ because lifting the swap in Lemma 5.2.2 via Lemma 4.3.2 (with $\alpha = \gamma = \oplus$) gives $t \underset{\neg\mathsf{h}}{\to}_\oplus p \underset{\mathsf{h}}{\to}_\oplus q \subseteq t \underset{\mathsf{h}}{\to}_\oplus \cdot \to_\oplus^{=} q$. Lemma 3.5.2 gives Factorization.
2. This is exactly Lemma 5.2.1.
3. By definition of substitution $(\oplus p_1 p_2)\{x{:=}q\} = \oplus\, p_1\{x{:=}q\}\, p_2\{x{:=}q\} \mapsto_\oplus p_i\{x{:=}q\}$. ◀

[2] Non-head steps are called *internal* ($\underset{\mathsf{i}}{\to}$) in [16].

6 Extensions of the CbV λ-Calculus: Left and Weak Factorization

Plotkin's call-by-value (CbV) λ-calculus [46] is the restriction of the λ-calculus where β-redexes can be fired only when the argument is a *value*, where values are defined by:

$$v \ ::= \ x \mid c \mid \lambda x.t \qquad (\textbf{values}, \mathcal{V})$$

The CbV λ-calculus is given by the pair $(\Lambda, \to_{\beta_v})$, where *β_v-reduction* $\to_{\beta_v}$ is the contextual closure of the following rule $\mapsto_{\beta_v}$: $(\lambda x.t)v \mapsto_{\beta_v} t\{x{:=}v\}$, where v is a value.

Left and Weak Reduction. In the literature on the CbV λ-calculus, factorization is considered with respect to various essential reductions. Usually, the essential reduction is *weak*, that is, it does not act under abstractions. There are three main weak schemes: reducing from left to right, as originally done by Plotkin [46], from right to left, as done for instance by Leroy's ZINC abstract machine [35], or in an unspecified non-deterministic order, used for example by Dal Lago and Martini [33].

Here we focus on the left(-to-right) and the (unspecified) weak schemes. *Left* contexts L and *weak* contexts W are respectively defined by:

$$\mathsf{L} ::= \langle\,\rangle \mid \mathsf{L}t \mid v\mathsf{L} \quad (\textbf{left contexts}) \qquad\qquad \mathsf{W} ::= \langle\,\rangle \mid \mathsf{W}t \mid t\mathsf{W} \quad (\textbf{weak contexts})$$

Given a rule $\mapsto_\gamma$, a *left* step $\underset{\mathsf{l}}{\to}_\gamma$ (resp., a *weak* step $\underset{\mathsf{w}}{\to}_\gamma$) is the closure of $\mapsto_\gamma$ under a left (resp. weak) context. A *non-left* step $\underset{\neg\mathsf{l}}{\to}_\gamma$ (resp. *non-weak* step $\underset{\neg\mathsf{w}}{\to}_\gamma$) is the closure of $\mapsto_\gamma$ under a context that is not left (resp. not weak).

Left/Weak Factorization, Modularly. For both left and weak reductions, we derive a test for modular factorization analogous to the test for head factorization (Proposition 4.5). Note that we already know that $(\Lambda, \to_{\beta_v})$ satisfies left and weak factorization: the former was proved by Plotkin [46], the latter is folklore – a proof can be found in our previous work [3].

▶ **Proposition 6.1** (A test for modular left/weak factorization). *Let $\to_{\beta_v}$ be β_v-reduction, $\to_\gamma$ be the contextual closure of a rule $\mapsto_\gamma$, and $\mathsf{e} \in \{\mathsf{l}, \mathsf{w}\}$. Their union $\to_{\beta_v} \cup \to_\gamma$ satisfies* e-*factorization* $\texttt{Fact}(\underset{\mathsf{e}}{\to}_{\beta_\mathsf{v}} \cup \underset{\mathsf{e}}{\to}_\gamma, \underset{\neg\mathsf{e}}{\to}_{\beta_\mathsf{v}} \cup \underset{\neg\mathsf{e}}{\to}_\gamma)$ *if:*

1. e-factorization of $\to_\gamma$*:* $\texttt{Fact}(\underset{\mathsf{e}}{\to}_\gamma, \underset{\neg\mathsf{e}}{\to}_\gamma)$.
2. Root linear swap*:* $\underset{\neg\mathsf{e}}{\to}_{\beta_v} \cdot \mapsto_\gamma \ \subseteq \ \underset{\mathsf{e}}{\to}_\gamma \cdot \to^*_{\beta_v}$.
3. Substitutivity*: $\mapsto_\gamma$ is substitutive.*

The easy proof is in Appendix A.2.

7 The Shuffling Calculus

Plotkin's CbV λ-calculus is usually considered on closed terms. When dealing with open terms, it is well known that a mismatch between the operational and the denotational semantics arises, as first pointed out by Paolini and Ronchi della Rocca [45, 44, 49]. The literature contains several proposals of extensions of β_v reduction to overcome this issue, see Accattoli and Guerrieri for discussions [4]. One such refinement is Carraro and Guerrieri's *shuffling calculus* [14], which extends Plotkin's λ-calculus with extra rules (without adding new operators). These rules are inspired by linear logic proof nets, and are the CbV analogous of Regnier's σ rules [47]. Left factorization for the shuffling calculus is studied by Guerrieri, Paolini, and Ronchi della Rocca in [24, 25, 23], by adapting Takahashi's technique [53].

We recall the calculus, then use our technique to give a new proof of factorization, both left (as in [25]) and weak (new). Remarkably, our proofs are *very short*, whereas the original requires several pages (to define parallel reductions and prove their properties).

The Syntax. The *shuffling calculus* is simply Plotkin's calculus extended with *σ-reduction* $\rightarrow_{\sigma}$, that is, the contextual closure of the root relation $\mapsto_{\sigma} = \mapsto_{\sigma_1} \cup \mapsto_{\sigma_3}$, where

$$(\lambda x.t)us \mapsto_{\sigma_1} (\lambda x.ts)u \;\text{ if } x \notin \mathsf{fv}(s) \qquad\qquad v((\lambda x.t)u) \mapsto_{\sigma_3} (\lambda x.vt)u \;\text{ if } x \notin \mathsf{fv}(v)$$

We write $\rightarrow_{\sigma_i}$ for the contextual closure of $\mapsto_{\sigma_i}$ (so $\rightarrow_{\sigma} = \rightarrow_{\sigma_1} \cup \rightarrow_{\sigma_3}$), and $\rightarrow_{\mathsf{sh}} = \rightarrow_{\beta_v} \cup \rightarrow_{\sigma}$.

Subtleties. From a rewriting perspective, the shuffling calculus is an interesting extension of the λ-calculus because its intricate rules do not fit into easy-to-manage classes of rewriting systems. Orthogonal systems have only simple forms of overlaps of redexes. While the λ-calculus is an orthogonal system, the σ-rules introduce non-trivial overlaps such as the following ones. Setting $I := \lambda x.x$ and $\delta := \lambda x.xx$, the term $\delta I \delta$ is a σ_1-redex and contains the β_v-redex δI, while the term $\delta(I\delta)(xI)$ is a σ_1-redex and contains the σ_3-redex $\delta(I\delta)$, which contains in turn the β_v-redex $I\delta$.

Left and Weak Factorization. Despite all these traits, the shuffling calculus has good properties, such as confluence [14], and left factorization [25]. Moreover, $\rightarrow_{\sigma}$ is terminating [14]. The tests developed in the previous section allow us to easily prove both left and weak factorization. We check the hypotheses of Proposition 6.1 – all the ingredients we need are in Lemma 7.1 (the easy details are in Appendix A.3). Note that the *empty context is both a left and a weak* context, hence $t \mapsto_{\sigma_i} u$ implies both $t \underset{\mathsf{l}}{\rightarrow}_{\sigma_i} u$ and $t \underset{\mathsf{w}}{\rightarrow}_{\sigma_i} u$.

▶ **Lemma 7.1** (Root linear swaps). *Let* $\mathsf{e} \in \{\mathsf{l}, \mathsf{w}\}$ *and* $i \in \{1, 3\}$*. Then:*

1. $\underset{\neg\mathsf{e}}{\rightarrow}_{\beta_v} \cdot \mapsto_{\sigma_i} \subseteq \mapsto_{\sigma_i} \cdot \rightarrow_{\beta_v}$.
2. $\underset{\neg\mathsf{e}}{\rightarrow}_{\sigma} \cdot \mapsto_{\sigma_i} \subseteq \mapsto_{\sigma_i} \cdot \rightarrow_{\sigma}$.

▶ **Theorem 7.2** (Testing left (weak) factorization). *Let* $\mathsf{e} \in \{\mathsf{l}, \mathsf{w}\}$*.* $\texttt{Fact}(\underset{\mathsf{e}}{\rightarrow}_{\mathsf{sh}}, \underset{\neg\mathsf{e}}{\rightarrow}_{\mathsf{sh}})$ *holds, as:*

1. Left (resp. weak) factorization of $\rightarrow_{\sigma}$*:* $\texttt{Fact}(\underset{\mathsf{e}}{\rightarrow}_{\sigma}, \underset{\neg\mathsf{e}}{\rightarrow}_{\sigma})$.
2. Root linear swap*:* $\underset{\neg\mathsf{e}}{\rightarrow}_{\beta_v} \cdot \mapsto_{\sigma} \subseteq \underset{\mathsf{e}}{\rightarrow}_{\sigma} \cdot \rightarrow_{\beta_v}$.
3. Substitutivity*:* $\mapsto_{\sigma_i}$ *is substitutive, for* $i \in \{1, 3\}$.

Proof. We prove the hypotheses of Proposition 6.1:

1. Left (resp. weak) factorization of $\rightarrow_{\sigma}$ holds because $\underset{\neg\mathsf{e}}{\rightarrow}_{\sigma}$ linearly postpones after $\underset{\mathsf{e}}{\rightarrow}_{\sigma}$: indeed, by Lemma 7.1.2 $\underset{\neg\mathsf{e}}{\rightarrow}_{\sigma} \cdot \mapsto_{\sigma} \subseteq \underset{\mathsf{e}}{\rightarrow}_{\sigma} \cdot \rightarrow_{\sigma}$ and by contextual closure (Lemma A.3 with $\alpha = \gamma = \sigma$) we have that $\underset{\neg\mathsf{e}}{\rightarrow}_{\sigma} \cdot \underset{\mathsf{e}}{\rightarrow}_{\sigma} \subseteq \underset{\mathsf{e}}{\rightarrow}_{\sigma} \cdot \rightarrow_{\sigma}^{=}$. We conclude by Lemma 3.5.2.
2. This is Lemma 7.1.1.
3. By definition of substitution. The immediate proof is in Appendix A.3. ◀

8 Non-Terminating Relations

In this section we provide examples of the fact that our technique does not rest on termination hypotheses. We consider *fixpoint operators* in both CbN and CbV, which have non-terminating reductions. Obviously, when terms are not restricted by types, the operator is definable, so the example is slight artificial, but we hope clarifying.

There are also cases where the modules are terminating but the compound system is not. The technique, surprisingly, still works. Accattoli gives various examples based on λ-calculi with explicit substitutions in [2]. An insight of this paper – not evident in [2] – is that *termination is not needed to lift factorization* from the modules to the compound system.

CbN Fixpoint, Head Factorization. We first consider the calculus $\beta Y := (\Lambda_Y, \rightarrow_\beta \cup \rightarrow_Y)$ [27][3]. It extends the CbN λ-calculus with a constant Y and a reduction $\rightarrow_Y$, the contextual closure of the rule $Yt \mapsto_Y t(Yt)$. Points 1-3 below are immediate – details in Appendix A.4.

▶ **Proposition 8.1** (Testing head factorization for βY). $\rightarrow_\beta \cup \rightarrow_Y$ *satisfies head factorization:*

1. Head factorization of $\rightarrow_Y$*:* $\texttt{Fact}(\underset{\mathsf{h}}{\rightarrow}_{\mathsf{Y}}, \underset{\neg\mathsf{h}}{\rightarrow}_{\mathsf{Y}})$.
2. Root linear swap*:* $\underset{\neg\mathsf{h}}{\rightarrow}_\beta \cdot \mapsto_Y \subseteq \underset{\mathsf{h}}{\rightarrow}_Y \cdot \rightarrow_\beta^*$.
3. Substitutivity*:* $\mapsto_Y$ *is substitutive.*

CbV Fixpoint, Weak Factorization. We now consider weak factorization and a CbV counterpart of the previous example. We follow Abramsky and McCusker [1], who study a call-by-value PCF with a fixpoint (more precisely, a recursion) operator Z. Similarly, we extend the CbV λ-calculus with a constant Z and its reduction $\rightarrow_Z$, which is the contextual closure of the rule $Zv \mapsto_Z \lambda x.v(Zv)x$ where v is a value. The calculus $\beta_v Z$ is therefore $(\Lambda_Z, \rightarrow_{\beta_v} \cup \rightarrow_Z)$. Points 1-3 below are all immediate – details in Appendix A.4.

▶ **Proposition 8.2** (Testing weak factorization for $\beta_v Z$). $\rightarrow_\beta \cup \rightarrow_Z$ *satisfies weak factorization:*

1. Weak factorization of $\rightarrow_Z$*:* $\texttt{Fact}(\underset{\mathsf{w}}{\rightarrow}_{\mathsf{Z}}, \underset{\neg\mathsf{w}}{\rightarrow}_{\mathsf{Z}})$.
2. Root linear swap*:* $\underset{\neg\mathsf{w}}{\rightarrow}_{\beta_v} \cdot \mapsto_Z \subseteq \underset{\mathsf{w}}{\rightarrow}_Z \cdot \rightarrow_{\beta_v}^*$.
3. Substitutivity*:* $\mapsto_Z$ *is substitutive.*

9 Further Applications: Probabilistic Calculi

In this paper, we present our technique using examples which are within the familiar language of λ-calculus. However the core of the technique – Theorem 3.4 – is independent from a specific syntax. It can be used in calculi whose objects are richer than λ-terms. The probabilistic λ-calculus is a prime example.

A recent line of research is developing probabilistic calculi where evaluation is not limited to a deterministic strategy. Faggian and Ronchi della Rocca [19] define two calculi – $\Lambda_\oplus^{\mathsf{cbv}}$ and $\Lambda_\oplus^{\mathsf{cbn}}$ – which model respectively CbV and CbN probabilistic higher-order computation, while being conservative extensions of the CbV and CbN λ-calculi. For both calculi confluence and factorization (called *standardization* in [19]) hold. There is however a deep *asymmetry* between the two results. *Confluence* is neatly proved via Hindley-Rosen technique, by relying on the fact the β and β_v reductions are confluent. The proof of *factorization* is instead laborious: the authors define a notion of parallel reduction for the new calculus, and then adapt Takahashi's technique [53]. Leventis work [36] on the call-by-name probabilistic λ-calculus suffers a similar problem. He proves factorization by relying on the finite developments method, but the proof is equally laborious.

Our technique allows for *a neat, concise proof of factorization*, which reduces to only testing a single linear swap, with no need for parallel reductions or finite developments. Proving factorization turns out to be in fact *easier* than proving confluence. The technical details – that is, the definition of the calculus and the proof – are in Appendix B.

[3] Head factorization of βY is easily obtained by a high-level argument, as consequence of left-normality, see Terese [54, Ch. 8.5]. The point that we want to stress here is that the validity of *linear swaps* is not limited to *terminating* reduction, and βY provides a simple, familiar example.

10 Conclusions and Discussions

Summary. A well-established approach to model higher-order computation with advanced features is to start from the call-by-name or call-by-value λ-calculus, and enrich it with new constructs. We propose a sharp technique to establish factorization of a compound system from factorization of its components. As we point out, the natural transposition of Hindley-Rosen technique for confluence does not work here, because the obtained conditions are – in general – not validated by extensions of the λ-calculus. The turning point is the identification of an alternative sufficient condition, called linear swap. Moreover, on common factorization schemes such as head or weak factorization, our technique reduces to a straightforward test. Concretely, we apply our technique to various examples, stressing its independence from common simplifying hypotheses such as confluence, orthogonality, and termination.

Black Box and Elementary Commutations. A key feature of our technique is to take factorization of the core relations – the modules – as *black boxes*. The focus is then on the analysis of the *interaction* between the modules. The benefit is both practical and conceptual: we disentangle the components – and the issues – under study. This is especially appealing when dealing with extensions of the λ-calculus, built on top of β or β_v reduction, because often most of the difficulties come from the higher-order component, that is, β or β_v itself – whose factorization is non-trivial but already proven – rather than from the added features.

Good illustrations of these points are our proofs of factorization. We stress that:

- the proof of factorization of the compound system is independent of the specific technique (finite developments, parallel reduction, etc.) used to prove factorization of the modules;
- to verify good interaction between the modules, it often suffices to check *elementary, local commutations* – the linear swaps.

These features provide a neat proof technique *supporting the development and the analysis* of complex compound systems.

Conclusions. When one wants to model new computational features, the calculus is often not given, but it has to be designed, in such a way that it satisfies confluence and factorization. The process of *developing* the calculus and the process of *proving* its good properties, go hand in hand. If the latter is difficult and prone to errors, the former also is. The black-box approach makes our technique efficient and accessible also to working scientists who are not specialists in rewriting. And even for the λ-calculus expert who masters tools such as finite developments, labeling or parallel reduction, it still appears desirable to limit the amount of difficulties. The more advanced and complex are the computational systems we study, the more crucial it is to have reasoning tools as simple to use as possible.

References

1 Samson Abramsky and Guy McCusker. Call-by-value games. In *Computer Science Logic, 11th International Workshop, CSL '97, Annual Conference*, volume 1414 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1997. doi:10.1007/BFb0028004.

2 Beniamino Accattoli. An abstract factorization theorem for explicit substitutions. In *23rd International Conference on Rewriting Techniques and Applications, RTA 2012*, volume 15 of *LIPIcs*, pages 6–21. Schloss Dagstuhl, 2012. doi:10.4230/LIPIcs.RTA.2012.6.

3 Beniamino Accattoli, Claudia Faggian, and Giulio Guerrieri. Factorization and normalization, essentially. In *Programming Languages and Systems - 17th Asian Symposium, APLAS 2019*, volume 11893 of *Lecture Notes in Computer Science*, pages 159–180. Springer, 2019. doi:10.1007/978-3-030-34175-6_9.

4 Beniamino Accattoli and Giulio Guerrieri. Open call-by-value. In *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016*, volume 10017 of *Lecture Notes in Computer Science*, pages 206–226. Springer, 2016. doi:10.1007/978-3-319-47958-3_12.

5 Yohji Akama. On Mints' reduction for ccc-calculus. In *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93*, volume 664 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1993. doi:10.1007/BFb0037094.

6 Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. In *22nd ACM Symposium on Principles of Programming Languages, POPL'95*, pages 233–246. ACM Press, 1995. doi:10.1145/199448.199507.

7 Pablo Arrighi and Gilles Dowek. Lineal: A linear-algebraic lambda-calculus. *Log. Methods Comput. Sci.*, 13(1), 2017. doi:10.23638/LMCS-13(1:8)2017.

8 Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. On probabilistic term rewriting. In *Functional and Logic Programming - 14th International Symposium, FLOPS 2018*, volume 10818 of *Lecture Notes in Computer Science*, pages 132–148. Springer, 2018. doi:10.1007/978-3-319-90686-7_9.

9 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998. doi:10.1017/CBO9781139172752.

10 Leo Bachmair and Nachum Dershowitz. Commutation, transformation, and termination. In *8th International Conference on Automated Deduction, CADE 1986*, volume 230 of *Lecture Notes in Computer Science*, pages 5–20. Springer, 1986. doi:10.1007/3-540-16780-3_76.

11 Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundations of mathematics.* North-Holland, 1984.

12 Frédéric Blanqui. Size-based termination of higher-order rewriting. *J. Funct. Program.*, 28:e11, 2018. doi:10.1017/S0956796818000072.

13 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Observability = typability + inhabitation. *CoRR*, 2018. URL: http://arxiv.org/abs/1812.06009.

14 Alberto Carraro and Giulio Guerrieri. A semantical and operational account of call-by-value solvability. In *Foundations of Software Science and Computation Structures, 17th International Conference, FoSSaCS 2014*, volume 8412 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 2014. doi:10.1007/978-3-642-54830-7_7.

15 Haskell B. Curry and Robert Feys. *Combinatory Logic, Volume 1.* Studies in logic and the foundations of mathematics. North-Holland, 1958.

16 Ugo de' Liguoro and Adolfo Piperno. Non deterministic extensions of untyped lambda-calculus. *Inf. Comput.*, 122(2):149–177, 1995. doi:10.1006/inco.1995.1145.

17 Nachum Dershowitz. On lazy commutation. In O. Grumberg, M. Kaminski, S. Katz, and S. Wintner, editors, *Languages: From Formal to Natural, Essays Dedicated to Nissim Francez on the Occasion of His 65th Birthday*, pages 59–82. Springer, 2009. doi:10.1007/978-3-642-01748-3_5.

18 Henk Doornbos and Burghard von Karger. On the union of well-founded relations. *Logic Journal of the IGPL*, 6(2):195–201, 1998. doi:10.1093/jigpal/6.2.195.

19 Claudia Faggian and Simona Ronchi Della Rocca. Lambda calculus and probabilistic computation. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019*, pages 1–13. IEEE Computer Society, 2019. doi:10.1109/LICS.2019.8785699.

20 Alfons Geser. *Relative Termination.* PhD thesis, University of Passau, Germany, 1990. URL: http://vts.uni-ulm.de/docs/2012/8146/vts_8146_11884.pdf.

21 Georges Gonthier, Jean-Jacques Lévy, and Paul-André Melliès. An abstract standardisation theorem. In *Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS '92)*, pages 72–81. IEEE Computer Society, 1992. doi:10.1109/LICS.1992.185521.

22 Bernhard Gramlich. Modularity in term rewriting revisited. *Theor. Comput. Sci.*, 464:3–19, 2012. doi:10.1016/j.tcs.2012.09.008.

23 Giulio Guerrieri. Head reduction and normalization in a call-by-value lambda-calculus. In *2nd International Workshop on Rewriting Techniques for Program Transformations and*

Evaluation, WPTE 2015, volume 46 of *OASICS*, pages 3–17. Schloss Dagstuhl, 2015. doi:10.4230/OASIcs.WPTE.2015.3.

24 Giulio Guerrieri, Luca Paolini, and Simona Ronchi Della Rocca. Standardization of a call-by-value lambda-calculus. In *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015*, volume 38 of *LIPIcs*, pages 211–225. Schloss Dagstuhl, 2015. doi:10.4230/LIPIcs.TLCA.2015.211.

25 Giulio Guerrieri, Luca Paolini, and Simona Ronchi Della Rocca. Standardization and conservativity of a refined call-by-value lambda-calculus. *Logical Methods in Computer Science*, 13(4), 2017. doi:10.23638/LMCS-13(4:29)2017.

26 J. Roger Hindley. *The Church-Rosser Property and a Result in Combinatory Logic.* PhD thesis, University of Newcastle-upon-Tyne, 1964.

27 J. Roger Hindley. Reductions of residuals are finite. *Transactions of the American Mathematical Society*, 240:345–361, 1978.

28 J. Roger Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators: An Introduction.* Cambridge University Press, New York, NY, USA, 2 edition, 2008.

29 Nao Hirokawa, Aart Middeldorp, and Georg Moser. Leftmost outermost revisited. In *26th International Conference on Rewriting Techniques and Applications, RTA 2015*, volume 36 of *LIPIcs*, pages 209–222. Schloss Dagstuhl, 2015. doi:10.4230/LIPIcs.RTA.2015.209.

30 Katsumasa Ishii. A proof of the leftmost reduction theorem for $\lambda\beta\eta$-calculus. *Theor. Comput. Sci.*, 747:26–32, 2018. doi:10.1016/j.tcs.2018.06.003.

31 Jan Willem Klop. *Combinatory Reduction Systems.* Phd thesis, Mathematisch Centrum, Amsterdam, 1980.

32 Masahito Kurihara and Ikuo Kaji. Modular term rewriting systems and the termination. *Inf. Process. Lett.*, 34(1):1–4, 1990. doi:10.1016/0020-0190(90)90221-I.

33 Ugo Dal Lago and Simone Martini. The weak lambda calculus as a reasonable machine. *Theor. Comput. Sci.*, 398(1-3):32–50, 2008. doi:10.1016/j.tcs.2008.01.044.

34 Ugo Dal Lago and Margherita Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO Theor. Informatics Appl.*, 46(3):413–450, 2012. doi:10.1051/ita/2012012.

35 Xavier Leroy. The ZINC experiment: an economical implementation of the ML language. Technical report 117, INRIA, 1990. URL: http://gallium.inria.fr/~xleroy/publi/ZINC.pdf.

36 Thomas Leventis. A deterministic rewrite system for the probabilistic λ-calculus. *Math. Struct. Comput. Sci.*, 29(10):1479–1512, 2019. doi:10.1017/S0960129519000045.

37 Jean-Jacques Lévy. *Réductions corrcectes et optimales dans le lambda calcul.* PhD thesis, University of Paris 7, 1978.

38 Paul-André Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In *Typed Lambda Calculi and Applications, Second International Conference on Typed Lambda Calculi and Applications, TLCA '95*, volume 902 of *Lecture Notes in Computer Science*, pages 328–334. Springer, 1995. doi:10.1007/BFb0014062.

39 Paul-André Melliès. A factorisation theorem in rewriting theory. In *Category Theory and Computer Science, 7th International Conference, CTCS '97*, volume 1290 of *Lecture Notes in Computer Science*, pages 49–68. Springer, 1997. doi:doi.org/10.1007/BFb0026981.

40 Aart Middeldorp. Modular aspects of properties of term rewriting systems related to normal forms. In *Rewriting Techniques and Applications, 3rd International Conference, RTA-89*, volume 355 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1989. doi:10.1007/3-540-51081-8_113.

41 Aart Middeldorp. A sufficient condition for the termination of the direct sum of term rewriting systems. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89)*, pages 396–401. IEEE Computer Society, 1989. doi:10.1109/LICS.1989.39194.

42 Aart Middeldorp. Confluence of the disjoint union of conditional term rewriting systems. In *Conditional and Typed Rewriting Systems, 2nd International Workshop, CTRS 1990*,

volume 516 of *Lecture Notes in Computer Science*, pages 295–306. Springer, 1990. doi:10.1007/3-540-54317-1_99.

43 Gerd Mitschke. The standardization theorem for λ-calculus. *Mathematical Logic Quarterly*, 25(1-2):29–31, 1979. doi:10.1002/malq.19790250104.

44 Luca Paolini. Call-by-value separability and computability. In *Theoretical Computer Science, 7th Italian Conference, ICTCS 2001*, volume 2202 of *Lecture Notes in Computer Science*, pages 74–89. Springer, 2001. doi:10.1007/3-540-45446-2_5.

45 Luca Paolini and Simona Ronchi Della Rocca. Call-by-value solvability. *RAIRO Theor. Informatics Appl.*, 33(6):507–534, 1999. doi:10.1051/ita:1999130.

46 Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

47 Laurent Regnier. Une équivalence sur les lambda-termes. *Theor. Comput. Sci.*, 126(2):281–292, 1994. doi:10.1016/0304-3975(94)90012-4.

48 György E. Révész. A list-oriented extension of the lambda-calculus satisfying the Church-Rosser theorem. *Theor. Comput. Sci.*, 93(1):75–89, 1992. doi:10.1016/0304-3975(92)90212-X.

49 Simona Ronchi Della Rocca and Luca Paolini. *The Parametric Lambda Calculus - A Metamodel for Computation*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-10394-4.

50 J. Barkley Rosser. A mathematical logic without variables. *Duke Mathematical Journal*, 1(3):328–355, September 1935. doi:10.1215/S0012-7094-35-00123-5.

51 Michaël Rusinowitch. On termination of the direct sum of term-rewriting systems. *Inf. Process. Lett.*, 26(2):65–70, 1987. doi:10.1016/0020-0190(87)90039-1.

52 Alexis Saurin. On the relations between the syntactic theories of lambda-mu-calculi. In *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference*, volume 5213 of *Lecture Notes in Computer Science*, pages 154–168. Springer, 2008. doi:10.1007/978-3-540-87531-4_13.

53 Masako Takahashi. Parallel reductions in lambda-calculus. *Inf. Comput.*, 118(1):120–127, 1995. doi:10.1006/inco.1995.1057.

54 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

55 Yoshihito Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Inf. Process. Lett.*, 25(3):141–143, 1987. doi:10.1016/0020-0190(87)90122-0.

56 Yoshihito Toyama. On the church-rosser property for the direct sum of term rewriting systems. *J. ACM*, 34(1):128–143, 1987. doi:10.1145/7531.7534.

57 Yoshihito Toyama, Jan Willem Klop, and Hendrik Pieter Barendregt. Termination for the direct sum of left-linear term rewriting systems -preliminary draft-. In *Rewriting Techniques and Applications, 3rd International Conference, RTA-89*, volume 355 of *Lecture Notes in Computer Science*, pages 477–491. Springer, 1989. doi:10.1007/3-540-51081-8_127.

58 Vincent Van Oostrom. Some symmetries of commutation diamonds. Talk at the International Workshop on Confluence, 30 June 2020.

59 Vincent van Oostrom. Confluence by decreasing diagrams. In *Rewriting Techniques and Applications, 19th International Conference, RTA 2008,*, volume 5117 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2008. doi:10.1007/978-3-540-70590-1_21.

60 Vincent van Oostrom and Yoshihito Toyama. Normalisation by random descent. In *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016*, volume 52 of *LIPIcs*, pages 32:1–32:18. Schloss Dagstuhl, 2016. doi:10.4230/LIPIcs.FSCD.2016.32.

61 Vincent van Oostrom and Hans Zantema. Triangulation in rewriting. In *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012*, volume 15 of *LIPIcs*, pages 240–255. Schloss Dagstuhl, 2012. doi:10.4230/LIPIcs.RTA.2012.240.

APPENDIX

Appendix A collects omitted details of proofs. In Appendix B we illustrate a more *advanced example* of application of our technique, namely to the *probabilistic λ-calculus.*

A Appendix: Omitted Proofs

A.1 Head Factorization (Sect. 4)

Consequences of Property 4.2. Note that the empty context $\langle\,\rangle$ is a *head context.* Hence, for a non-head step $\mathsf{C}\langle r\rangle \underset{\neg\mathsf{h}}{\to} \mathsf{C}\langle r'\rangle$, necessarily $\mathsf{C} \neq \langle\,\rangle$, and Property 4.2 applies. Consequently:

▶ **Property A.1** (Shape preservation). *For any γ-rule, $\underset{\neg\mathsf{h}}{\to}_\gamma$ preserves the shapes of terms:*

1. Atom*: there is no t such that $t \underset{\neg\mathsf{h}}{\to}_\gamma a$, for any variable or constant a.*
2. Abstraction*: $t \underset{\neg\mathsf{h}}{\to}_\gamma \lambda x.u_1$ implies $t = \lambda x.t_1$ and $t_1 \underset{\neg\mathsf{h}}{\to}_\gamma u_1$.*
3. Application*: $t \underset{\neg\mathsf{h}}{\to}_\gamma u_1u_2$ implies $t = t_1t_2$ with: either $t_2 \to_\gamma u_2$ and $t_1 = u_1$; or $t_1 \to_\gamma u_1$ and $t_2 = u_2$, and if moreover $t_1 \underset{\mathsf{h}}{\to}_\gamma u_1$ then t_1 and u_1 are abstractions.*
4. Redex*: if $t \underset{\neg\mathsf{h}}{\to}_\gamma u$, and u is a β-redex, then t is a β-redex.*

   Similarly, if $t \underset{\neg\mathsf{h}}{\to}_\gamma u$ and u has shape $ct_1 \dots t_k$ (c is a constant), t has the same shape.

Point 4. follows from points 1. to 3. Note that, in particular, if u is a $\oplus$-redex, so is t.

Head Factorization, Modularly.

▶ **Lemma** (4.3 Lifting root linear swaps). *Let $\mapsto_\alpha$ and $\mapsto_\gamma$ be root relations on Λ.*

1. $\underset{\neg\mathsf{h}}{\to}_\alpha \cdot \mapsto_\gamma \subseteq \underset{\mathsf{h}}{\to}_\gamma \cdot \to_\alpha^*$ *implies* $\mathtt{lSwap}(\underset{\neg\mathsf{h}}{\to}_\alpha, \underset{\mathsf{h}}{\to}_\gamma)$.
2. *Similarly,* $\underset{\neg\mathsf{h}}{\to}_\alpha \cdot \mapsto_\gamma \subseteq \underset{\mathsf{h}}{\to}_\gamma \cdot \to_\alpha^=$ *implies* $\underset{\neg\mathsf{h}}{\to}_\alpha \cdot \to_\gamma \subseteq \underset{\mathsf{h}}{\to}_\gamma \cdot \to_\alpha^=$.

Proof.

1. We prove that $t \underset{\neg\mathsf{h}}{\to}_\alpha u \underset{\mathsf{h}}{\to}_\gamma s$ implies $t \underset{\mathsf{h}}{\to}_\gamma \cdot \to_\alpha^* s$ by induction on the head context $\mathsf{H} = \lambda x_1 \dots \lambda x_k.\langle\,\rangle t_1 \dots t_n$ of the reduction step $u \underset{\mathsf{h}}{\to}_\gamma s$. Cases:
   - **a.** u is a γ-redex (*i.e.* $\mathsf{H} = \langle\,\rangle$, $k = 0$, $n = 0$) and hence $t \underset{\neg\mathsf{h}}{\to}_\alpha u \mapsto_\gamma s$. Thus, $t \underset{\mathsf{h}}{\to}_\gamma \cdot \to_\alpha^* s$ by the hypothesis $\underset{\neg\mathsf{h}}{\to}_\alpha \cdot \mapsto_\gamma \subseteq \underset{\mathsf{h}}{\to}_\gamma \cdot \to_\alpha^*$.
   - **b.** $u = \lambda x.u_1$ (*i.e.* $k > 0$) with $u_1 \underset{\mathsf{h}}{\to}_\gamma s_1$ and $u = \lambda x.u_1 \underset{\mathsf{h}}{\to}_\gamma \lambda x.s_1 = s$. By shape preservation (Property A.1.2) $t = \lambda x.t_1$ and $t_1 \underset{\neg\mathsf{h}}{\to}_\alpha u_1$. By *i.h.*, we can conclude.
   - **c.** $u = u_1u_2$ (*i.e.*, $k = 0$, $n > 0$) with $u_1 \underset{\mathsf{h}}{\to}_\gamma u_1'$ and $u = u_1u_2 \underset{\mathsf{h}}{\to}_\gamma u_1'u_2 = s$. By shape preservation (Property A.1.3), there are two sub-cases for $t \underset{\neg\mathsf{h}}{\to}_\alpha u_1u_2$.
     - **i.** $t = t_1u_2$ and $t_1 \to_\alpha u_1$. Note that it impossible that $t_1 \underset{\mathsf{h}}{\to}_\alpha u_1$, otherwise u_1 would be an abstraction (by Property A.1.3) and this would contradict $u_1u_2 \underset{\mathsf{h}}{\to}_\gamma u_1'u_2$. Therefore, $t_1 \underset{\mathsf{h}}{\to}_\alpha u_1$. By *i.h.*, $t_1 \underset{\mathsf{h}}{\to}_\gamma t_1' \to_\alpha^* u_1'$ so $t = t_1u_2 \underset{\mathsf{h}}{\to}_\gamma t_1'u_2 \to_\alpha^* u_1'u_2 = s$.
     - **ii.** $t = u_1t_2$ and $t_2 \to_\alpha u_2$. Then, $t = u_1t_2 \underset{\mathsf{h}}{\to}_\gamma u_1't_2 \to_\alpha u_1'u_2 = s$.
2. The proof is similar to Point 1. ◀

▶ **Lemma** (4.4 Swap with $\underset{\mathsf{h}}{\to}_\beta$). *If $\mapsto_\gamma$ is substitutive then* $\mathtt{lSwap}(\underset{\neg\mathsf{h}}{\to}_\gamma, \underset{\mathsf{h}}{\to}_\beta)$ *holds.*

Proof. By Lemma 4.3, it is enough to prove that $t \underset{\neg\mathsf{h}}{\to}_\gamma u \mapsto_\beta s$ implies $t \underset{\mathsf{h}}{\to}_\beta \cdot \to_\gamma^* s$. According to Property 4.1.1, $\to_\gamma$ is substitutive. Let $u = (\lambda x.u_1)u_2 \mapsto_\beta u_1\{x{:=}u_2\} = s$. By shape preservation (Property A.1) applied to $t \underset{\neg\mathsf{h}}{\to}_\gamma u$, there are only two cases:

1. either $t = (\lambda x.p)u_2$ and $p \to_\gamma u_1$, so $t \underset{\mathsf{h}}{\to}_\beta p\{x{:=}u_2\} \to_\gamma u_1\{x{:=}u_2\}$ as $\to_\gamma$ is substitutive;
2. or $t = (\lambda x.u_1)q$ and $q \to_\gamma u_2$, so $t \underset{\mathsf{h}}{\to}_\beta u_1\{x{:=}q\} \to_\gamma^* u_1\{x{:=}u_2\}$ by Property 4.1.2. ◀

A.2 Call-by-Value λ-Calculus (Sect. 6)

Consequences of Property 4.2. The empty context $\langle\,\rangle$ is *both a left and a weak context.* Hence, Property 4.2 always applies to non-left and non-weak steps. Consequently:

▶ **Property A.2** (Shape preservation). *For any* $\mathsf{e} \in \{\mathsf{l}, \mathsf{w}\}$, $\underset{\neg\mathsf{e}}{\to}_\gamma$ *preserves the shape of terms:*

1. Atom*: there is no t such that $t \underset{\neg\mathsf{e}}{\to}_\gamma a$, for any variable or constant a.*
2. Abstraction*: $t \underset{\neg\mathsf{e}}{\to}_\gamma \lambda x.u_1$ implies $t = \lambda x.t_1$ and $t_1 \to_\gamma u_1$.*
3. Application*: $t \underset{\neg\mathsf{e}}{\to}_\gamma u_1u_2$ implies $t = t_1t_2$, with either (i) $t_1 \underset{\neg\mathsf{e}}{\to}_\gamma u_1$ and $t_2 = u_2$, or (ii) $t_2 \to_\gamma u_2$ and $t_1 = u_1$. Moreover, in case (ii): if $vt_2 \underset{\neg\mathsf{l}}{\to} vu_2$ (v is a value), then $t_2 \underset{\neg\mathsf{l}}{\to} u_2$; if $t_1t_2 \underset{\neg\mathsf{w}}{\to}_\gamma u_1u_2$, then $t_2 \underset{\neg\mathsf{w}}{\to}_\gamma u_2$.*
4. Redex*: if $t \underset{\neg\mathsf{e}}{\to}_\gamma u$, and u is a β_v-redex, then t also is (as a consequence of points 1. to 3.).*

Left and Weak Factorization, Modularly. To prove Proposition 6.1 we proceed similarly to Sect. 4.2.

▶ **Lemma A.3** (Lifting root linear swaps). *Let $\mapsto_\alpha$ and $\mapsto_\gamma$ be root relations on Λ.*

1. *If $\underset{\neg\mathsf{l}}{\to}_\alpha \cdot \mapsto_\gamma \subseteq \underset{\mathsf{l}}{\to}_\gamma \cdot \to_\alpha^*$ then* $\mathtt{lSwap}(\underset{\neg\mathsf{l}}{\to}_\alpha, \underset{\mathsf{l}}{\to}_\gamma)$.
2. *If $\underset{\neg\mathsf{w}}{\to}_\alpha \cdot \mapsto_\gamma \subseteq \underset{\mathsf{w}}{\to}_\gamma \cdot \to_\alpha^*$ then* $\mathtt{lSwap}(\underset{\neg\mathsf{w}}{\to}_\alpha, \underset{\mathsf{w}}{\to}_\gamma)$.
3. *Similarly, $\underset{\neg\mathsf{e}}{\to}_\alpha \cdot \mapsto_\gamma \subseteq \underset{\mathsf{e}}{\to}_\gamma \cdot \to_\alpha^=$ implies $\underset{\neg\mathsf{e}}{\to}_\alpha \cdot \to_\gamma \subseteq \underset{\mathsf{e}}{\to}_\gamma \cdot \to_\alpha^=$, with* $\mathsf{e} \in \{\mathsf{l}, \mathsf{w}\}$

Proof.

1. We prove that $t \underset{\neg\mathsf{l}}{\to}_\alpha u \underset{\mathsf{l}}{\to}_\gamma s$ implies $t \underset{\mathsf{l}}{\to}_\gamma \cdot \to_\alpha^* s$, by induction on the context L of $u \underset{\mathsf{l}}{\to}_\gamma s$.
   - **a.** $\mathsf{L} = \langle\,\rangle$, *i.e.* $u \mapsto_\gamma s$. The claim holds by hypothesis.
   - **b.** $\mathsf{L} = \mathsf{L}'u_2$, *i.e.* $u = u_1u_2 \underset{\mathsf{l}}{\to}_\gamma s_1u_2 = s$ with $u_1 \underset{\mathsf{l}}{\to}_\gamma s_1$. By Property A.2.3, $t = t_1t_2$ and
     - **i.** either $t_2 \to_\alpha u_2$ (and $t_1 = u_1$), so $t = u_1t_2 \underset{\mathsf{l}}{\to}_\gamma s_1t_2 \to_\alpha s_1u_2 = s$;
     - **ii.** or $t_1 \underset{\neg\mathsf{l}}{\to}_\alpha u_1$ (and $t_2 = u_2$); by *i.h.*, $t_1 \underset{\mathsf{l}}{\to}_\gamma \cdot \to_\alpha^* s_1$, so $t = t_1u_2 \underset{\mathsf{l}}{\to}_\gamma \cdot \to_\alpha^* s_1u_2 = s$.
   - **c.** $\mathsf{L} = v\mathsf{L}'$, *i.e.* $u = vu_2 \underset{\mathsf{l}}{\to}_\gamma vs_2 = s$ with $u_2 \underset{\mathsf{l}}{\to}_\gamma s_2$. By Property A.2.3, $t = t_1t_2$ and
     - **i.** either $t_2 \underset{\neg\mathsf{l}}{\to}_\alpha u_2$ (and $t_1 = v$); by *i.h.*, $t_2 \underset{\mathsf{l}}{\to}_\gamma \cdot \to_\alpha^* s_2$, so $t = vt_2 \underset{\mathsf{l}}{\to}_\gamma \cdot \to_\alpha^* vs_2 = s$;
     - **ii.** or $t_1 \underset{\neg\mathsf{l}}{\to}_\alpha v$ (and $t_2 = u_2$); since v is a value, by Property A.2.1-2, t_1 is also a value and so $t = t_1u_2 \underset{\mathsf{l}}{\to}_\gamma t_1s_2 \to_\alpha vs_2 = s$.
2. The proof is similar Point 1, but simpler. Case $\mathsf{W} = \langle\,\rangle$ is the same. Case $\mathsf{W} = \mathsf{W}'u_2$ is exactly like $\mathsf{L} = \mathsf{L}'u_2$, and case $\mathsf{W} = u_1\mathsf{W}'$ is symmetric to case $\mathsf{W} = \mathsf{W}'u_2$.
3. The proof is similar to Points 1-2. ◀

▶ **Lemma A.4** (Swap with $\underset{\mathsf{e}}{\to}_{\beta_v}$). *If $\mapsto_\gamma$ is substitutive then* $\mathtt{lSwap}(\underset{\neg\mathsf{e}}{\to}_\gamma, \underset{\mathsf{e}}{\to}_{\beta_v})$, *for* $\mathsf{e} \in \{\mathsf{l}, \mathsf{w}\}$.

Proof. We prove $\mathtt{lSwap}(\underset{\neg\mathsf{l}}{\to}_\gamma, \underset{\mathsf{l}}{\to}_{\beta_v})$, the other swap is similar. By Lemma A.3, it is enough to prove that $t \underset{\neg\mathsf{l}}{\to}_\gamma u \mapsto_{\beta_v} s$ implies $t \underset{\mathsf{l}}{\to}_{\beta_v} \cdot \to_\gamma^* s$. According to Property 4.1.1, $\to_\gamma$ is substitutive. Let $u = (\lambda x.u_1)v \mapsto_{\beta_v} u_1\{x{:=}v\} = s$. By shape preservation (Property A.2) applied to $t \underset{\neg\mathsf{l}}{\to}_\gamma u$, there are only two cases:

1. either $t = (\lambda x.t_1)v$ and $t_1 \to_\gamma u_1$, so $t \underset{\mathsf{l}}{\to}_{\beta_v} t_1\{x{:=}v\} \to_\gamma u_1\{x{:=}v\}$ as $\to_\gamma$ is substitutive;
2. or $t = (\lambda x.u_1)w$ where $w \to_\gamma v$ and w is a value; hence, $t = (\lambda x.u_1)w \underset{\mathsf{l}}{\to}_{\beta_v} u_1\{x{:=}w\} \to_\gamma^* u_1\{x{:=}v\}$, where the $\to_\gamma$ steps take place by Property 4.1.2. ◀

A.3 The Shuffling Calculus (Sect. 7)

▶ **Property A.5** (Values are closed under substitution). *If v and w are values, so is $v\{x{:=}w\}$.*

▶ **Lemma** (7.1 Root linear swaps). *Let $\mathsf{e} \in \{\mathsf{l}, \mathsf{w}\}$ and $i \in \{1, 3\}$. Then:*

1. $\underset{\neg\mathsf{e}}{\to}_{\beta_v} \cdot \mapsto_{\sigma_i} \subseteq \mapsto_{\sigma_i} \cdot \to_{\beta_v}$.
2. $\underset{\neg\mathsf{e}}{\to}_{\sigma} \cdot \mapsto_{\sigma_i} \subseteq \mapsto_{\sigma_i} \cdot \to_{\sigma}$.

Proof. We prove the following, more general, statement: Let $\mathsf{e} \in \{\mathsf{l}, \mathsf{w}\}$ and $i \in \{1, 3\}$, let $\to_\gamma$ be the contextual closure of *any* rule $\mapsto_\gamma$ on Λ. Then, $t \underset{\neg\mathsf{e}}{\to}_\gamma u \mapsto_{\sigma_i} s \subseteq t \mapsto_{\sigma_i} \cdot \to_\gamma s$.

First, let us consider $\mathsf{e} = \mathsf{l}$. We examine the two cases for $u \mapsto_{\sigma_i} s$ ($i = 1$ or $i = 3$).

σ_1: By hypothesis, $u = (\lambda x.q)pr \mapsto_{\sigma_1} (\lambda x.qr)p = s$ with $x \notin \mathsf{fv}(r)$. We can assume $x \notin \mathsf{fv}(t)$. Since $t \underset{\neg\mathsf{l}}{\to}_\gamma u$, by Property A.2 (iterated), we have $t = (\lambda x.q')p'r'$ and moreover:

- either $q' \to_\gamma q$ and $r' = r$ and $p' = p$,
- or $r' \to_\gamma r$ and $q' = q$ and $p' = p$,
- or $p' \underset{\neg\mathsf{l}}{\to}_\gamma p$ and $q' = q$ and $r' = r$.

In any case, $t = (\lambda x.q')p'r' \mapsto_{\sigma_1} (\lambda x.q'r')p' \underset{\neg\mathsf{l}}{\to}_\gamma (\lambda x.qr)p = s$, since $x \notin \mathsf{fv}(t) \supseteq \mathsf{fv}(r')$.

σ_3: By hypothesis, $u = v((\lambda x.r)p) \mapsto_{\sigma_3} (\lambda x.vr)p = s$ with $x \notin \mathsf{fv}(v)$. We can assume $x \notin \mathsf{fv}(t)$. As $t \underset{\neg\mathsf{l}}{\to}_\gamma u$, by Property A.2 (iterated), we have $t = v'((\lambda x.r')p')$ and moreover:

- either $v' \underset{\neg\mathsf{l}}{\to}_\gamma v$ and $r' = r$ and $p' = p$,
- or $r' \to_\gamma r$ and $v' = v$ and $p' = p$,
- or $p' \underset{\neg\mathsf{l}}{\to}_\gamma p$ and $v' = v$ and $r' = r$.

In any case, $t = v'((\lambda x.r')p') \mapsto_{\sigma_3} (\lambda x.v'r')p' \underset{\neg\mathsf{l}}{\to}_\gamma (\lambda x.vr)p = s$, as $x \notin \mathsf{fv}(t) \supseteq \mathsf{fv}(v')$.

As usual, the proof in the case $\mathsf{e} = \mathsf{w}$ is similar, and simpler. ◀

▶ **Lemma A.6** (Substitutivity of $\to_\sigma$). *If $t \mapsto_{\sigma_i} t'$ then $t\{x{:=}v\} \mapsto_{\sigma_i} t'\{x{:=}v\}$, for $i \in \{1, 3\}$.*

Proof. σ_1: $t = (\lambda y.r)su \mapsto_{\sigma_1} (\lambda x.ru)s = t'$ with $y \notin \mathsf{fv}(u)$ and we can suppose without loss of generality that $y \notin \mathsf{fv}(v) \cup \{x\}$. Therefore, $t\{x{:=}v\} = (\lambda y.r\{x{:=}v\})s\{x{:=}v\}u\{x{:=}v\} \mapsto_{\sigma_1} (\lambda y.r\{x{:=}v\}u\{x{:=}v\})s\{x{:=}v\} = t'\{x{:=}v\}$ since $y \notin (\mathsf{fv}(u) \setminus \{x\}) \cup \mathsf{fv}(v) = \mathsf{fv}(u\{x{:=}v\})$.

σ_3: $t = w((\lambda y.u)s) \mapsto_{\sigma_3} (\lambda y.wu)s = t'$ with $y \notin \mathsf{fv}(w)$ and we can suppose without loss of generality that $y \notin \mathsf{fv}(v) \cup \{x\}$. Therefore, $t\{x{:=}v\} = w((\lambda y.u\{x{:=}v\})s\{x{:=}v\}) \mapsto_{\sigma_3} (\lambda y.w\{x{:=}v\}u\{x{:=}v\})s\{x{:=}v\} = t'\{x{:=}v\}$ as $w\{x{:=}v\}$ is a value (Property A.5) and $y \notin (\mathsf{fv}(w) \setminus \{x\}) \cup \mathsf{fv}(v) = \mathsf{fv}(w\{x{:=}v\})$. ◀

A.4 Non-Terminating Relations (Sect. 8)

▶ **Proposition** (8.1 Testing head factorization for βY). *$\to_\beta \cup \to_Y$ satisfies head factorization:*

1. Head factorization of $\to_Y$*:* $\mathtt{Fact}(\underset{\mathsf{h}}{\to}_{\mathsf{Y}}, \underset{\neg\mathsf{h}}{\to}_{\mathsf{Y}})$.
2. Root linear swap*:* $\underset{\neg\mathsf{h}}{\to}_\beta \cdot \mapsto_Y \subseteq \underset{\mathsf{h}}{\to}_Y \cdot \to_\beta^*$.
3. Substitutivity*: $\mapsto_Y$ is substitutive.*

Proof. We verify the hypotheses of Proposition 4.5:

1. To verify that the reduction $\to_Y$ satisfies head factorization is routine.
2. Assume $t \underset{\neg\mathsf{h}}{\to}_\beta Yp \mapsto_Y p(Yp)$. By Property 4.2 (as spelled out in Property A.1), if $t \underset{\neg\mathsf{h}}{\to}_\beta Yp$ then $t = Yq$ and $q \to_\beta p$. Hence $t = Yq \underset{\mathsf{h}}{\to}_Y q(Yq) \to_\beta^* p(Yp)$.
3. Simply $(Yp)\{x{:=}q\} = Y(p\{x{:=}q\}) \mapsto_Y (p\{x{:=}q\})(Y(p\{x{:=}q\})) = (p(Yp))\{x{:=}q\}$. ◀

▶ **Proposition** (8.2 Testing weak factorization for $\beta_v Z$). $\rightarrow_\beta \cup \rightarrow_Z$ *satisfies weak factorization:*

1. Weak factorization of $\rightarrow_Z$*:* $\mathtt{Fact}(\underset{\mathsf{w}}{\rightarrow}_{\mathsf{Z}}, \underset{\neg\mathsf{w}}{\rightarrow}_{\mathsf{Z}})$.
2. Root linear swap*:* $\underset{\neg\mathsf{w}}{\rightarrow}_{\beta_v} \cdot \mapsto_Z \subseteq \underset{\mathsf{w}}{\rightarrow}_Z \cdot \rightarrow^*_{\beta_v}$.
3. Substitutivity*:* $\mapsto_Z$ *is substitutive.*

Proof. We prove the hypotheses of Proposition 4.5:

1. It is easy to verify that $\underset{\neg\mathsf{w}}{\rightarrow}_Z \cdot \underset{\mathsf{w}}{\rightarrow}_Z \subseteq \underset{\mathsf{w}}{\rightarrow}_Z \cdot \underset{\neg\mathsf{w}}{\rightarrow}_Z{}^*$. Then apply Lemma 3.5.1.
2. Assume $t \underset{\neg\mathsf{w}}{\rightarrow}_{\beta_v} Zv \mapsto_Z \lambda x.v(Zv)x$. By Property 4.2 (as spelled-out in Property A.2), if $t \underset{\neg\mathsf{w}}{\rightarrow}_{\beta_v} Zv$ then $t = Zw$ and $w \rightarrow_{\beta_v} v$. So, $t = Zw \underset{\mathsf{w}}{\rightarrow}_Z \lambda x.w(Zw)x \rightarrow^*_{\beta_v} \lambda x.v(Zv)x$.
3. Simply $(Zv)\{x{:=}q\} = Z(v\{x{:=}q\}) \mapsto_Z \lambda y.v\{x{:=}q\}(Z(v\{x{:=}q\})y = (\lambda y.v(Zv)y)\{x{:=}q\}$.

◀

B Appendix: Factorizing Factorization in Probabilistic λ-calculus

Faggian and Ronchi della Rocca [19] define two calculi – $\Lambda_\oplus^{\mathsf{cbn}}$ and $\Lambda_\oplus^{\mathsf{cbv}}$ – which model respectively CbV and CbN probabilistic higher-order computation, and are conservative extensions of the CbN and CbV λ-calculi. Here we focus on CbV, which is the most relevant paradigm for calculi with effects, but the same approach applies to CbN.

We first recall the syntax of $\Lambda_\oplus^{\mathsf{cbv}}$ (we refer to [19] for background and details), and then give a new proof of weak factorization, using our technique and obtaining a neat, compact proof of factorization, which only requires a few lines.

Terms. $\Lambda_\oplus^{\mathsf{cbv}}$ is a rewrite system where the objects to be rewritten are not terms, but monadic structures on terms, namely multi-distributions [8]. Intuitively, a multi-distribution represents a probability distribution on the possible reductions from a term. Terms and contexts are the same as for the non-deterministic λ-calculus, but here we write the $\oplus$ infix, to facilitate reference to [19]. *Terms and values* are generated by the grammars

$$M ::= x \mid \lambda x.M \mid MM \mid M \oplus M \qquad \textbf{(Terms } \Lambda_\oplus\textbf{)}$$
$$V ::= x \mid \lambda x.M \qquad \textbf{(Values } \mathcal{V}\textbf{)}$$

where x ranges over a countable set of *variables*. *Contexts* and *weak contexts* are given by:

$$\mathsf{C} ::= \langle\ \rangle \mid \mathsf{C}M \mid M\mathsf{C} \mid \lambda x.\mathsf{C} \mid \mathsf{C} \oplus M \mid M \oplus \mathsf{C} \qquad \textbf{(Contexts)}$$
$$\mathsf{W} ::= \langle\ \rangle \mid \mathsf{W}M \mid M\mathsf{W} \qquad \textbf{(Weak Contexts)}$$

where $\langle\ \rangle$ denotes the *hole* of the context.

The intended behaviour of $M \oplus N$ is to reduce to either M or N, *with equal probability* $\frac{1}{2}$. This is formalized by means of *multi-distributions*.

Multi-distributions. A *multi-distribution* $\mathtt{m} = [p_i M_i \mid i \in I]$ is a multiset of pairs of the form pM, with $p \in]0,1]$, $M \in \Lambda_\oplus$, and $\sum p_i \leq 1$. We denote by $\mathcal{M}(\Lambda_\oplus)$ the set of all multi-distributions. The sum of multi-distributions is denoted by $+$. The product $q \cdot \mathtt{m}$ of a scalar q and a multi-distribution $\mathtt{m}$ is defined pointwise $q[p_i M_i]_{i\in I} := [(qp_i)M_i]_{i\in I}$.

$$\mathsf{C}\langle(\lambda x.M)V\rangle \to_{\beta_v} [\mathsf{C}\langle M\{x{:=}V\}\rangle] \qquad \mathsf{W}\langle M \oplus N\rangle \to_{\oplus} [\tfrac{1}{2}\mathsf{W}(M), \tfrac{1}{2}\mathsf{W}(N)] \qquad \begin{array}{l} \to := \to_{\beta_v} \cup \to_{\oplus} \\ \underset{\mathsf{w}}{\to} := \underset{\mathsf{w}}{\to}_{\beta_v} \cup \to_{\oplus} \end{array}$$

Figure 1 Reduction Steps.

$$\frac{}{[M] \Rightarrow_r [M]} \qquad \frac{M \to_r \mathtt{m}}{[M] \Rightarrow_r \mathtt{m}} \qquad \frac{([M_i] \Rightarrow_r \mathtt{m}_i)_{i\in I}}{[p_i M_i \mid i \in I] \Rightarrow_r \ +_{i\in I}\ p_i \cdot \mathtt{m}_i}$$

Figure 2 Lifting $\to_r$ to $\Rightarrow_r$.

The calculus $(\mathcal{M}(\Lambda_\oplus), \Rightarrow)$. The calculus $\Lambda_\oplus^{\mathsf{cbv}}$ is the rewrite system $(\mathcal{M}(\Lambda_\oplus), \Rightarrow)$ where $\mathcal{M}(\Lambda_\oplus)$ is the set of multi-distributions on $\Lambda_\oplus$ and the relation $\Rightarrow \subseteq \mathcal{M}(\Lambda_\oplus) \times \mathcal{M}(\Lambda_\oplus)$ is defined in Fig. 1 and Fig. 2. First, we define one-step reductions from terms to multi-distributions – so for example, $M \oplus N \to [\frac{1}{2}M, \frac{1}{2}N]$. Then, we lift the definition of reduction to a binary relation on $\mathcal{M}(\Lambda_\oplus)$, in the natural way – for instance $[\frac{1}{2}(\lambda x.x)z, \frac{1}{2}(M \oplus N)] \Rightarrow [\frac{1}{2}z, \frac{1}{4}M, \frac{1}{4}N]$. Precisely:

1. The reductions $\to_{\beta_v}, \to_\oplus \subseteq \Lambda_\oplus \times \mathcal{M}(\Lambda_\oplus)$ are defined in Fig. 1. Observe that the $\oplus$ rule – probabilistic choice – is closed only under weak contexts (no reduction in the body of a function nor in the scope of an operator $\oplus$). Instead, the β_v rule is closed under general contexts. Its restriction to closure under weak context is denoted $\underset{\mathsf{w}}{\to}_\beta$. The relation $\to$ is the union $\to_\beta \cup \to_\oplus$, while weak[4] reduction $\underset{\mathsf{w}}{\to}$ is the union of the weak reductions $\underset{\mathsf{w}}{\to}_{\beta_v} \cup \to_\oplus$. A $\to$-step which is not weak is noted $\underset{\neg\mathsf{w}}{\to}$.
2. The lifting of a relation $\to_r \subseteq \Lambda_\oplus \times \mathcal{M}(\Lambda_\oplus)$ to a reduction on multi-distribution is defined in Fig. 2. In particular, $\to, \to_{\beta_v}, \to_\oplus, \underset{\mathsf{w}}{\to}, \underset{\neg\mathsf{w}}{\to}$ lift to $\Rightarrow, \Rightarrow_{\beta_v}, \Rightarrow_\oplus, \underset{\mathsf{w}}{\Rightarrow}, \underset{\neg\mathsf{w}}{\Rightarrow}$.

The restriction of $\to_\oplus$ to weak contexts is necessary to have confluence, see [19] for a discussion. The fact that reduction $\to_{\beta_v}$ is unrestricted guarantees that the new calculus is a *conservative extension* of CbV λ-calculus.

Factorization, Modularly. Faggian and Ronchi della Rocca prove – by defining suitable notions of parallel reduction and internal parallel reduction with respect to $\Rightarrow_{\beta_v} \cup \Rightarrow_\oplus$ – that $\Lambda_\oplus^{\mathsf{cbv}}$ satisfies $\mathtt{Fact}(\underset{\mathsf{w}}{\Rightarrow}, \underset{\neg\mathsf{w}}{\Rightarrow})$, that is $\mathtt{m} \Rightarrow^* \mathtt{n}$ implies $\mathtt{m} \underset{\mathsf{w}}{\Rightarrow}^* \cdot \underset{\neg\mathsf{w}}{\Rightarrow}^* \mathtt{n}$.

This result – there called *finitary surface standardization* – is central in [19] because it is the base of the asymptotic constructions which are the core of that paper.

We now give a novel, strikingly short proof of the same result, by using Theorem 3.4. It turns out that we only need to verify the following swap, which is immediate to check.

▶ **Lemma B.1.** *$M \underset{\neg\mathsf{w}}{\to}_{\beta_v} \cdot \to_\oplus \mathtt{n}$ implies $M \to_\oplus \cdot \Rightarrow_{\beta_v} \mathtt{n}$.*

▶ **Theorem B.2** (Factorization of $\Rightarrow$)**.** *Let $\underset{\mathsf{w}}{\Rightarrow} := (\underset{\mathsf{w}}{\Rightarrow}_{\beta_v} \cup \Rightarrow_\oplus)$ and $\underset{\neg\mathsf{w}}{\Rightarrow} := (\underset{\neg\mathsf{w}}{\Rightarrow}_{\beta_v})$. Then $(\mathcal{M}(\Lambda_\oplus), \{\Rightarrow_{\beta_v}, \Rightarrow_\oplus\})$ satisfies w-factorization $\mathtt{Fact}(\underset{\mathsf{w}}{\Rightarrow}, \underset{\neg\mathsf{w}}{\Rightarrow})$.*

Proof. We verify that the conditions of Theorem 3.4 hold. Note that $\oplus$ has no internal steps, therefore, it suffices to verify only two conditions:

1. weak factorization of $\Rightarrow_{\beta_v}$: $\mathtt{Fact}(\underset{\mathsf{w}}{\Rightarrow}_{\beta_\mathsf{v}}, \underset{\neg\mathsf{w}}{\Rightarrow}_{\beta_\mathsf{v}})$.

[4] In [19], a *weak* reduction (resp. weak context) is called *surface*, and hence noted $\underset{\mathsf{s}}{\to}$.

2. $\underset{\neg\mathsf{w}}{\Rrightarrow}_{\beta_v}$ linearly swaps with $\Rightarrow_{\oplus}$: $\underset{\neg\mathsf{w}}{\Rrightarrow}_{\beta_v} \cdot \Rightarrow_{\oplus} \subseteq \Rightarrow_{\oplus} \cdot \Rightarrow_{\beta_v}^{*}$.

The other two conditions of Theorem 3.4 hold vacuously, because $\underset{\neg\mathsf{w}}{\Rrightarrow}_{\oplus} = \emptyset$ (and $\underset{\mathsf{w}}{\Rrightarrow}_{\oplus} = \Rightarrow_{\oplus}$).

1. $\mathtt{Fact}(\underset{\mathsf{w}}{\Rrightarrow}_{\beta_v}, \underset{\neg\mathsf{w}}{\Rrightarrow}_{\beta_v})$ follows from weak factorization of the CbV λ-calculus $\mathtt{Fact}(\underset{\mathsf{w}}{\rightarrow}_{\beta_v}, \underset{\neg\mathsf{w}}{\rightarrow}_{\beta_v})$ (see Sect. 6) because clearly ($[1M] \Rightarrow_{\beta_v} [1N]$ if and only if $M \rightarrow_{\beta_v} N$), ($[1M] \underset{\mathsf{w}}{\Rrightarrow}_{\beta_v} [1N]$ if and only if $M \underset{\mathsf{w}}{\rightarrow}_{\beta_v} N$), and similarly ($[1M] \underset{\neg\mathsf{w}}{\Rrightarrow}_{\beta_v} [1N]$ if and only if $M \underset{\neg\mathsf{w}}{\rightarrow}_{\beta_v} N$).
2. Lemma B.1 implies $\mathtt{m} \underset{\neg\mathsf{w}}{\Rrightarrow}_{\beta_v} \cdot \Rightarrow_{\oplus} \mathtt{n} \subseteq \mathtt{m} \Rightarrow_{\oplus} \cdot \Rightarrow_{\beta_v} \mathtt{n}$, by the definition of lifting. ◀

The Best a Monitor Can Do

Luca Aceto
Reykjavik University, Iceland
Gran Sasso Science Institute, L'Aquila, Italy
luca@ru.is

Antonis Achilleos
Reykjavik University, Iceland
antonios@ru.is

Adrian Francalanza
University of Malta, Malta
afra1@um.edu.mt

Anna Ingólfsdóttir
Reykjavik University, Iceland
annai@ru.is

Karoliina Lehtinen
University of Liverpool, UK
k.lehtinen@liverpool.ac.uk

Abstract

Existing notions of monitorability for branching-time properties are fairly restrictive. This, in turn, impacts the ability to incorporate prior knowledge about the system under scrutiny – which corresponds to a branching-time property – into the runtime analysis. We propose a definition of optimal monitors that verify the best monitorable under- or over-approximation of a specification, regardless of its monitorability status. Optimal monitors can be obtained for arbitrary branching-time properties by synthesising a sound and complete monitor for their *strongest monitorable consequence*. We show that the strongest monitorable consequence of specifications expressed in Hennessy-Milner logic with recursion is itself expressible in this logic, and present a procedure to find it. Our procedure enables prior knowledge to be optimally incorporated into runtime monitors.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Software and its engineering → Formal software verification

Keywords and phrases monitorability, branching-time logics, runtime verification

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.7

Funding Research supported by the Icelandic Research Fund projects "Theoretical Foundations for Monitorability" (No:163406-051) and "Epistemic Logic for Distributed Runtime Monitoring" (No:184940-051), the MIUR project PRIN 2017FTXR7S IT MATTERS, project BehAPI, funded by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No:778233, and project FouCo, funded by EU H2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No:892704.

1 Introduction

Branching-time properties, as described by logics such as CTL, CTL* and the modal μ-calculus, are normally verified using well-established pre-deployment techniques like model checking [18, 10]. However, there are cases where the system model is either unavailable (*e.g.*, due to third-party intellectual property restrictions), or not fully understood (*e.g.*, when parts of the system logic is governed by machine-learning tools). In these cases, monitors

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 7; pp. 7:1–7:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

can be used effectively to observe the execution of a system (rather than its state space) for verification purposes, as demonstrated in [26, 1, 4]; this technique is broadly referred to as runtime verification (RV) [25, 11].

RV is a best-effort strategy since it is limited to the incremental analysis of a single execution. The study of monitorability [4, 5] asks what correctness guarantees RV can provide and what properties can be monitored adequately with these guarantees. A wide body of work [46, 29, 50, 21, 26, 1, 28, 2, 4] primarily considers *safety* properties [7] ("something bad never happens") as those worth monitoring for, as they correspond to properties for which violations can always be identified from some finite prefix of an execution. However, limiting monitoring to this class of properties severely restricts the utility of RV. The restriction is particularly acute for branching-time properties [26, 1], which explains, in part, why RV tools generally restrict themselves to linear-time properties. But there are cases where formal specifications (a scarce resource in verification) have already been expressed in a branching-time logic and perhaps formally verified for some subcomponents. In such cases, a systematic method to incorporate such prior knowledge about the system into the runtime analysis would be beneficial.

A number of alternatives can be used to mitigate the shortcomings of RV for branching-time properties. One method would be to increase the observational powers of the monitor or employ multiple runs of the same system [2]. Alternatively, one can weaken the *monitor guarantees* expected during RV. The latter approach is the one explored in this paper. We propose the use of *optimal* monitors, which flag *all* violations that can be determined from execution prefixes contradicting the property (and ignore the violations that cannot). Although such monitors may fail to identify all violations, they represent the *best* monitors can do, and do not impose any restrictions on the considered class of properties. We show how these optimal monitors can be obtained systematically by computing the *strongest monitorable consequence* of the property to be dynamically verified.

▶ **Example 1.** A system with two (enumerated) components produces the events *open*, o_i, *write*, w_i, and *close*, c_i, for $i \in \{1, 2\}$. A specification for the first component states that:

"In all executions, w_1 (write) occurs, but only after an open, o_1." (Spec 1)

According to the existing notion of branching-time monitorability [26], a specification is monitorable if there is monitor that correctly identifies all violating processes from some prefix they produce. In other words, all violating processes must produce an execution prefix such that any process that produces this prefix also violates the property. This is one way of generalising the notion of safety property to the branching-time setting. (Spec 1) is *not* monitorable because there is *no* monitor which correctly identifies *all* violations of this specification. In particular, a first component that *never* reaches w_1 violates this property, but this cannot be determined from any finite prefix of its executions. However, there are other violations of (Spec 1) than can be detected. For instance, monitors can detect executions where w_1 occurs before o_1. Consider the (weaker) specification

"In all executions, w_1 (write) does not occur before an open, o_1." (Spec 2)

Since there is a monitor that detects *all* violations for (Spec 2), it is monitorable. More importantly, this monitor also turns out to be optimal with respect to (Spec 1) since these violations are the *only* ones that can be detected in (Spec 1). Conveniently, [26] also describes a procedure to synthesise the complete monitor from the logical formula describing (Spec 2) which could, in turn, also be used as the optimal monitor for (Spec 1). ⌟

▶ **Example 2.** The previous example illustrates the difficulty of monitoring for unbounded or infinite behaviour ("In all executions, w_1 occurs ...") which applies equally to linear and branching-time properties. Branching-time properties present additional challenges relating to the branching structure of computation. Consider the following specification:

"After o_2 (opening the second component), (closing it) c_2 is reachable
but always via w_2 (by writing to the second component beforehand)." (Spec 3)

This is intrinsically a branching time property as it concerns the state space of the system. In particular, no single execution can provide information about whether c_2 is reachable from all states entered via o_2. This property is therefore classified as unmonitorable. It turns out that its strongest monitorable consequence is the following property:

"After o_2, c_2 is only reachable via w_2." (Spec 4)

A sound and complete monitor for this specification flags a violation when it witnesses an execution in which o_2 is followed by c_2 without first seeing w_2. Such a monitor is also the optimal monitor for (Spec 3). Computing the strongest monitorable consequence of a property allows us to extract the part of the property amenable to runtime analysis. ⌟

Our proposed methodology allows us to address another common weakness found in existing RV approaches. Specifically, these approaches often treat the system under scrutiny as a *black box*, without leveraging any prior *partial* knowledge about the system.

▶ **Example 3.** Recall the system in Ex. 1 and consider the property:

"In all executions, c_1 (close) is never immediately followed by a write, w_1." (Spec 5)

(Spec 5) is monitorable according to [26]: a monitor can flag a violation whenever it observes $\mathsf{c}_1\mathsf{w}_1$ during an execution. On the other hand, if the monitor observes the sequence of events $\mathsf{c}_1\mathsf{w}_2\mathsf{w}_1$ in an execution, it cannot determine whether the system violates (Spec 5) or not.

But, suppose we have prior knowledge that the executions of the first and second components are completely independent. Then events such as w_2 and w_1 – coming from independent concurrent components – can be interleaved arbitrarily. A monitor that observes $\mathsf{c}_1\mathsf{w}_2\mathsf{w}_1$ can then infer that the system can also produce the sequence of events $\mathsf{c}_1\mathsf{w}_1\mathsf{w}_2$, meaning that observing $\mathsf{c}_1\mathsf{w}_2\mathsf{w}_1$, or more generally $\mathsf{c}_1\mathsf{w}_2^*\mathsf{w}_1$, provides enough evidence to flag that the system violates (Spec 5). In other words, the prior knowledge allows the monitor to infer violations from executions which by themselves wouldn't suffice to reach a verdict. ⌟

When synthesising monitors for a property P, such as (Spec 5), we would like to systematically incorporate any prior knowledge on the system, such as the independence of components or state-reachability, that can be expressed as a branching-time property K. To do this, we build a monitor based on the conjunction $K \wedge P$ rather than just P. Then, if an execution of a system known to satisfy K is inconsistent with $K \wedge P$, we can deduce that the system violates P. However, as $K \wedge P$ can be an arbitrary branching-time property, it might not itself be monitorable, even if P is monitorable, and the known monitor synthesis procedures might not apply. Again, we can adopt the procedure discussed earlier to obtain an *optimal* monitor for $K \wedge P$ instead. Note that while P might be designed to be a monitorable property, or even a linear-time property, K typically cannot be restricted in this way. In particular, properties such as those in Ex. 3 describing the possible interleavings of concurrent components, and those in Ex. 2 describing the system state-space, are inherently unmonitorable branching-time properties. Yet, so far, approaches to incorporate prior knowledge into runtime monitoring, referred to as grey-box monitoring or monitoring with assumptions, has restricted itself to knowledge representable as a linear-time property [30, 49, 17, 39].

Our contribution is twofold. First, we propose a general procedure to obtain optimal monitors for arbitrary branching-time properties (Sec. 3): following the intuition of Ex. 1 and 2, we find the *strongest monitorable consequence*, *e.g.,* (Spec 2), of an arbitrary branching-time property, *e.g.,* (Spec 1), which allows us to use existing synthesis procedures (*e.g.,* those in [9, 8]) to produce the sound and complete monitor from this monitorable consequence. We show that the resulting monitor is *optimal* for the original specification. This approach allows arbitrary branching-time specifications, for instance those originally designed for model checking, or those combining a monitorable property with prior knowledge, to be verified at runtime. We show that this is indeed the best a monitor can do with prior knowledge. Note that although we use an existing definition of branching-time monitorability to define the strongest monitorable consequence, our optimality result proves that using a different definition cannot improve the procedure. Our result can be seen as the generalisation of the notion of *bad prefixes* [35], i.e. prefixes that monitors can use to reach a negative verdict, to the branching-time setting. Although the set of bad prefixes appears frequently in various works in RV, its generalisation to the branching-time setting and the proposed disciplined methodology for obtaining optimal monitors from non-monitorable properties via the strongest monitorable consequence is, to the best of our knowledge, new.

Our second contribution is technical: we show in Sec. 5 how to compute the strongest monitorable consequence of an arbitrary property expressed in the Hennessy–Milner logic with Recursion, a variant of the modal μ-calculus. This is a popular verification logic that captures all regular tree languages, embeds other popular modal and temporal logics, such as LTL, CTL and CTL*, and corresponds to the bisimulation invariant fragment of monadic second order logic [31]. The size of the strongest monitorable consequence that we compute is bounded by a double exponential in the size of the original formula. This matches the bound on the size of a deterministic automaton that recognises the bad prefixes of an LTL formula [35]. In contrast, the transformation from an LTL formula to its strongest monitorable consequence, also expressed in LTL, is non-elementary (see Sec. 5.5).

We discuss related work, and in particular how this work compares to monitoring in the linear-time setting, in Sec. 6. Omitted proofs can be found in the appendix.

2 Preliminaries

Actions, Processes, Properties and Traces. Fix a finite set $\textsc{Act}$ of *actions* where $a, b \in \textsc{Act}$, a set of *process states* $p, q, r, \ldots \in \textsc{Prc}$ (sometimes called processes), and a transition relation, $\longrightarrow \subseteq (\textsc{Prc} \times \textsc{Act} \times \textsc{Prc})$. The triple $\langle \textsc{Prc}, \textsc{Act}, \longrightarrow \rangle$ forms a Labelled Transition System (LTS) [33] where the suggestive notation $p \xrightarrow{a} q$ denotes $(p, a, q) \in \longrightarrow$. For simplicity, we assume that all the processes that we refer to in this paper can be found in the same fixed infinite LTS, such as the one obtained from the set of CCS processes [43]. *Specifications*, or *properties*, are subsets of $\textsc{Prc}$, ranged over by P, Q, R. A property P is a *consequence* of property Q whenever $Q \subseteq P$. Actions may be sequenced to form *finite or infinite traces* $t, u \in (\textsc{Act}^* \cup \textsc{Act}^\omega)$; the trace prefix-ordering $t \leq u$ denotes that t is a prefix of u. We say that a process p *produces* a trace t, or that t is a trace of p, if there is a sequence of transitions $p \xrightarrow{a} q \xrightarrow{b} \cdots$, such that $t = ab \cdots$; the trace t is also referred to as an *execution* of p. Note that if t is a trace of p, then so are all of its prefixes.

Runtime Monitoring, Verification and Monitorability. Runtime monitors are computational entities that reach a verdict after observing a finite prefix of an execution. A verdict, once reached, is irrevocable [5]. We only consider single-verdict monitors, namely *rejection monitors*, which flag violations of a property, and *acceptance monitors*, which validate a property. Although mixed-verdict monitors can be used in a linear-time setting [4], only single-verdict monitors make sense in a branching-time setting[1]. Rejection and acceptance monitors are dual to one another in this setting. Our technical development thus focuses on rejection monitors, and obtains results for acceptance monitors by duality.

A monitor, denoted by $m, n, \ldots$, may be abstractly described as a (possibly infinite) set of finite traces, $m \subseteq \textsc{Act}^*$, that satisfies the following condition: if $t \in m$, then for any $u \in \textsc{Act}^*$ where $t \leq u$ it holds that $u \in m$. Intuitively, the traces in m are those that witness a violation of a property. The closure condition describes the irrevocability of verdicts. The collection of upward-closed subsets of $\textsc{Act}^*$, denoted by $\textsc{Mon}$, is therefore the set of all possible monitors. Often we restrict our discussion to a *subset* of $\textsc{Mon}$, $M \subseteq \textsc{Mon}$.

▶ **Definition 4.** *Monitor m rejects process p, $\boldsymbol{rej}(m, p)$, if p produces a trace t in m.* ⌟

Earlier work [23, 26, 24, 4] provides an operational interpretation of Def. 4 via an instrumentation of the monitor m executing with process p. Soundness and completeness relate monitors to the specifications they are expected to monitor [26, 4, 5]. *Soundness* requires that a monitor give only correct verdicts, while *completeness* demands that a monitor reject whenever the specification is violated.

▶ **Definition 5** (Soundness and Completeness). *A monitor $m \in \textsc{Mon}$ is:*

1. sound *for specification P if for all $p \in \textsc{Prc}$, $\boldsymbol{rej}(m, p)$ implies $p \notin P$;*
2. complete *for specification P if for all $p \in \textsc{Prc}$, $p \notin P$ implies $\boldsymbol{rej}(m, p)$.* ⌟

▶ **Definition 6** (Monitorability). *A specification P is* monitorable *in a monitor set M if there exists some $m \in M$ that is sound and complete for P.* ⌟

The notion of monitorability given in Def. 6 comes from [26]; although it is one of many possible definitions [5], it is the only one that has been extensively studied in the branching-time setting [26, 3, 1, 2, 4]. It also turns out to be the right one to use in the quest for optimal monitors, as argued in Sec. 3. One important consequence of Def. 6 is that there are some properties that are *not* monitorable.

▶ **Example 7.** The monitor from Ex. 3 that rejects all traces containing the consecutive events $\mathsf{c}_1\mathsf{w}_1$ is sound and complete for Spec 5, whereas Spec 1 in Ex. 1 is not monitorable. In the sequel, we simplify our example and assume that there is only one component in the system generating events $\mathsf{o}, \mathsf{w}, \mathsf{c}$. Another property that is not monitorable is the following:

"cw never occurs and on all infinite executions, w occurs." (Spec 6)

Indeed, a process whose only maximal trace is o^ω is not in this property but there is *no* monitor that is sound for Spec 6 and rejects it. ⌟

In practice, we often have (prior) knowledge about the type of process the monitor will be analysing at runtime, and the definition of monitorability should take such information into account, *i.e.*, grey-box monitoring. For our setting, we can express this prior knowledge as a set of processes, denoted as $R \subseteq \textsc{Prc}$, *i.e.*, the processes satisfying that prior knowledge.

▶ **Definition 8** (Soundness/Completeness with Knowledge). *The monitor $m \in \textsc{Mon}$ is:*

- Sound *for specification P with knowledge R if for all $p{\in}R$, $\boldsymbol{rej}(m, p)$ implies $p{\notin}P$.*
- *Complete for specification P with knowledge R if for all $p{\in}R$, $p{\notin}P$ implies $\boldsymbol{rej}(m, p)$.* ⌟

[1] Multi-verdict monitors are necessarily unsound in the branching-time setting [26].

▶ **Definition 9** (Monitorability with Knowledge). *A specification P is monitorable in a monitor set M, with prior knowledge R, if there exists a monitor $m \in M$ that is both sound and complete for P with knowledge R.* ⌟

3 The Strongest Monitorable Consequence

Since not all specifications have a sound and complete monitor, we are interested in computing an *optimal* monitor: a monitor which is sound for the specification, and rejects all violations that can be flagged. In this section we argue that to find the optimal monitor of a specification, we first need to compute its *strongest monitorable consequence.*

Although we focus on rejection monitors, optimal acceptance monitors are dual. An *optimal* monitor for a property P is a sound monitor for P that rejects each trace rejected by some sound monitor for that property.

▶ **Definition 10** (Optimality). *For a fixed monitor set $M \subseteq \textsc{Mon}$, monitor $m \in M$ is* optimal *in M for the property P whenever:*

- *it is sound for P and*
- *for all $n \in M$, if n is sound for P then $n \subseteq m$.* ⌟

Since the definition of a monitor as a set of finite traces does not guarantee computability, it is useful to parameterise this definition with the set of monitors M that determines the computational power of the monitors under scrutiny.

We now aim to characterise optimal monitors in terms of the properties they monitor for. First, for every monitor m, we can easily define a property for which it is both sound and complete:

$$P_m = \{\, p \mid p \text{ does not produce any trace } t \in m \,\}.$$

It is not hard to see that such a property P_m is unique for every monitor m.

▶ **Lemma 11.** *Monitor m is sound and complete for P if and only if $P = P_m$.* ◀

▶ **Proposition 12.** *For all $m, n \in \textsc{Mon}$, $m \subseteq n$ iff $P_n \subseteq P_m$.*

Proof. For the *if case*, assume $P_n \subseteq P_m$ and pick a $t \in m$ and the process p that produces only t. Then, $p \notin P_m$, which implies $p \notin P_n$ from $P_n \subseteq P_m$. By definition of P_n, this implies $t \in n$. We conclude that $m \subseteq n$. For the *only-if case*, assume $m \subseteq n$ and pick a $p \notin P_m$ that produces some $t \in m$. By inclusion $t \in n$ and therefore $p \notin P_n$. ◀

We can now characterise optimal monitors, Def. 10, in terms a notion of a *strongest monitorable consequence.*

▶ **Definition 13** (Strongest Monitorable Consequence). *Let $M \subseteq \textsc{Mon}$. The strongest monitorable consequence of a specification P with respect to M is a property Q that is monitorable in M such that:*

- *it is a consequence of P, i.e., $P \subseteq Q$, and*
- *for any R monitorable in M, $P \subseteq R$ implies $Q \subseteq R$.* ⌟

Note that the existence of a strongest monitorable consequence and of an optimal monitor, depends on M. We establish the correspondence between strongest monitorable consequences and optimal monitors (Thm. 16) using the following two lemmas.

▶ **Lemma 14.** *A* sound *monitor for a consequence of P is sound for P.*

Proof. Pick a consequence Q of P, *i.e.*, $P \subseteq Q$, and a sound monitor m for Q. If $\mathbf{rej}(m,p)$ for some p, then $p \notin Q$ by Def. 5. By $P \subseteq Q$ we obtain $p \notin P$, so m is sound for P. ◀

▶ **Lemma 15.** *If m is complete for P and sound for Q then $Q \subseteq P$.*

Proof. Pick a process $p \notin P$; for P to be a consequence of Q, *i.e.*, $Q \subseteq P$, we need to show that $p \notin Q$. By Def. 5.2 we know $\mathbf{rej}(m,p)$ and by Def. 5.1 we obtain $p \notin Q$. ◀

▶ **Theorem 16.** *A monitor $m \in M$ that is sound for P is optimal for P in M iff it is sound and complete for the strongest monitorable consequence of P with respect to M.*

Proof. For the *if case*, assume that m is sound and complete for Q, the strongest monitorable consequence of P with respect to M. We must show that m is optimal for P in M. Pick any other monitor $n \in M$ that is also sound for P. From Lem. 11, P_n is monitorable in M, and by Lem. 15 we know $P \subseteq P_n$. Since Q is the strongest monitorable consequence of P, we also know $Q \subseteq P_n$, and by Prop. 12 we obtain $n \subseteq m_Q$ as required.

For the *only-if case*, let m be an optimal monitor for P. By Lem. 11 and the soundness of m for P, it follows that P_m is a consequence of P, *i.e.*, $P \subseteq P_m$. Next, we show that P_m is the strongest monitorable consequence of P, from which the claim follows because m is sound and complete for P_m. Let Q be a monitorable consequence of P and let m_Q be a monitor for it. Since m is optimal (Def. 10), we know that $m_Q \subseteq m$. Thus by Prop. 12 we obtain $P_m \subseteq Q$. This implies that P_m is the strongest monitorable consequence of P. ◀

To find the optimal monitor of an arbitrary property, it therefore suffices to compute the sound and complete monitor of its strongest monitorable consequence. We can also extend this result for the cases with prior knowledge about the process to be monitored, Thm. 19.

▶ **Definition 17** (Optimality with Knowledge). *For a fixed monitor set $M \subseteq$ MON, monitor $m \in M$ is* optimal *in M for property P with knowledge R whenever:*

- *it is sound for P with knowledge R and*
- *for all $n \in M$, if n is sound for P with knowledge R then $n \subseteq m$.* ⌟

Soundness and completeness with prior knowledge can be characterised with respect to soundness and completeness in the setting with no prior knowledge, PRC.

▶ **Proposition 18.** *Monitor m is sound with knowledge R for P iff it is sound for $P \cap R$.*

▶ **Theorem 19.** *For a fixed monitor set $M \subseteq$ MON, a monitor $m \in M$ is optimal in M for the property P with knowledge R iff m is optimal in M for property $P \cap R$.*

Proof. For the *only-if* case, assume that m is optimal for P with knowledge R. From Def. 17, we know that m is sound for P with knowledge R, and therefore, by Prop. 18, m is sound for $P \cap R$. From Def. 17, we also know that if some n is sound for P with R, then $n \subseteq m$; again, by Prop. 18, if n is sound for $P \cap R$, then $n \subseteq m$. Therefore, m is also optimal for $P \cap R$. The *if* case is symmetric. ◀

4 Monitorability in recHML

Following Thms. 16 and 19, we investigate how to compute the strongest monitorable consequence for properties expressible in the Hennessy–Milner logic with recursion, RECHML [37], as a means to obtain optimal monitors for such properties. RECHML is a specification logic describing regular properties of processes, and can be seen as a reformulation of the well-studied modal μ-calculus [13, 14]. Since there are standard translations from CTL and

Syntax

$$\begin{array}{rllll}
\varphi,\psi \in \textsc{rec}\mathrm{HML} ::= \mathsf{tt} & \text{(truth)} & \mid\ \mathsf{ff} & \text{(falsehood)} \\
\mid\ \varphi \vee \psi & \text{(disjunction)} & \mid\ \varphi \wedge \psi & \text{(conjunction)} \\
\mid\ \langle a \rangle \varphi & \text{(existential modality)} & \mid\ [a]\varphi & \text{(universal modality)} \\
\mid\ \mathsf{min}\, X.\varphi & \text{(least fixpoint)} & \mid\ \mathsf{max}\, X.\varphi & \text{(greatest fixpoint)} \\
\mid\ X & \text{(recursion variable)} & &
\end{array}$$

Branching-Time Semantics

$$\begin{array}{rclrcl}
[\![\mathsf{tt},\rho]\!] & \stackrel{\text{def}}{=} & \textsc{Prc} & [\![\mathsf{ff},\rho]\!] & \stackrel{\text{def}}{=} & \emptyset \\
[\![\varphi_1 \vee \varphi_2,\rho]\!] & \stackrel{\text{def}}{=} & [\![\varphi_1,\rho]\!] \cup [\![\varphi_2,\rho]\!] & [\![\varphi_1 \wedge \varphi_2,\rho]\!] & \stackrel{\text{def}}{=} & [\![\varphi_1,\rho]\!] \cap [\![\varphi_2,\rho]\!] \\
[\![\langle a \rangle \varphi,\rho]\!] & \stackrel{\text{def}}{=} & \left\{ p \mid \exists q \cdot p \xrightarrow{a} q \text{ and } q \in [\![\varphi,\rho]\!] \right\} & [\![X,\rho]\!] & \stackrel{\text{def}}{=} & \rho(X) \\
[\![[a]\varphi,\rho]\!] & \stackrel{\text{def}}{=} & \left\{ p \mid \forall q \cdot p \xrightarrow{a} q \text{ implies } q \in [\![\varphi,\rho]\!] \right\} & & & \\
[\![\mathsf{min} X.\varphi,\rho]\!] & \stackrel{\text{def}}{=} & \bigcap \{ P \mid [\![\varphi,\rho[X \mapsto P]]\!] \subseteq P \} & [\![\mathsf{max} X.\varphi,\rho]\!] & \stackrel{\text{def}}{=} & \bigcup \{ P \mid P \subseteq [\![\varphi,\rho[X \mapsto P]]\!] \}
\end{array}$$

Figure 1 recHML Syntax and Branching-Time Semantics.

CTL* [34] into recHML, our investigation extends to these logics as well. The appeal of recHML comes from its generality, the pre-existence of procedures to compute sound and complete monitors for its monitorable fragment and its good closure properties. Indeed, we show that the strongest monitorable consequence of recHML formulae is itself expressible in recHML. It is unclear whether this is also the case for other branching-time logics, although in the linear time setting, this question is settled positively for LTL in [40].

recHML formulae are generated from the syntax given in Fig. 1, according to the following order of precedence: the existential and universal modal operators ($\langle a \rangle \varphi$ and $[a]\varphi$), conjunctions, disjunctions, and fixpoint operators ($\mathsf{min}\, X.\varphi$ and $\mathsf{max}\, X.\varphi$). The negation of a formula φ can be constructed with the duality rules in the usual way, and we use $\neg\varphi$ as a shorthand for it. In a formula $\mathsf{min}\, X.\varphi$ or $\mathsf{max}\, X.\varphi$, the fixpoint operator binds all free occurrences of X in φ. The subformula φ is then said to be the binding formula of X. We assume that for each variable X, there is exactly one formula $\mathsf{min}\, X.\varphi$ or $\mathsf{max}\, X.\varphi$ that binds X, denoted φ_X. Furthermore, without loss of generality, all formulas are assumed to be guarded [36]: every occurrence of a fixpoint variable within its binding is within the scope of a modal operator. We extend the notion of subformula and say that φ_X is the immediate subformula of X. We write $\mathit{sf}(\varphi)$ for the set of subformulas of φ. We take the size of a formula to be the number of its distinct subformulae, up to α-conversion.

A formula φ from recHML is evaluated on a state of an LTS. In addition to true, false and boolean connectives – which have their usual semantics – recHML has modal and fixpoints operators. The existential modality $\langle a \rangle \varphi$ holds at a state if there is an a-successor in which φ holds, whereas the universal modality $[a]\varphi$ holds if φ holds in all the a-successors of that state. The least fixpoint $\mathsf{min}\, X.\varphi$ and its dual $\mathsf{max}\, X.\varphi$ add recursion to the logic, allowing for the description of temporal properties such as reachability and invariance. Formally, the semantics is defined with respect to an interpretation ρ of the free variables of the formula. We write $[\![\varphi,\rho]\!]$ for the set of process states in an LTS which satisfy φ according to ρ. This set is defined by induction on the structure of the formula φ, following the semantics given in Fig. 1. Two formulas are equivalent if they agree on all processes. We often consider closed formulas – namely those without free variables. In these cases, we can ignore the environment from the semantics and simply write $[\![\varphi]\!]$ instead of $[\![\varphi,\rho]\!]$.

▶ Remark 20. A system state p trivially satisfies a specification $[a]\varphi$ if it *cannot* transition with action a. Consequently the basic formula $[a]\mathsf{ff}$ describes states that cannot perform a-transitions; the dual basic formula $\langle a\rangle\mathsf{tt}$ denotes states that can perform a-transitions.

▶ **Example 21.** Property Spec 6 from Ex. 7 for $\textsc{Act} = \{\mathsf{o},\mathsf{w},\mathsf{c}\}$ can be expressed as:

$$\varphi_1 = (\mathsf{max}\,X.([\mathsf{o}]X \wedge [\mathsf{c}]X \wedge [\mathsf{w}]X \wedge [\mathsf{c}][\mathsf{w}]\mathsf{ff}))\ \wedge\ \mathsf{min}\,Y.([\mathsf{o}]Y \wedge [\mathsf{c}]Y).$$

The first conjunct in φ_1 prohibits the occurrence of cw while the second conjunct requires w to eventually occur on infinite traces (the sub-formula $\langle\mathsf{w}\rangle\mathsf{tt}$ disjuncted with $[\mathsf{o}]Y \wedge [\mathsf{c}]Y$ can be left implicit since $\textsc{Act} = \{\mathsf{o},\mathsf{w},\mathsf{c}\}$). A variation of Spec 1 from Ex. 1 on one component (for $\textsc{Act} = \{\mathsf{o},\mathsf{w},\mathsf{c}\}$) is Spec 7, described below and formalised as the formula φ_2:

“On all infinite executions, w occurs, but w only occurs after o.” (Spec 7)

Whereas the outermost fixpoint formula in φ_2 below prohibits w from occurring before o, the innermost fixpoint formula requires w to occur eventually in any infinite execution.

$$\varphi_2 = \mathsf{min}\,X.([\mathsf{w}]\mathsf{ff} \wedge [\mathsf{c}]X \wedge [\mathsf{o}](\mathsf{min}\,Y.[\mathsf{c}]Y \wedge [\mathsf{o}]Y)).$$ ⌟

Monitorability for recHML was investigated in [26, 1], where monitors are specified as regular processes and monitorable properties have a syntactic characterisation:

▶ **Theorem 22.** *[26, Theorems 1 and 4] A formula of* recHML *is* (violation) monitorable *iff it is equivalent to a formula in the fragment* sHML *defined as follows:*

$$\varphi,\psi \in \textit{s}\mathit{HML} ::= \quad \mathsf{tt} \quad \mid \mathsf{ff} \quad \mid [a]\varphi \quad \mid \varphi\wedge\psi \quad \mid \mathsf{max}\,X.\varphi \quad \mid X$$ ◀

A synthesis function that generates a regular (sound and complete) monitor from a sHML formula is also presented; such monitors are also shown to be finite state [4].

▶ **Example 23.** Since φ_1 and φ_2 are not sHML formulas, we cannot use the synthesis function from [26] to obtain runtime monitors for them. In fact, neither formula is monitorable according to [26]. Although Spec 5 from Ex. 3, with $\textsc{Act}_i{=}\{\mathsf{o}_i,\mathsf{w}_i,\mathsf{c}_i\}$ and $\textsc{Act}{=}\textsc{Act}_1{\cup}\textsc{Act}_2$, can be expressed as the sHML formula φ_3, the knowledge (component independence) can be only expressed using formulas like φ_4, which are neither in sHML nor monitorable [26].

$$\varphi_3 = \mathsf{max}\,X.\Big([\mathsf{c}_1][\mathsf{w}_1]\mathsf{ff} \wedge \bigwedge_{a\in\textsc{Act}} [a]X\Big)$$

$$\varphi_4 = \mathsf{max}\,X.\ \bigwedge_{\substack{a\in\textsc{Act}_1\\ b\in\textsc{Act}_2}} ([a][b]\mathsf{ff} \vee \langle b\rangle\langle a\rangle\mathsf{tt}) \wedge \bigwedge_{\substack{a\in\textsc{Act}_2\\ b\in\textsc{Act}_1}} ([a][b]\mathsf{ff} \vee \langle b\rangle\langle a\rangle\mathsf{tt}) \wedge \bigwedge_{a\in\textsc{Act}} [a]X$$

The sub-formula $([a][b]\mathsf{ff} \vee \langle b\rangle\langle a\rangle\mathsf{tt})$ in φ_4 encodes the implication $(\langle a\rangle\langle b\rangle\mathsf{tt} \Rightarrow \langle b\rangle\langle a\rangle\mathsf{tt})$. The strongest monitorable consequence of $\varphi_3 \wedge \varphi_4$ is expressed by φ_5:

$$\varphi_5 = \mathsf{max}\,X.[\mathsf{c}_1](\mathsf{max}\,Y.\bigwedge_{b\in\textsc{Act}_2} [b]Y \wedge [\mathsf{w}_1]\mathsf{ff}) \wedge \bigwedge_{a\in\textsc{Act}} [a]X$$

A sound and complete monitor for this property will reject a process based on executions containing $\mathsf{c}_1\textsc{Act}_2^*\mathsf{w}_1$, rather than just $\mathsf{c}_1\mathsf{w}_1$. ⌟

In cases such as Ex. 23, we can obtain the optimal monitor of an arbitrary recHML specification φ by: (i) computing the strongest monitorable consequence $\psi \in$ sHML of φ; (ii) synthesising a sound and complete monitor for ψ using the synthesis function from [26].

5 Computing Strongest Monitorable Consequences in recHML

In this section, we describe a method for computing the strongest monitorable consequence of a recHML formula. The full proofs for this section can be found in the appendix. Our constructions rely on a disjunctive representation of formulas, as given in Def. 24.

▶ **Definition 24** (Disjunctive Form [51]). *The set of disjunctive formulas of* recHML *is given by the following grammar:*

$$\varphi, \psi \in \textsc{dis}\mathrm{HML} ::= \quad \mathsf{tt} \quad \mid \mathsf{ff} \quad \mid \varphi \vee \psi \quad \mid \bigwedge_{a \in A} \Big(\big(\bigwedge_{\varphi \in \mathcal{B}_a} \langle a \rangle \varphi \big) \wedge [a] \bigvee_{\varphi \in \mathcal{B}_a} \varphi \Big)$$
$$\mid \mathsf{max}\, X.\varphi \quad \mid \mathsf{min}\, X.\varphi \quad \mid X,$$

where $A \subseteq$ Act and, for each action a in A, the set $\mathcal{B}_a \subseteq$ disHML is a finite set of formulas. ◀

In disjunctive formulas, conjunctions occur to express that for each $a{\in}A$, every a-successor satisfies a formula in some set $\mathcal{B}_a$ and every formula in $\mathcal{B}_a$ is satisfied by some a-successor.

▶ **Lemma 25** ([51]). *Every* recHML *formula is equivalent to a disjunctive one.* ◀

In [51], Walukiewicz provides a way to construct an equivalent disjunctive formula from a recHML one, based on a tableau method. He also shows that the satisfiability of disjunctive recHML formulas is decidable in linear time. Thus, we assume that, with the exception of ff, all subformulas of disjunctive formulas are satisfiable. This pre-processing accounts for one exponential in the complexity of our transformation.

We now establish a fundamental property of sHML formulas: if a process q does not satisfy $\theta \in$ sHML, then no process p that can produce all traces of q satisfies θ.

▶ **Definition 26.** *Process p* covers *process q when all traces of q are traces of p.* ◀

▶ **Lemma 27.** *If process p covers process q, then for closed $\theta{\in}$sHML, $q{\notin}[\![\theta]\!]$ implies $p{\notin}[\![\theta]\!]$.*

Proof. From [26] there is a sound and complete m for θ. By $q \notin [\![\theta]\!]$ and the completeness of m, there is a trace of q (and of p), rejected by m. By the soundness of m, $p \notin [\![\theta]\!]$. ◀

We present the construction of the strongest monitorable consequence of a formula Ψ in three stages. We first eliminate the existential modalities in a formula. Then we eliminate least fixpoints. Finally, we use a more involved tableau method to remove all disjunctions.

5.1 Eliminating Existential Modalities

▶ **Definition 28.** *The operator f_1 :* recHML → recHML *is defined such that $f_1(\langle a \rangle \varphi) = \mathsf{tt}$, while commuting with all other logical connectives.* ◀

That is, $f_1(\Psi)$ results from Ψ by replacing every occurrence of a subformula $\langle a \rangle \varphi$ by tt.

▶ **Lemma 29.** *For every $\Psi \in$ disHML, $f_1(\Psi)$ has the same* sHML *consequences as Ψ.*

(Proof outline). We show that every sHML formula is a consequence of Ψ iff it is a consequence of $f_1(\Psi)$. For the *if* direction, it suffices to prove $[\![\Psi]\!] \subseteq [\![f_1(\Psi)]\!]$ using the monotonicity of recHML operators resulting from the absence of negation.

$$\frac{\Gamma \cup \{\psi \vee \varphi\}}{\Gamma \cup \{\varphi, \psi\}}\,(\vee) \qquad \frac{\Gamma \cup \{\psi \wedge \varphi\}}{\Gamma \cup \{\varphi\} \qquad \Gamma \cup \{\psi\}}\,(\wedge) \qquad \frac{\Gamma \cup \{\mathsf{max}\, X.\varphi\}}{\Gamma \cup \{\varphi\}}\,(\mathsf{max}) \qquad \frac{\Gamma}{\{\psi \mid [a]\psi \in \Gamma\}}\,([a])$$

$$\frac{\Gamma \cup \{X\}}{\Gamma \cup \{\varphi_X\}}\,(X) \qquad \frac{\Gamma \cup \{\mathsf{tt}\}}{\{\mathsf{tt}\}}\,(\mathsf{tt}) \qquad \frac{\Gamma \cup \{\mathsf{ff}\}}{\Gamma}\,(\mathsf{ff}) \qquad \frac{\Gamma \cup \{[a]\psi, [b]\varphi\} \quad a \neq b}{\{\mathsf{tt}\}}\,([a,b])$$

Figure 2 Tableau rules where Γ is a set of formulas.

For the *only-if* direction, the intuition is as follows (see App. A). Let θ be a sHML formula such that $[\![\Psi]\!] \subseteq [\![\theta]\!]$. To show $[\![f_1(\Psi)]\!] \subseteq [\![\theta]\!]$, we proceed by contradiction: starting from a process $p \in [\![f_1(\Psi) \wedge \neg\theta]\!]$ we build a cover q of p such that $q \in [\![\theta]\!]$, which contradicts Lem. 27. To obtain this cover, we use the fact that f_1 turns the conjunctions $\bigwedge_{a \in A}((\bigwedge_{\psi \in \mathcal{B}_a} \langle a \rangle \psi) \wedge [a] \bigvee \mathcal{B}_a)$ of a disjunctive formula into conjunctions of the form $\bigwedge_{a \in A} [a] \bigvee \mathcal{B}_a$. The cover is obtained by finding the states r in which conjunctions of the latter form must be true for $f_1(\Psi)$ to be true in p, and adding an a-successor s_φ to r for each $\varphi \in \mathcal{B}_a$ and $a \in A$. This is possible, because all subformulae of disjunctive formulas are assumed to be satisfiable. The state r with these additional successors then satisfies $\bigwedge_{a \in A}((\bigwedge_{\psi \in \mathcal{B}_a} \langle a \rangle \psi) \wedge [a] \bigvee \mathcal{B}_a)$, which allows us to argue that $q \in [\![\Psi]\!] \subseteq [\![\theta]\!]$. ◀

▶ Remark 30. Disjunctive form is key here: Applying f_1 to formula φ_4 from Ex. 23, which is not disjunctive, yields $\left(\bigwedge_{a \in \textsc{Act}_1} \bigwedge_{b \in \textsc{Act}_2} ([a][b]\mathsf{ff} \vee \mathsf{tt})\right) \wedge \left(\bigwedge_{a \in \textsc{Act}_2} \bigwedge_{b \in \textsc{Act}_1} ([a][b]\mathsf{ff} \vee \mathsf{tt})\right)$ which can be simplified to tt and does not provide any useful information for monitoring.

5.2 Eliminating Least Fixpoints

▶ **Definition 31.** *The operator $f_2 : \textsc{rec}HML \to \textsc{rec}HML$ is defined such that $f_2(\mathsf{min}\, X.\varphi) = \mathsf{max}\, X.\varphi$, while commuting with all other logical connectives.* ◀

▶ **Lemma 32.** *For every closed formula $\Psi \in \textsc{rec}HML$ without existential modalities, $f_2(\Psi)$ has the same sHML consequences as Ψ.*

(Proof outline). One direction follows from $[\![\mathsf{min}\, X.\varphi]\!] \subseteq [\![\mathsf{max}\, X.\varphi]\!]$: since $\textsc{rec}$HML is negation-free, it behaves in a monotone way, and therefore $f_2(\Psi)$ is a consequence of Ψ.

The intuition for the other direction is as follows (see App. A). If a process p violates a consequence $\theta \in$ sHML of Ψ but satisfies $f_2(\Psi)$, then, due to the monitorability of θ, there is a finite trace t of p, where every process producing t must also violate θ. Thus, there is a *finite* process q that violates θ, but also satisfies $f_2(\Psi)$ due to the absence of existential modalities in $f_2(\Psi)$. Since $f_2(\Psi)$ and Ψ only differ with respect to their fixpoint operators, they agree on all finite processes: q satisfies Ψ and its consequence θ, a contradiction. ◀

▶ Remark 33. Lem. 32 does not hold for formulas *with* existential modalities. For instance, the formula $\mathsf{min}\, X.\langle a \rangle X$ is equivalent to, and thus implies, ff; yet $f_2(\mathsf{min}\, X.\langle a \rangle X) = \mathsf{max}\, X.\langle a \rangle X$, which is satisfiable by a system producing the infinite trace a^ω.

▶ **Example 34.** Formula φ_2 from Ex. 21 becomes $\mathsf{max}\, X.([\mathsf{w}]\mathsf{ff} \wedge [\mathsf{c}]X \wedge [\mathsf{o}](\mathsf{max}\, Y.[\mathsf{c}]Y \wedge [\mathsf{o}]Y))$ under $f_2(-)$, which simplifies to $\mathsf{max}\, X.([\mathsf{w}]\mathsf{ff} \wedge [\mathsf{c}]X)$ as $\mathsf{max}\, Y.[\mathsf{c}]Y \wedge [\mathsf{o}]Y$ simplifies to tt. Since $\textsc{Act}{=}\{\mathsf{o}, \mathsf{c}, \mathsf{w}\}$ this formula expresses the property that "w does not occur before o." ◀

5.3 Eliminating Disjunctions

The final and hardest step turns a formula without existential modalities and least fixpoints into its strongest sHML consequence. The intuition is that a violation of a specification of the form $[a]\psi \vee [a]\varphi$ can only be monitored if there is an a-successor in which violations for *both* ψ and φ can be detected. Hence, we turn $[a]\psi \vee [a]\varphi$ into $[a](\psi \vee \varphi)$. In contrast, no violation of $[a]\psi \vee [b]\varphi$ can be identified from a single branch, so we rewrite it to tt.

To transform fixpoint-free formulas, it suffices to recursively push disjunctions through the formula. The transformation in the presence of fixpoints is roughly dual to that for disjunctive form presented by Janin and Walukiewicz in [32] and, like theirs, uses a set of tableau rules, but this time to eliminate disjunctions rather than conjunctions. Our rules differ significantly from those in [32] in how they deal with modalities; in particular, our transformation does not preserve the semantics of formulae, but only sHML consequences.

▶ **Definition 35** (Tableau elimination of disjunctions). *Given a closed formula Ψ with neither* min *operators nor existential modalities, we build a tableau $\mathcal{T}(\Psi)$ consisting of a tree with back edges, where each node n is labelled with a set $L_\Psi(n)$ of subformulae of Ψ, such that:*

- *The root is labelled $\{\Psi\}$,*
- *For each node n and its children, there is a tableau rule (Fig. 2) such that n is labelled with the premise and its children are labelled with its conclusions,*
- *This tableau rule is the rule $[a]$ only if $L_\Psi(n)$ matches the premise of no other tableau-rule.*

The disjunction-free formula equivalent to Ψ is then retrieved from $\mathcal{T}(\Psi)$ by defining the labelling L' as follows and applying it to each node. For each leaf node n:

- *If it has a back-edge to an inner node m, it is labelled X_m;*
- *If it does not have a back-edge, it is labelled with* tt, *if it contains* tt, *and* ff, *otherwise.*

For each inner node n that is not the target of a back-edge:

- *If n has a child m via the rules $\vee, \mathsf{tt}, [a,b], X, \mathsf{max}$, then l has the label $L'(m)$;*
- *If n has children m, m' via rule $\wedge$, then l is label $L'(m) \wedge L'(m')$;*
- *If n has a child m via $[a]$, then l is label $[a]L'(m)$.*

In a second pass, if n is the target of back-edges, then its label is $\mathsf{max}\, X_n.l$, and otherwise it is l, where $l = L'(n)$ as defined above. Let $f_3(\Psi)$ be the L'-label of the root of $\mathcal{T}(\Psi)$.

▶ **Example 36.** Consider the bespoke formula $\mathsf{max}\, X.[a]([a]X \wedge [b]\mathsf{ff}) \vee [a]([a]\mathsf{ff} \wedge [b]X)$. The tableau for this formula labelled with subsets of subformulas using Def. 35 is given below.

$$\dfrac{\mathsf{max}\, X.[a]([a]X \wedge [b]\mathsf{ff}) \vee [a]([a]\mathsf{ff} \wedge [b]X)}{\dfrac{[a]([a]X \wedge [b]\mathsf{ff}) \vee [a]([a]\mathsf{ff} \wedge [b]X)}{\dfrac{[a]([a]X \wedge [b]\mathsf{ff}), [a]([a]\mathsf{ff} \wedge [b]X)}{\dfrac{[a]X \wedge [b]\mathsf{ff}, [a]\mathsf{ff} \wedge [b]X}{\dfrac{[a]X, [a]\mathsf{ff} \wedge [b]X}{\dfrac{[a]X, [a]\mathsf{ff}}{\dfrac{X, \mathsf{ff}}{X}\,(\mathsf{ff})}\,([a]) \quad \dfrac{[a]X, [b]X}{\mathsf{tt}}\,([a,b])}\,(\wedge) \qquad \dfrac{[b]\mathsf{ff}, [a]\mathsf{ff} \wedge [b]X}{\dfrac{[b]\mathsf{ff}, [a]\mathsf{ff}}{\mathsf{tt}}\,([a,b]) \quad \dfrac{[b]\mathsf{ff}, [b]X}{\dfrac{\mathsf{ff}, X}{X}\,(\mathsf{ff})}\,([b])}\,(\wedge)}\,(\wedge)}\,([a])}\,(\vee)}\,(\mathsf{max})$$

The corresponding tableau relabelled as L' yielding the strongest sHML consequence is:

$$\dfrac{\mathsf{max}\, X_1.[a]([a]X_1 \wedge \mathsf{tt} \wedge \mathsf{tt} \wedge [b]X_1)}{\dfrac{[a]([a]X_1 \wedge \mathsf{tt} \wedge \mathsf{tt} \wedge [b]X_1)}{\dfrac{[a]([a]X_1 \wedge \mathsf{tt} \wedge \mathsf{tt} \wedge [b]X_1)}{\dfrac{[a]X \wedge \mathsf{tt} \wedge \mathsf{tt} \wedge [b]X_1}{\dfrac{[a]X_1 \wedge \mathsf{tt}}{\dfrac{[a]X_1}{\dfrac{X_1}{X_1}\,(\mathsf{ff})}\,([a]) \quad \dfrac{\mathsf{tt}}{\mathsf{tt}}\,([a,b])}\,(\wedge) \qquad \dfrac{\mathsf{tt} \wedge [b]X_1}{\dfrac{\mathsf{tt}}{\mathsf{tt}}\,([a,b]) \quad \dfrac{[b]X_1}{\dfrac{X_1}{X_1}\,(\mathsf{ff})}\,([b])}\,(\wedge)}\,(\wedge)}\,([a])}\,(\vee)}\,(\mathsf{max})$$

┘

▶ **Lemma 37.** *Given a closed formula Ψ of* recHML without min *operators or existential modalities, $f_3(\Psi)$ has the same* sHML *consequences as Ψ.*

Proof sketch. The proof of this lemma rests on the observation that all violations of $f_3(\Psi)$ and Ψ correspond to a single path in $\mathcal{T}(\Psi)$. We can then use the two labellings of $\mathcal{T}(\Psi)$ to move between the witnesses that we use for the violation of $f_3(\Psi)$ and Ψ. ◀

5.4 The strongest sHML consequence

▶ **Theorem 38.** *$f_3 \circ f_2 \circ f_1(\Psi)$ is the strongest* sHML *consequence of any closed $\Psi \in$* recHML*.*

Proof. Follows from Lems. 29 and 37 and Def. 31. By construction $f_3 \circ f_2 \circ f_1(\Psi) \in$ sHML. Moreover, $f_3 \circ f_2 \circ f_1(\Psi)$ has the same sHML consequences as Ψ, making it the strongest sHML consequence of Ψ. ◀

We can symmetrically compute the weakest satisfaction-monitorable antecedent of Ψ, in order to synthesize an optimal acceptance-monitor, or construct the weakest satisfaction-monitorable antecedent by negating $f_3 \circ f_2 \circ f_1(\neg\Psi)$ where $\neg\Psi$ is the negation of Ψ in disjunctive form. In principle, one could also consider constructing optimal monitors from *both* violations and satisfactions of a property Ψ, by deducing the strongest violation-monitorable consequence φ_V of Ψ and the weakest satisfaction-monitorable antecedent φ_S of Ψ; the monitors could be used in tandem to detect all possible satisfactions or violations for Ψ. However, in a branching-time setting either φ_V or φ_S must be trivial:

▶ **Proposition 39.** *For any branching-time property P, its strongest monitorable consequence P_V and its weakest monitorable antecedent P_S, we either have $P_V = \textsc{Prc}$ or $P_S = \emptyset$.* ◀

Proof. If there is a process $p \notin P_V$ and a process $q \in P_S$, then by merging the initial states of p and q we obtain a process that covers p and therefore violates P_V and therefore also P, and that covers q and therefore satisfies P_S and therefore also P, a contradiction. ◀

5.5 Complexity

Eliminating existential modalities and fixpoints does not increase the size of a formula. However, the two tableau constructions used – the first one required to turn the initial formula into disjunctive form, and the second one used to eliminate disjunctions – each can cause an exponential blow-up.

Morally, this is just the cost of determinising alternating automata (already double exponential for finite automata [16]): the automaton corresponding to our final formula, obtained via standard formula-automata correspondences [20], is deterministic (even though automata over trees are not in general determinisable). Indeed, the synthesis from [26], when applied to the formulas we obtain, yields deterministic monitors, in the sense of [6], because our formulas contain no disjunctions, and only conjunctions over disjoint modalities (of the form $\bigwedge_{a\in A}[a]\psi_a$). Whether a more compact non-deterministic monitor can be synthesised instead, or whether the last step, of constructing $f_3(-)$, can be implemented on-the-fly (in the spirit of [35]) is left for future work.

This double-exponential complexity is already present, and necessary, in the corresponding linear-time problem computing a deterministic automaton that recognises the *bad prefixes* of a linear-time property [35]. As Kupferman and Vardi write, this procedure has the flavour of determinisation, hence its double-exponential complexity. Our procedure, despite the added complications associated with branching-time, follows the same principle without a significant additional cost. Interestingly, obtaining the strongest monitorable consequence of an LTL formula in LTL form is much harder. While the (counter-free) non-deterministic automaton that recognises executions without bad prefixes, *i.e.,* the strongest monitorable consequence of an LTL formula, requires exponential blow-up, the best procedure known to date to go from a (counter-free) non-deterministic automaton to an LTL formula uses star-free regular expressions and does not have an elementary complexity upper bound [41, 45].

On a more pragmatic note, both f_1 and f_2 only simplify formulas while f_3 eliminates subformulae containing mixed modalities $[a]\psi \wedge [b]\varphi$, so blow-ups can only occur in f_3 if disjunctions and modalities over the same action interact in a pathological way.

6 Related Work

Linear- vs. Branching-time. Runtime monitoring can be used to verify whether an execution satisfies a linear-time property, for example before the output of a third party component is used as input for a critical component. It can also be used to verify whether a system satisfies a branching-time property, for example as a best-effort light-weight verification strategy. The branching-time properties that one verifies at runtime often consist of properties of the form "on all paths, φ_L holds", where φ_L is a linear-time property. For these kinds of properties, the distinction between the branching-time and linear-time cases can be subtle. In particular, the branching-time case is then implicitly reduced to the linear-time case, *i.e.,* just checking for violations of φ_L. However, in this situation it *only* makes sense to check for violations of φ_L, as satisfactions do not give enough information to deduce anything about the system itself. In contrast, if we are interested in truly linear-time properties, then a monitor can simultaneously check both for violations and satisfactions, as it is done in [27].

Here we are in the branching-time setting: the prior knowledge can be an arbitrary branching-time property, and the property to be monitored can either be a linear-time property quantified universally over all branches, or any other branching-time property. Note that given an LTL formula φ_L, there are standard translations to build a recHML formula φ_B such that φ_B holds in a system if and only if φ_L holds in all of its executions [15]. These can be used to combine a linear-time property, to verify at runtime, with a branching-time property representing the prior knowledge.

As discussed in Sec. 5.5, finding optimal monitors for properties over infinite traces corresponds to computing the good/bad prefixes of the property. Kupferman and Vardi [35] describe how to do this for safety properties described as LTL formulas or Büchi automata. Havelund and Peled [28] describe the same procedure for arbitrary trace properties.

Hierarchies of monitorability. There are many definitions of monitorability (surveyed in [5]) and property classifications (for instance [28, 44]) that help us understand the guarantees we can expect from RV tools for different properties. However monitorable a property is, its optimal monitor is by definition the gold standard to which any RV tool can aspire. Optimal monitors might help determine the degree of monitorability of a property.

Monitoring with prior knowledge. Recently, Henzinger and Saraç [30] studied how assumptions (prior knowledge) can make non-monitorable linear-time properties monitorable. Interestingly, in their setting a property P is not monitorable under assumption K if and only if $P \wedge K$ is monitorable without assumptions, as is the case here. This is because they study a different definition of monitorability, which is not as well behaved under assumption as the notion we use. (Our choice of notion of monitorability is utilitarian: it enables us to compute optimal monitors.) Independently, Cimatti *et al.* and Leucker have also considered a form of monitoring of linear-time properties with (linear-time) prior knowledge in [17, 38]. Leucker proposes an LTL semantics parameterised by this prior knowledge while Cimatti *et al.* incorporate the assumption directly into the monitoring algorithm, thereby treating violations of the assumptions and violations of the property to be monitored differently. Stucki *et al.* [49] parameterise monitorability for hyperproperties with the system under consideration. Their notion of *perfect monitor* corresponds to our *optimal monitor.* Although the authors in [22] study the decidability of monitorability for hyperproperties, neither work describes methods for computing the optimal monitors of hyperproperties.

Multi-valued logics. Logics with three-valued semantics (yes, no, indecisive) can be used to describe monitors [12, 21, 19]. However, whether monitor semantics are given by a many-valued logic or other means, questions of soundness, completeness and optimality with respect to the (two valued) specification formula remain the same.

Monitoring for under-specified components. In orthogonal work that has similarities with ours, Sistla and co-authors [42, 48, 47] address the following problem: given an under-specification φ, and a goal specification ψ, compute a safety property θ such that $\varphi \wedge \theta \implies \psi$. The intuition for this is that if φ is assumed, and violations of θ can be monitored at runtime, then ψ can be assumed whenever the monitor does not detect a violation of θ. This problem then reduces to computing a *safety antecedent* of a specification, namely $\neg\varphi \wedge \psi$. Unlike the strongest monitorable consequence, there is no weakest safety antecedent: properties can be approximated from below with arbitrary precision using a safety formula.

7 Conclusion

We have shown how to compute optimal monitors for arbitrary regular branching-time properties, following a procedure which is sound for arbitrary (not just regular) properties. Our core insight is that the theory of runtime monitors can be extended to the (partial) verification of specifications previously dismissed as unmonitorable, such as most branching-time properties. In particular, this enables us to integrate any prior contextual knowledge of the system into our monitors. We show that this is indeed the best a monitor can do.

References

1 Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. Monitoring for silent actions. In Satya Lokam and R. Ramanujam, editors, *FSTTCS*, volume 93 of *LIPIcs*, pages 7:1–7:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

2 Luca Aceto, Antonis Achilleos, Adrian Francalanza, and Anna Ingólfsdóttir. A framework for parameterized monitorability. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018*, volume 10803 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2018. doi:10.1007/978-3-319-89366-2_11.

3 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Sævar Örn Kjartansson. On the complexity of determinizing monitors. In Arnaud Carayol and Cyril Nicaud, editors, *Implementation and Application of Automata - 22nd International Conference, CIAA 2017*, volume 10329 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2017. doi:10.1007/978-3-319-60134-2_1.

4 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Adventures in monitorability: From branching to linear time and back again. *Proceedings*

of the ACM on Programming Languages, 3(POPL):52:1–52:29, 2019. URL: `https://dl.acm.org/citation.cfm?id=3290365`.

5 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. An operational guide to monitorability. In *Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings*, volume 11724 of *LNCS*, pages 433–453. Springer, 2019. `doi:10.1007/978-3-030-30446-1_23`.

6 Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Sævar Örn Kjartansson. Determinizing monitors for HML with recursion. *Journal of Logical and Algebraic Methods in Programming*, 111:100515, February 2020. `doi:10.1016/j.jlamp.2019.100515`.

7 Bowen Alpern and Fred B Schneider. Defining liveness. *Information processing letters*, 21(4):181–185, 1985.

8 Duncan Paul Attard, Ian Cassar, Adrian Francalanza, Luca Aceto, and Anna Ingolfsdottir. *Behavioural Types: from Theory to Tools*, chapter A Runtime Monitoring Tool for Actor-Based Systems, pages 49–74. River Publishers, 2017.

9 Duncan Paul Attard and Adrian Francalanza. A monitoring tool for a branching-time logic. In Yliès Falcone and César Sánchez, editors, *Runtime Verification - 16th International Conference, RV 2016*, volume 10012 of *Lecture Notes in Computer Science*, pages 473–481. Springer, 2016. `doi:10.1007/978-3-319-46982-9_31`.

10 Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking.* MIT press, 2008.

11 Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. *Introduction to Runtime Verification*, pages 1–33. Springer International Publishing, Cham, 2018. `doi:10.1007/978-3-319-75632-5_1`.

12 Andreas Bauer, Martin Leucker, and Christian Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In Oleg Sokolsky and Serdar Taşıran, editors, *Runtime Verification*, pages 126–138, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

13 Julian Bradfield and Colin Stirling. Chapter 4 - Modal logics and mu-calculi: An introduction. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 293–330. Elsevier Science, Amsterdam, 2001. `doi:10.1016/B978-044482830-9/50022-9`.

14 Julian Bradfield and Colin Stirling. Modal μ-calculi. *Studies in Logic and Practical Reasoning*, 3:721–756, 2007.

15 Julian Bradfield and Igor Walukiewicz. *The mu-calculus and Model Checking*, pages 871–919. Springer, May 2018. `doi:10.1007/978-3-319-10575-8_26`.

16 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, January 1981. `doi:10.1145/322234.322243`.

17 Alessandro Cimatti, Chun Tian, and Stefano Tonetta. Assumption-based runtime verification with partial observability and resets. In *International Conference on Runtime Verification*, pages 165–184. Springer, 2019.

18 Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model Checking.* MIT press, 1999.

19 Volker Diekert and Martin Leucker. Topology, monitorable properties and runtime verification. *Theoretical Computer Science*, 537:29–41, 2014. Theoretical Aspects of Computing (ICTAC 2011). `doi:10.1016/j.tcs.2014.02.052`.

20 E Allen Emerson and Charanjit S Jutla. Tree automata, mu-calculus and determinacy. In *FoCS*, volume 91, pages 368–377. Citeseer, 1991.

21 Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 14(3):349–382, 2012. `doi:10.1007/s10009-011-0196-8`.

22 Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Monitoring hyperproperties. *Formal Methods in System Design*, 54(3):336–363, 2019.

23 Adrian Francalanza. A Theory of Monitors (Extended Abstract). In *FoSSaCS*, volume 9634 of *LNCS*, pages 145–161, 2016.

24 Adrian Francalanza. Consistently-detecting monitors. In Roland Meyer and Uwe Nestmann, editors, *28^{th} International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *LIPIcs*, pages 8:1–8:19, Dagstuhl, Germany, 2017. Schloss Dagstuhl. doi:10.4230/LIPIcs.CONCUR.2017.8.

25 Adrian Francalanza, Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Ian Cassar, Dario Della Monica, and Anna Ingólfsdóttir. A foundation for runtime monitoring. In Shuvendu K. Lahiri and Giles Reger, editors, *Runtime Verification - 17th International Conference, RV 2017*, volume 10548 of *Lecture Notes in Computer Science*, pages 8–29. Springer, 2017. doi:10.1007/978-3-319-67531-2_2.

26 Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. Monitorability for the Hennessy-Milner logic with recursion. *Formal Methods in System Design*, 51(1):87–116, 2017. doi:10.1007/s10703-017-0273-z.

27 M.C.W. Geilen. On the construction of monitors for temporal logic properties. *Electronic Notes in Theoretical Computer Science*, 55(2):181–199, 2001. RV'2001, Runtime Verification (in connection with CAV '01). doi:10.1016/S1571-0661(04)00252-X.

28 Klaus Havelund and Doron Peled. Runtime Verification: From Propositional to First-Order Temporal Logic. In *Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings*, volume 11237 of *LNCS*, pages 90–112. Springer, 2018. doi:10.1007/978-3-030-03769-7_7.

29 Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In *TACAS*, volume 2, pages 342–356. Springer, 2002.

30 Thomas A Henzinger and N Ege Saraç. Monitorability under assumptions. In *International Conference on Runtime Verification*, pages 3–18. Springer, 2020.

31 David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *International Conference on Concurrency Theory*, pages 263–277. Springer, 1996.

32 David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *CONCUR '96: Concurrency Theory*, pages 263–277. Springer Berlin Heidelberg, 1996. doi:10.1007/3-540-61604-7_60.

33 Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976. doi:10.1145/360248.360251.

34 Dexter C. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

35 Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

36 Orna Kupferman, Moshe Y Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

37 Kim G. Larsen. Proof Systems for Satisfiability in Hennessy-Milner Logic with recursion. *Theoretical Computer Science*, 72(2):265–288, 1990. doi:10.1016/0304-3975(90)90038-J.

38 Martin Leucker. Sliding between model checking and runtime verification. In *International Conference on Runtime Verification*, pages 82–87. Springer, 2012.

39 Martin Leucker, César Sánchez, Torben Scheffel, Malte Schmitz, and Daniel Thoma. Runtime verification for timed event streams with partial information. In Bernd Finkbeiner and Leonardo Mariani, editors, *Runtime Verification - 19th International Conference, RV 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11757 of *LNCS*, pages 273–291. Springer, 2019. doi:10.1007/978-3-030-32079-9_16.

40 Grgur Petric Maretić, Mohammad Torabi Dashti, and David Basin. Ltl is closed under topological closure. *Information Processing Letters*, 114(8):408–413, 2014.

41 Grgur Petric Maretić, Mohammad Torabi Dashti, and David Basin. Ltl is closed under topological closure. *Information Processing Letters*, 114(8):408–413, 2014.

42 Tiziana Margaria, A. Prasad Sistla, Bernhard Steffen, and Lenore D. Zuck. Taming interface specifications. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 – Concurrency Theory*, pages 548–561, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

43 R Milner. A calculus of communicating systems. *Lecture Notes in Comput. Sci. 92*, 1980.

44 Doron Peled and Klaus Havelund. Refining the safety–liveness classification of temporal properties according to monitorability. In *Models, Mindsets, Meta: The What, the How, and the Why Not?*, pages 218–234. Springer, 2019.

45 A Peuli and Lenore Zuck. In and out of temporal logic. In *[1993] Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 124–135. IEEE, 1993.

46 Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

47 A Prasad Sistla and Abhigna R Srinivas. Monitoring temporal properties of stochastic systems. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 294–308. Springer, 2008.

48 A Prasad Sistla, Min Zhou, and Lenore D Zuck. Monitoring off-the-shelf components. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 222–236. Springer, 2006.

49 Sandro Stucki, César Sánchez, Gerardo Schneider, and Borzoo Bonakdarpour. Gray-box monitoring of hyperproperties. In *Formal Methods–The Next 30 Years: Third World Congress, FM 2019, Porto, Portugal, October 7–11, 2019, Proceedings*, volume 11800, page 406. Springer Nature, 2019.

50 Mahesh Viswanathan and Moonzoo Kim. Foundations for the run-time monitoring of reactive systems - fundamentals of the MaC language. In Zhiming Liu and Keijiro Araki, editors, *Theoretical Aspects of Computing - ICTAC 2004, First International Colloquium*, volume 3407 of *Lecture Notes in Computer Science*, pages 543–556. Springer, 2004. `doi:10.1007/978-3-540-31862-0_38`.

51 Igor Walukiewicz. Completeness of Kozen's axiomatisation of the propositional μ-calculus. *Information and Computation*, 157(1-2):142–182, February 2000. `doi:10.1006/inco.1999.2836`.

A Technical Proofs

In our proofs, instead of working with the classical semantics, we use *consistent annotations* and *counter-annotations* which respectively witness that a property holds or does not hold for a process. The intuition is that an evaluation of $\psi \vee \varphi$ to true must also evaluate either ψ or φ to true, and an annotation indicates which one. Similarly, for $\langle a \rangle \psi$ to be true at a state, one of the state's a-successors must be annotated with ψ.

▶ **Example 40.** The witness of the reachability specification $\mathsf{min}\, X.(\langle a \rangle X \vee \langle b \rangle \mathsf{tt})$ (there is a sequence of a-transitions that leads to a b-transition) for a process p would consist of the following annotation: p is annotated with

$$\{\mathsf{min}\, X.\langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle a \rangle X\},$$

a finite sequence of a-successors will be annotated with

$$\{X, \mathsf{min}\, X.\langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle a \rangle X\}$$

and finally an a-successor will be annotated with

$$\{X, \mathsf{min}\, X.\langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle a \rangle X \vee \langle b \rangle \mathsf{tt}, \langle b \rangle \mathsf{tt}\}$$

and its b-successor will be annotated with tt. ◀

In the following, for each formula φ with free variables we consider the closure $c(\varphi)$ of φ, which results by replacing in φ all free variables X by φ_X. We use $[\![\varphi]\!]$ for an open formula φ, to mean $[\![c(\varphi)]\!]$. We say that a formula φ is satisfiable when $[\![\varphi]\!] \neq \emptyset$.

▶ **Definition 41** (Locally consistent annotation). *An* annotation $A : P \to \mathcal{P}(sf(\Psi))$, *where* $P \subseteq$ *Prc is a labelling of* P *(a partial labelling of* Prc*) with sets of subformulae of a closed formula* Ψ *of* recHML*. An annotation is* locally consistent *if for all states* $s \in P$*:*

- $\mathsf{ff} \notin A(s)$*;*
- *If* $\min X.\varphi \in A(s)$ *or* $\max X.\varphi \in A(s)$ *then* $\varphi \in A(s)$*;*
- *If* $X \in A(s)$ *then* $\min X.\varphi \in A(s)$ *if* X *is a least fixpoint variable and* $\max X.\varphi \in A(s)$ *otherwise;*
- *If* $\varphi \wedge \psi \in A(s)$ *then* $\varphi \in A(s)$ *and* $\psi \in A(s)$*;*
- *If* $\varphi \vee \psi \in A(s)$ *then* $\varphi \in A(s)$ *or* $\psi \in A(s)$*;*
- *If* $\langle a\rangle\varphi \in A(s)$ *then* $\varphi \in A(s')$ *for some* $s' \in P$*, such that* $s \xrightarrow{a} s'$*;*
- *If* $[a]\varphi \in A(s)$ *then* $\varphi \in A(s')$ *for all* $s' \in$ Prc*, such that* $s \xrightarrow{a} s'$*.* ◀

▶ **Definition 42.** *For annotation* A*, an* annotated sequence *is a (finite or infinite) sequence* $\pi = (\varphi_0, s_0)(\varphi_1, s_1)\cdots$*, such that*

- *for each* i*,* $\varphi_i \in A(s_0)$*;*
- *for all* $i, i+1$ *that appear as indexes in* π*,* φ_i *is of the form* $\varphi_{i+1} \wedge \psi$*,* $\psi \wedge \varphi_{i+1}$*,* $\varphi_{i+1} \vee \psi$*,* $\psi \vee \varphi_{i+1}$*,* $[a]\varphi_{i+1}$*,* $\langle a\rangle\varphi_{i+1}$*,* $\min X.\varphi_{i+1}$*,* $\min X.\varphi_{i+1}$*, or* X*, where* $\varphi_{i+1} = \min X.\psi$ *or* $\min X.\psi$*;*
- *if* $\varphi_i = [a]\varphi_{i+1}$ *or* $\langle a\rangle\varphi_{i+1}$*, then* $s_i \xrightarrow{a} s_{i+1}$*, and otherwise* $s_i = s_{i+1}$*;* ◀

It is not hard to see that if two fixpoint formulas φ_1, φ_2 appear in an annotated sequence, then in the subsequence between (but including) the respective appearances of φ_1 and φ_2, there appears a fixpoint formula φ_3, such that φ_1 and φ_2 are subformulae of φ_3. Therefore, in every infinite annotated sequence there appears infinitely often a fixpoint formula ψ, such that all other fixpoint formulas that appear infinitely often are subformulae of ψ. Then, ψ is called the outermost fixpoint formula that appears infinitely often in the sequence.

▶ **Definition 43** (Consistent Annotation). *An annotation is* consistent *if it is both locally consistent and for every infinite annotated sequence, the outermost fixpoint formula that appears infinitely often in the sequence is a* max*-formula.* ◀

It is a standard result (see for example [14] for a more thorough discussion) that for a process p and a subformula φ of Ψ, we have that $p \in [\![\varphi]\!]$ if and only if there is a consistent annotation such that $\varphi \in A(p)$. We call this a consistent φ-annotation of p.

We observe that, because formulas are assumed to be guarded, every annotation on processes with no infinite traces is consistent if and only if it is locally consistent. The same is true if no min-fixpoints appear in the annotation.

For convenience, we also define the dual, a consistent counter-annotation, which witnesses that a computation tree violates a property.

▶ **Definition 44** (Consistent counter-annotation). *A counter-annotation* $C : P \to P(sf(\Psi))$ *is a labelling of* $P \subseteq$ Prc *with sets of subformulae of a formula* Ψ *of* recHML*. A counter-annotation is locally consistent if for all states* $s \in P$*:*

- $\mathsf{tt} \notin C(s)$*;*
- *If* $\min X.\varphi \in C(s)$ *or* $\max X.\varphi \in C(s)$ *then* $\varphi \in C(s)$*;*
- *If* $X \in C(s)$ *then* $\min X.\varphi \in C(s)$*;*
- *If* $\varphi \wedge \psi \in C(s)$ *then* $\varphi \in C(s)$ *or* $\psi \in C(s)$*;*

- *If $\varphi \vee \psi \in C(s)$ then $\varphi \in C(s)$ and $\psi \in C(s)$;*
- *If $\langle a\rangle\varphi \in C(s)$ then $\varphi \in C(s')$ for all $s' \in$ Prc, such that $s \xrightarrow{a} s'$;*
- *If $[a]\varphi \in C(s)$ then $\varphi \in C(s')$ for some $s' \in P$, such that $s \xrightarrow{a} s'$.*

Counter-annotated sequences are defined similarly to annotated sequences. A counter-annotation is consistent if it is both locally consistent and for every infinite annotated sequence of subformulae, the outermost fixpoint formula that appears infinitely often in the sequence is a μ-formula. ◀

Then, a process p violates a subformula φ of Ψ if and only if there is a consistent counter-annotation C, such that $\varphi \in C(p)$.

Eliminating existentials

Lemma 29. *For every closed disjunctive* recHML *formula Ψ, the formula $f_1(\Psi)$ has the same* sHML *consequences as Ψ.*

Proof. Observe that we can construct a consistent annotation for $f_1(\Psi)$ from a consistent annotation for Ψ, by simply replacing each ψ in the annotation by $f_1(\psi)$. Then, all conditions for a consistent annotation are satisfied, and therefore Ψ implies $f_1(\Psi)$.

Let $\theta \in$ sHML be a consequence of Ψ. We show that $f_1(\Psi)$ also implies θ

Assume otherwise: let p be a process such that $p \in [\![f_1(\Psi) \wedge \neg\theta]\!]$. Let A_1 be an annotation that witnesses $p \in [\![f_1(\Psi)]\!]$.

We know, by Thm. 22, that θ is monitorable, so there is a finite trace $t = a_1 a_2 \dots a_k$ of p, such that for every p', if t is a trace of p', then $p' \in [\![\neg\theta]\!]$. Let $p = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} p_k$ be states reachable from p while producing t, and let $q_0, q_1, \dots, q_k$ be processes with only the following transitions: $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} q_k$. From the above, we see that $q_0 \notin [\![\theta]\!]$. Furthermore, we can define an annotation A_2 on $\{q_0, q_1, \dots, q_k\}$, such that for all $i = 1, \dots, k$, $A_2(q_i) = A_1(p_i)$. It is not hard to see, exploiting the absence of existential modalities in $f_1(\Psi)$, that A_2 is a consistent annotation, witnessing that $q_0 \in [\![f_1(\Psi)]\!]$.

Let A_3 be a *minimal* consistent annotation witnessing that $q_0 \in [\![f_1(\Psi)]\!]$. Let us observe that by the definition of Ψ and f_1, all formulas in A_3 can be of the form tt, X, $\mathsf{min}\, X.\psi$, $\mathsf{max}\, X.\psi$, $\psi_1 \vee \psi_2$, or $\bigwedge_{a\in B}[a]\bigvee \mathcal{B}_a$. The last kind of formula we call a conjunction. We say that $\bigwedge_{a\in B}[a]\bigvee \mathcal{B}_a \in A_3(q_i)$ is maximal in $A_3(q_i)$ if it is not a conjunct in any other conjunction in $A_3(q_i)$. We now show that due to the disjunctive form of our formulas, a minimal annotation only has one maximal conjunction per state.

We also define recursively for two formulas $\psi_1, \psi_2 \in A_3(q_i)$ what is a path from ψ_1 to ψ_2: $\{\psi_1\}$ is a path from ψ_1 to ψ_1; and if F is a path from ψ_1 to ψ_2, then:

- if $\psi_1 \vee \psi_1' \in A_3(q_i)$, then $F \cup \{\psi_1 \vee \psi_1'\}$ is a path from $\psi_1 \vee \psi_1'$ to ψ_2;
- if $\psi_1' \vee \psi_1 \in A_3(q_i)$, then $F \cup \{\psi_1' \vee \psi_1\}$ is a path from $\psi_1' \vee \psi_1$ to ψ_2;
- if $\mathsf{max}\, X.\psi_1 \in A_3(q_i)$, then $F \cup \{\mathsf{max}\, X.\psi_1\}$ is a path from $\mathsf{max}\, X.\psi_1$ to ψ_2; and
- if $\mathsf{min}\, X.\psi_1 \in A_3(q_i)$, then $F \cup \{\mathsf{min}\, X.\psi_1\}$ is a path from $\mathsf{min}\, X.\psi_1$ to ψ_2.

Finally, for each $\bigwedge_{a\in B}[a]\bigvee \mathcal{B}_a \in A_3(q_i)$, we define the set of conjunctions that it subsumes, in the following sense:

$$ss\left(\bigwedge_{a\in B}[a]\bigvee \mathcal{B}_a \in A_3(q_i)\right) = \left\{\bigwedge_{a\in C}[a]\bigvee \mathcal{B}_a \in A_3(q_i) \mid \emptyset \neq C \subseteq B\right\}.$$

We are now ready to prove the following claim on the minimal annotation A_3, which will allow us to focus on a single maximal conjunction per state:

Claim: for every i, if $\bigwedge_{a\in B}[a]\bigvee \mathcal{B}_a \in A_3(q_i)$ and $\bigwedge_{a\in B'}[a]\bigvee \mathcal{B}'_a \in A_3(q_i)$, are maximal in $A_3(q_i)$, then $B = B'$ and for all $a \in B$, $\mathcal{B}_a = \mathcal{B}'_a$ – in other words, there is at most one maximal conjunction in $A_3(q_i)$

We prove the claim by induction on i. *For the case where $i = 0$,* we first observe that $f_1(\Psi) \in A_3(q_0)$. Since our formulas are guarded and $f_1(\Psi)$ is closed, there is no path from $f_1(\Psi)$ to a variable X. Therefore, according to the conditions for local consistency, there must be a path F from $f_1(\Psi)$ to either tt or to a conjunction ψ_c. In the first case, we observe that there can be no conjunction in F (by the definition of a path), and substituting $A_3(q_0)$ by F results in a locally consistent annotation, and therefore, $A_3(q_0) = F$ as A_3 is minimal. In the second case, we observe that there can be no other conjunction in F (by the definition of a path), and that substituting $A_3(q_0)$ by $F \cup ss(\psi_c)$ (which is a subset of $A_3(q_0)$) results in a locally consistent annotation, and therefore, by minimality, $A_3(q_0) = F \cup ss(\psi_c)$. Therefore, in both cases, there is at most one maximal conjunction in $A_3(q_0)$.

We now tackle the case for $i > 0$. By the inductive hypothesis, there is at most one maximal conjunction $\bigwedge_{a\in B''}[a]\bigvee \mathcal{B}''_a \in A_3(q_{i-1})$. If there is none, or if $a_i \notin B''$, then $A_3(q_i) = \emptyset$ and we are done. Otherwise, let $\psi_1 = \bigvee \mathcal{B}''_{a_i}$. By the requirements of local consistency, $\psi_1 \in A_3(q_i)$, and there is a path F_1 from ψ_1 to tt, to a conjunction ψ_2, or to a variable X_1. We can handle the first two cases similarly for the case of $i = 0$. For the last case, from the requirements of local consistency, for some $k > 0$, we can construct k paths, F_j, $1 \le j \le k$, such that F_1 is as defined above, and for $1 < j < k$, F_j is a path from $\mathsf{max}\, X_{j-1}.\psi_j$ to X_j (with $X_j \neq X_{j'}$ for $j \neq j'$), and, due to the guardedness of our formulas and the finiteness of $A_3(q_i)$, F_k is a path from $\mathsf{max}\, X_{k-1}.\psi_k$ to ψ_k, where $\psi_k = \mathsf{tt}$ or ψ_k is a conjunction. Let $F = \bigcup_{j=1}^{k} F_i$. With the possible exception of ψ_k, there is no conjunction in F. In the case $\psi_k = \mathsf{tt}$, substituting $A_3(q_0)$ by F results in a locally consistent annotation, and therefore, $A_3(q_0) = F$. In the case ψ_k a conjunction, substituting $A_3(q_0)$ by $F \cup ss(\psi_2)$ results in a locally consistent annotation, and therefore, $A_3(q_0) = F \cup ss(\psi_2)$. Therefore, in both cases, there is at most one maximal conjunction in $A_3(q_0)$.

We can now use these maximal conjunctions (which all come from the elimination of existential modalities from the conjunctions of Ψ) to turn q_0 into a process that also satisfies Ψ, by adding successors that satisfy the consistency requirements of the eliminated existential modalities.

We have assumed that all subformulae of disjunctive formulas (except for ff) are satisfiable. Therefore, for every subformula $\langle a\rangle\psi$ of Ψ, we can fix a process $s_\psi \in [\![\psi]\!]$, and assume a consistent annotation A_4 that witnesses these facts. We now construct a process $r \in [\![\Psi]\!]$, such that t is a trace of r. For each q_i, $i = 0, \ldots, k$, we construct a process q'_i with exactly the following transitions: $q'_i \xrightarrow{a_{i+1}} q'_{i+1}$, if $i < k$, and $q'_i \xrightarrow{a} s_\psi$ for every $\psi \in \mathcal{B}_a$, for every $f_1((\bigwedge_{\psi\in\mathcal{B}_a}\langle a\rangle\psi) \wedge [a]\bigvee \mathcal{B}_a) \in A_3(q_i)$. We can now construct a consistent annotation A_5 to witness that $q'_i \in [\![\psi]\!]$, for every $f(\psi) \in A_3(q_i)$. For each subformula $(\bigwedge_{\psi\in\mathcal{B}_a}\langle a\rangle\psi) \wedge [a]\bigvee \mathcal{B}_a$ of Ψ and every $\psi \in \mathcal{B}_a$, $A_5(s_\psi) = A_4(s_\psi) \cup \{\bigvee \mathcal{B}_a\}$; for $i = 0, \ldots, k$, $A_5(q'_i) = \{\psi \in sf(\Psi) \mid f_1(\psi) \in A_3(q'_i)\}$, and for every other state s, $A_5(s) = A_4(s)$ if A_4 is defined on s. It is then not hard to see that all conditions for a consistent annotation are satisfied by A_5. Therefore, A_5 witnesses that $r \overset{\text{def}}{=} q'_0 \in [\![\Psi]\!]$. Furthermore, it is immediately evident that t is a trace of r, and therefore $r \notin [\![\theta]\!]$, and therefore θ cannot be a consequence of Ψ, contradicting our assumptions. This completes the proof of the lemma. ◀

▶ **Example 45.** The necessity of disjunctive form can be seen from the following example: $\psi = (\langle a\rangle[b]\mathsf{ff}) \wedge ([a]\langle b\rangle\mathsf{tt} \vee [a][c]\mathsf{ff})$. For $F = \{[b]\mathsf{ff} \wedge [c]\mathsf{ff}, [c]\mathsf{ff}\}$, the equivalent disjunctive formula is:

$$\bigwedge_{\varphi\in F} \langle a\rangle\varphi \wedge [a] \bigvee F.$$

In ψ, replacing existentials with tt would yield a formula itself equivalent to tt. However, from its disjunctive form we can extract its strongest sHML consequence $[a][c]\mathsf{ff}$ (rather than tt). ◀

Eliminating least fixpoints

Lemma 32. *For every closed formula Ψ of* recHML *without existentials, $f_2(\Psi)$ has the same* sHML *consequences as Ψ.*

Proof. First, observe that Ψ implies $f_2(\Psi)$: an annotation for Ψ is locally consistent, so by replacing all occurrences of min by max, we are certain to have no sequences with infinite occurrences of min-formulas, so we have a consistent annotation for $f_2(\Psi)$.

Now, let $\theta \in$ sHML, such that θ is not a consequence of $f_2(\Psi)$. We prove that θ is also not a consequence of Ψ, which completes the proof of the lemma. Since θ is not a consequence of $f_2(\Psi)$, there is a $p \in [\![f_2(\Psi) \wedge \neg\theta]\!]$. Similarly to the proof of Lem. 29, we know, by Thm. 22, that θ is monitorable, so we can construct a process q that has no infinite traces and satisfies $f_2(\Psi) \wedge \neg\theta$. Let A be a consistent annotation that witnesses the fact. From A, we can then construct an annotation A': $A'(s) = \{\psi \in sf(\Psi) \mid f_2(\psi) \in A(s)\}$, when $A(s)$ is defined. It is straightforward to see that A' is locally consistent, using the fact that A is locally consistent. It is also consistent, because q has no infinite traces. Therefore, A' witnesses that $q \in [\![\Psi]\!]$, which completes the proof. ◀

A.1 Eliminating disjunctions

Lemma 37. *Given a closed formula Ψ of* recHML *with neither* min *operators nor existentials, $f_3(\Psi)$ has the same* sHML *consequences as Ψ.*

Proof. We fix a tableau $\mathcal{T}(\Psi)$ and the corresponding labellings L and L' of its nodes, as defined in Def. 35.

We first show that $f_3(\Psi)$ is a consequence of Ψ, *i.e.*, $\neg f_3(\Psi)$ implies $\neg\Psi$. Let p be a process such that $p \notin [\![f_3(\Psi)]\!]$. Since $f_3(\Psi)$ is a sHML formula, and similarly to the proofs of Lems. 29 and 32, we can assume that process p has a single maximal trace t. Let $C : P \to \mathcal{P}(sf(f_3(\Psi)))$ be a counter-annotation that witnesses the fact that $p \notin [\![f_3(\Psi)]\!]$. Since p only has finite traces and $f_3(\Psi)$ is guarded, C has no infinite counter-annotated sequences. Therefore, ff appears somewhere in C. We now define C', a counter annotation for Ψ that is defined on the set $P \subseteq$ Prc of processes that are reachable from p by a (possibly empty) sequence of transitions:

$$C'(q) = \{\psi \in L(n) \mid n \text{ is a tableau node s.t. } L'(n) \in C(q)\},$$

for every $q \in P$. It is then, not hard to verify that C' is locally consistent, and therefore it is also consistent, thus witnessing that $p \notin [\![\Psi]\!]$.

We now show that if Ψ implies a formula $\theta \in$ sHML, then $f_3(\Psi)$ also implies θ. Assume that Ψ implies $\theta \in$ sHML. Let p be a process such that $p \notin [\![\theta]\!]$ – therefore, $p \notin [\![\Psi]\!]$. Since θ

is a sHML formula, as above, we can assume that process p has a single maximal trace t. Let C be a consistent counter-annotation that witnesses that $p \notin [\![\Psi]\!]$, defined over $P \subseteq \textsc{Prc}$. We now define C', a counter annotation for $f_3(\Psi)$ that is defined on $P' = \{q \in P \mid C(q) \neq \emptyset\}$:

$$C'(q) = \{L'(n) \mid n \text{ is a tableau node s.t. } L(n) \subseteq C(q)\},$$

for every $q \in P'$. Again, it is not hard to verify that C' is locally consistent – and since p has no infinite traces and our formulas are guarded, C' is also consistent. Therefore, $p \notin [\![f_3(\Psi)]\!]$, so we have showed that $f_3(\Psi)$ implies all sHML consequences of Ψ, which completes the proof. ◀

Are Two Binary Operators Necessary to Finitely Axiomatise Parallel Composition?

Luca Aceto
Reykjavik University, Iceland
Gran Sasso Science Institute, L'Aquila, Italy

Valentina Castiglioni
Reykjavik University, Iceland

Wan Fokkink
Vrije Universiteit Amsterdam, The Netherlands

Anna Ingólfsdóttir
Reykjavik University, Iceland

Bas Luttik
Eindhoven University of Technology, The Netherlands

Abstract

Bergstra and Klop have shown that *bisimilarity* has a *finite* equational axiomatisation over ACP/CCS extended with the binary *left* and *communication merge* operators. Moller proved that auxiliary operators are *necessary* to obtain a finite axiomatisation of bisimilarity over CCS, and Aceto et al. showed that this remains true when *Hennessy's merge* is added to that language. These results raise the question of whether there is *one* auxiliary *binary* operator whose addition to CCS leads to a finite axiomatisation of bisimilarity. This study provides a *negative answer* to that question based on three reasonable assumptions.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Process calculi; Theory of computation → Operational semantics

Keywords and phrases Equational logic, CCS, bisimulation, parallel composition, non-finitely based algebras

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.8

Related Version An extended version of the paper, with all technical proofs, is available at `https://arxiv.org/abs/2010.01943`.

Funding This work has been supported by the project '*Open Problems in the Equational Logic of Processes*' (OPEL) of the Icelandic Research Fund (grant No. 196050-051).

Acknowledgements We thank the anonymous reviewers for their valuable comments.

1 Introduction

The purpose of this paper is to provide an answer to the following problem (see [1, Problem 8]): *Are the left merge and the communication merge operators* necessary *to obtain a finite equational axiomatisation of bisimilarity over the language CCS?* The interest in this problem is threefold, as an answer to it would:

1. provide the first study on the finite axiomatisability of operators whose operational semantics is not determined a priori,
2. clarify the status of the auxiliary operators *left merge* and *communication merge*, proposed in [10], in the finite axiomatisation of parallel composition, and
3. give further insight into properties that auxiliary operators used in the finite equational characterisation of parallel composition ought to afford.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 8; pp. 8:1–8:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We prove that, under some reasonable simplifying assumptions, whose role in our technical developments we discuss below, there is no auxiliary binary operator that can be added to CCS to yield a finite equational axiomatisation of bisimilarity. Despite falling short of solving the above-mentioned problem in full generality, our negative result is a substantial generalisation of previous non-finite-axiomatisability theorems by Moller [19, 20] and Aceto et al. [4].

In order to put our contribution in context, we first describe the history of the problem we tackle and then give a bird's eye view of our results.

The story so far. In the late 1970s, Milner developed the *Calculus of Communicating Systems* (CCS) [17], a formal language based on a message-passing paradigm and aimed at describing communicating processes from an operational point of view. In detail, a *labelled transition system* (LTS) [16] was used to equip language expressions with an *operational semantics* [23] and was defined using a collection of syntax-driven rules. The analysis of process behaviour was carried out via an observational *bisimulation*-based theory [22] that defines when two states in an LTS describe the same behaviour. In particular, CCS included a *parallel composition operator* $\|$ to model the interactions among processes. Such an operator, also known as *merge* [10, 11], allows one both to *interleave* the behaviours of its argument processes (modelling concurrent computations) and to enable some form of *synchronisation* between them (modelling interactions). Later on, in collaboration with Hennessy, Milner studied the *equational theory* of (recursion free) CCS and proposed a *ground-complete axiomatisation* for it modulo bisimilarity [15]. More precisely, Hennessy and Milner presented a set $\mathcal{E}$ of *equational axioms* from which all equations over closed CCS terms (namely those with no occurrences of variables) that are *valid modulo bisimilarity* can be derived using the rules of *equational logic* [24]. Notably, the set $\mathcal{E}$ included infinitely many axioms, which were instances of the *expansion law* that was used to "simulate equationally" the operational semantics of the parallel composition operator.

The ground-completeness result by Hennessy and Milner started the quest for a finite axiomatisation of CCS's parallel composition operator modulo bisimilarity.

Bergstra and Klop showed in [10] that a finite ground-complete axiomatisation modulo bisimilarity can be obtained by enriching CCS with two auxiliary operators, namely the *left merge* $\mathbin{\|\!\_}$ and the *communication merge* $|$, expressing respectively one step in the asymmetric pure interleaving and the synchronous behaviour of $\|$. Their result was then strengthened by Aceto et al. in [6], where it is proved that, over the fragment of CCS without recursion, restriction and relabelling, the auxiliary operators $\mathbin{\|\!\_}$ and $|$ allow for finitely axiomatising $\|$ modulo bisimilarity also when CCS terms with variables are considered. Moreover, in [8] that result is extended to the fragment of CCS with relabelling and restriction, but without communication. From those studies, we can infer that the left merge and communication merge operators are *sufficient* to finitely axiomatise parallel composition modulo bisimilarity. But is the addition of auxiliary operators *necessary* to obtain a finite equational axiomatisation, or can the use of the expansion law in the original axiomatisation of bisimilarity by Hennessy and Milner be replaced by a finite set of sound CCS equations?

To address that question, in [19, 20] Moller considered a minimal fragment of CCS, including only action prefixing, nondeterministic choice and interleaving, and proved that, even in the presence of a single action, bisimilarity does not afford a finite ground-complete axiomatisation over the closed terms in that language. This showed that auxiliary operators are indeed necessary to obtain a finite equational axiomatisation of bisimilarity. Adapting Moller's proof technique, Aceto et al. proved, in [4], that if we replace $\mathbin{\|\!\_}$ and $|$ with the so called

Hennessy's merge $|\!\!/$ [14], which denotes an asymmetric interleaving with communication, then the collection of equations that hold modulo bisimilarity over the recursion, restriction and relabelling free fragment of CCS enriched with $|\!\!/$ is not finitely based (in the presence of at least two distinct complementary actions).

A natural question that arises from those *negative* results is the following:

> *Can one obtain a finite axiomatisation of the parallel composition operator in bisimulation semantics by adding* only one binary operator *to the signature of (recursion, restriction, and relabelling free) CCS?* (P)

In this paper, we provide a partial *negative answer* to that question. (Note that, in (P), we focus on binary operators, like all the variations on parallel composition mentioned above, since using a ternary operator one can express the left and communication merge operators and, in fact, an arbitrary number of binary operators.)

Our contribution. We analyse the axiomatisability of parallel composition over the language CCS_f, namely CCS enriched with a binary operator f that we use to express $\|$ as a derived operator. We prove that, under three reasonable assumptions, an auxiliary operator f alone does not allow us to obtain a finite ground-complete axiomatisation of CCS_f modulo bisimilarity.

To this end, the only knowledge we assume on the operational semantics of f is that it is formally defined by rules in the de Simone format [13] (Assumption 1) and that the behaviour of the parallel composition operator is expressed equationally by a law that is akin to the one used by Bergstra and Klop to define $\|$ in terms of $\llfloor$ and $|$ (Assumption 2). We then argue that the latter assumption yields that the equation

$$x\|y \approx f(x,y) + f(y,x) \tag{A}$$

is valid modulo bisimilarity. Next we proceed by a case analysis over the possible sets of de Simone rules defining the behaviour of f, in such a way that the validity of Equation (A) modulo bisimilarity is guaranteed. To fully characterise the sets of rules that may define f, we introduce a third simplifying assumption: the target of each rule for f is either a variable or a term obtained by applying a single CCS_f operator to the variables of the rule, according to the constraints of the de Simone format (Assumption 3). Then, for each of the resulting cases, we show the desired negative result using proof-theoretic techniques that have their roots in Moller's classic results in [19, 20]. This means that we identify a (case-specific) property of terms denoted by W_n for $n \geq 0$. The idea is that, when n is *large enough*, W_n is preserved by provability from finite, sound axiom systems. Hence, whenever $\mathcal{E}$ is a finite, sound axiom system and an equation $p \approx q$ is derivable from $\mathcal{E}$, then either both terms p and q satisfy W_n, or none of them does. The negative result is then obtained by exhibiting a (case-specific) infinite family of valid equations $\{e_n \mid n \geq 0\}$ in which W_n is not preserved, that is, for each $n \geq 0$, W_n is satisfied only by one side of e_n. Due to the choice of W_n, this means that the equations in the family cannot all be derived from a finite set of valid axioms and therefore no finite, sound axiom system can be complete.

To the best of our knowledge, in this paper we propose the first non-finite axiomatisability result for a process algebra in which one of the operators, namely the auxiliary operator f, does not have a fixed semantics. However, for our technical developments, it has been necessary to restrict the search space for f by means of the aforementioned simplifying assumptions. To our mind, those assumptions are "reasonable" because they allow us to simplify the combinatorial complexity of our analysis without excessively narrowing down the set of operators captured by our approach. There are three main reasons behind Assumption 1:

- The de Simone format is the simplest congruence format for bisimilarity. Hence we must be able to deal with this case before proceeding to any generalisation.
- The specification of parallel composition, left merge and communication merge operators (and of the vast majority of process algebraic operators) is in de Simone format. Hence, that format was a natural choice also for operator f.
- The simplicity of the de Simone rules allows us to reduce considerably the complexity of our case analysis over the sets of available rules for the operator f. However, as witnessed by the developments in this article, even with this simplification, the proof of the desired negative result requires a large amount of delicate, technical work.

Assumptions 2 and 3 still allow us to obtain a significant generalisation of related works, such as [4], as we can see them as an attempt to identify the requirements needed to apply Moller's proof technique to Hennessy's merge like operators. We stress that the reason for adding Assumption 3 is purely technical: it plays a role in the proof of *one* of the claims in our combinatorial analysis of the rules that f may have (see Lemma 11). Although we conjecture that the assumption is not actually necessary to obtain that claim, we were unable to prove it without the assumption.

Even though the vast literature on process algebras offers a plethora of non-finite axiomatisability results for a variety of languages and semantics (see, for instance, the survey [5] from 2005), we are not aware of any previous attempt at proving a result akin to the one we present here. We have already addressed at length how our contribution fits within the study of the equational logic of processes and how it generalises previous results in that field. The proof-theoretic tools and the approach we adopt in proving our main theorem, which links equational logic with structural operational semantics and builds on a number of previous achievements (such as those in [2]), may have independent interest for researchers in logic in computer science. To our mind, achieving an answer to question (P) in full generality would be very pleasing for the concurrency-theory community, as it would finally clarify the canonical role of Bergstra and Klop's auxiliary operators in the finite axiomatisation of parallel composition modulo bisimilarity.

Organisation of contents. After a brief review, in Section 2, of basic notions on process semantics, CCS and equational logic, in Section 3 we present the simplifying assumptions under which we tackle the problem (P). In Section 4 we study the operational semantics of auxiliary operators f meeting our assumptions. In Section 5 we give a detailed presentation of the proof strategy we will follow to address (P). Sections 6–9 are then devoted to the technical development of our negative results. We conclude by discussing future work in Section 10.

Due to space limitations, all proofs have been omitted, and they can be found in the technical report [3].

2 Background

In this section we introduce the basic definitions and results on which the technical developments to follow are based.

Labelled Transition Systems and Bisimilarity. As semantic model we consider classic *labelled transition systems* [16].

▶ **Definition 1.** *A labelled transition system (LTS) is a triple $(S, A, \rightarrow)$, where S is a set of states (or processes), A is a set of actions, and $\rightarrow \subseteq S \times A \times S$ is a (labelled) transition relation.*

As usual, we use $p \xrightarrow{\mu} p'$ in lieu of $(p, \mu, p') \in \rightarrow$. For each $p \in S$ and $\mu \in A$, we write $p \xrightarrow{\mu}$ if $p \xrightarrow{\mu} p'$ holds for some p', and $p \not\xrightarrow{\mu}$ otherwise.

In this paper, we shall consider the states in a labelled transition system modulo bisimilarity [18, 22], allowing us to establish whether two processes have the same behaviour.

▶ **Definition 2.** *Let $(S, A, \rightarrow)$ be a labelled transition system. Bisimilarity, denoted by $\underline{\leftrightarrow}$, is the largest binary symmetric relation over S such that whenever $p \underline{\leftrightarrow} q$ and $p \xrightarrow{\mu} p'$, then there is a transition $q \xrightarrow{\mu} q'$ with $p' \underline{\leftrightarrow} q'$. If $p \underline{\leftrightarrow} q$, then we say that p and q are* bisimilar.

It is well-known that bisimilarity is an equivalence relation (see, e.g., [18, 22]).

The Language CCS$_f$. The language we consider in this paper is obtained by adding a single binary operator f to the recursion, restriction and relabelling free subset of Milner's CCS [18], henceforth referred to as CCS$_f$, and is given by the following grammar:

$$t ::= \mathbf{0} \mid x \mid a.t \mid \bar{a}.t \mid \tau.t \mid t + t \mid t \parallel t \mid f(t,t) \ ,$$

where x is a variable drawn from a countably infinite set $\mathcal{V}$, a is an action, and $\bar{a}$ is its complement. We assume that the actions a and $\bar{a}$ are distinct. Following [18], the action symbol τ will result from the synchronised occurrence of the complementary actions a and $\bar{a}$.

In order to obtain the desired negative results, it will be sufficient to consider the above language with three unary prefixing operators; so there is only one action a with its corresponding complementary action $\bar{a}$. Our results carry over unchanged to a setting with an arbitrary number of actions, and corresponding unary prefixing operators. Henceforth, we let $\mu \in \{a, \bar{a}, \tau\}$ and $\alpha \in \{a, \bar{a}\}$. As usual, we postulate that $\bar{\bar{a}} = a$. We shall use the meta-variables t, u, v, w to range over process terms, and write $\mathit{var}(t)$ for the collection of variables occurring in the term t. The *size* of a term is the number of operator symbols in it. A process term is *closed* if it does not contain any variables. Closed terms, or *processes*, will be typically denoted by p, q, r. Moreover, trailing $\mathbf{0}$'s will often be omitted from terms.

A *(closed) substitution* is a mapping from process variables to (closed) CCS$_f$ terms. For every term t and substitution σ, the term obtained by replacing every occurrence of a variable x in t with the term $\sigma(x)$ will be written $\sigma(t)$. Note that $\sigma(t)$ is closed, if so is σ. We shall sometimes write $\sigma[x \mapsto p]$ to denote the substitution that maps the variable x into process p and behaves like σ on all other variables.

In the remainder of this paper, we exploit the associativity and commutativity of $+$ modulo bisimilarity and we consider process terms modulo them, namely we do not distinguish $t + u$ and $u + t$, nor $(t + u) + v$ and $t + (u + v)$. In what follows, the symbol $=$ will denote equality modulo the above identifications. We use a *summation* $\sum_{i \in \{1,\ldots,k\}} t_i$ to denote the term $t = t_1 + \cdots + t_k$, where the empty sum represents $\mathbf{0}$. We can also assume that the terms t_i, for $i \in \{1, \ldots, k\}$, do not have $+$ as head operator, and refer to them as the *summands* of t.

Henceforth, for each action μ and $m \geq 0$, we let μ^0 denote $\mathbf{0}$ and μ^{m+1} denote $\mu(\mu^m)$. For each action μ and positive integer $i \geq 0$, we also define

$$\mu^{\leq i} = \mu + \mu^2 + \cdots + \mu^i \ .$$

Table 1 The rules of equational logic.

$$(e_1)\ t \approx t \qquad (e_2)\ \frac{t \approx u}{u \approx t} \qquad (e_3)\ \frac{t \approx u \quad u \approx v}{t \approx v} \qquad (e_4)\ \frac{t \approx u}{\sigma(t) \approx \sigma(u)}$$

$$(e_5)\ \frac{t \approx u}{\mu.t \approx \mu.u} \qquad (e_6)\ \frac{t \approx u \quad t' \approx u'}{t + t' \approx u + u'} \qquad (e_7)\ \frac{t \approx u \quad t' \approx u'}{f(t,t') \approx f(u,u')} \qquad (e_8)\ \frac{t \approx u \quad t' \approx u'}{t \parallel t' \approx u \parallel u'}\ .$$

Equational Logic. An *axiom system* $\mathcal{E}$ is a collection of (*process*) *equations* $t \approx u$ over CCS_f. An equation $t \approx u$ is *derivable* from an axiom system $\mathcal{E}$, notation $\mathcal{E} \vdash t \approx u$, if there is an *equational proof* for it from $\mathcal{E}$, namely if $t \approx u$ can be inferred from the axioms in $\mathcal{E}$ using the *rules* of *equational logic*, which are reflexivity, symmetry, transitivity, substitution and closure under CCS_f contexts. In Table 1 we report the rules of equational logic over CCS_f.

Without loss of generality one may assume that substitutions happen first in equational proofs, i.e., that the rule

$$\frac{t \approx u}{\sigma(t) \approx \sigma(u)}$$

may only be used when $(t \approx u) \in \mathcal{E}$. In this case $\sigma(t) \approx \sigma(u)$ is called a *substitution instance* of an axiom in $\mathcal{E}$. Moreover, by postulating that for each axiom in $\mathcal{E}$ also its symmetric counterpart is present in $\mathcal{E}$, one may assume that applications of symmetry happen first in equational proofs, i.e., that the rule

$$\frac{t \approx u}{u \approx t}$$

is never used in equational proofs. In the remainder of the paper, we shall always tacitly assume that equational axiom systems are closed with respect to symmetry.

We are interested in equations that are valid modulo some congruence relation $\mathcal{R}$ over closed terms. The equation $t \approx u$ is said to be *sound* modulo $\mathcal{R}$ if $\sigma(t)\,\mathcal{R}\,\sigma(u)$ for all closed substitutions σ. For simplicity, if $t \approx u$ is sound, then we write $t\,\mathcal{R}\,u$. An axiom system is *sound* modulo $\mathcal{R}$ if, and only if, all of its equations are sound modulo $\mathcal{R}$. Conversely, we say that $\mathcal{E}$ is *ground-complete* modulo $\mathcal{R}$ if $p\,\mathcal{R}\,q$ implies $\mathcal{E} \vdash p \approx q$ for all closed terms p, q. We say that $\mathcal{R}$ has a *finite*, ground-complete, axiomatisation, if there is a *finite* axiom system $\mathcal{E}$ that is sound and ground-complete for $\mathcal{R}$.

3 The simplifying assumptions

The aim of this paper is to investigate whether bisimilarity admits a finite equational axiomatisation over CCS_f, for some binary operator f. Of course, this question only makes sense if f is an operator that preserves bisimilarity. In this section we discuss two assumptions we shall make on the auxiliary operator f in order to meet such requirement and to tackle problem (P) in a simplified technical setting.

3.1 The de Simone format

One way to guarantee that f preserves bisimilarity is to postulate that the behaviour of f is described using Plotkin-style rules that fit a rule format that is known to preserve bisimilarity, see, e.g., [7] for a survey of such rule formats. The simplest format satisfying this criterion is the format proposed by de Simone in [13]. We believe that if we can't deal with operations specified in that format, then there is little hope to generalise our results. Therefore, we make the following

▶ **Assumption 1.** The behaviour of f is described by rules in de Simone format.

▶ **Definition 3.** *An SOS rule ρ for f is in de Simone format if it has the form*

$$\rho = \frac{\{x_i \xrightarrow{\mu_i} y_i \mid i \in I\}}{f(x_1, x_2) \xrightarrow{\mu} t} \tag{1}$$

where $I \subseteq \{1,2\}$, $\mu, \mu_i \in \{a, \bar{a}, \tau\}$ $(i \in I)$, and moreover

- *the variables x_1, x_2 and y_i $(i \in I)$ are all different and are called the variables of the rule,*
- *t is a CCS$_f$ term over variables $\{x_1, x_2, y_i \mid i \in I\}$, called the target of the rule, such that*
  - *each variable occurs at most once in t, and*
  - *if $i \in I$, then x_i does not occur in t.*

Henceforth, we shall assume, without loss of generality, that the variables x_1, x_2, y_1 and y_2 are the only ones used in operational rules. Moreover, if μ is the label of the transition in the conclusion of a de Simone rule ρ, we shall say that ρ has μ as *label*.

The SOS rules for all of the classic CCS operators, reported below, are in de Simone format, and so are those for Hennessy's $/\!\!/$ operator from [14] and for Bergstra and Klop's left and communication merge operators [9], at least if we disregard issues related to the treatment of successful termination. Thus restricting ourselves to operators whose operational behaviour is described by de Simone rules leaves us with a good degree of generality.

$$\frac{}{\mu.x \xrightarrow{\mu} x} \qquad \frac{x \xrightarrow{\mu} x'}{x + y \xrightarrow{\mu} x'} \qquad \frac{y \xrightarrow{\mu} y'}{x + y \xrightarrow{\mu} y'}$$

$$\frac{x \xrightarrow{\mu} x'}{x \parallel y \xrightarrow{\mu} x' \parallel y} \qquad \frac{y \xrightarrow{\mu} y'}{x \parallel y \xrightarrow{\mu} x \parallel y'} \qquad \frac{x \xrightarrow{\alpha} x',\ y \xrightarrow{\bar{\alpha}} y'}{x \parallel y \xrightarrow{\tau} x' \parallel y'}$$

The transition rules for the classic CCS operators above and those for the operator f give rise to transitions between CCS$_f$ terms. The operational semantics for CCS$_f$ is thus given by the LTS whose states are CCS$_f$ terms, and whose transitions are those that are provable using the rules.

In what follows, we shall consider the collection of *closed CCS$_f$ terms* modulo bisimilarity. Since the SOS rules defining the operational semantics of CCS$_f$ are in de Simone's format, we have that bisimilarity is a congruence with respect to CCS$_f$ operators, that is, $\mu p \leftrightarrow \mu q$, $p + p' \leftrightarrow q + q'$, $p \| p' \leftrightarrow q \| q'$ and $f(p, p') \leftrightarrow f(q, q')$ hold whenever $p \leftrightarrow q$, $p' \leftrightarrow q'$ and p, p', q, q' are closed CCS$_f$ terms.

Bisimilarity is extended to arbitrary CCS$_f$ terms thus:

▶ **Definition 4.** *Let t, u be CCS$_f$ terms. We write $t \leftrightarrow u$ if and only if $\sigma(t) \leftrightarrow \sigma(u)$ for every closed substitution σ.*

3.2 Axiomatising $\|$ with f

Our second simplifying assumption concerns how the operator f can be used to axiomatise parallel composition. To this end, a fairly natural assumption on an axiom system over CCS$_f$ is that it includes an equation of the form

$$x \| y \approx t(x, y) \tag{2}$$

where t is a CCS_f term that does not contain occurrences of $\|$ with $var(t) \subseteq \{x, y\}$. More precisely, the term will be in the general form $t(x,y) = \sum_{i \in I} t_i(x,y)$, where I is a finite index set and, for each $i \in I$, $t_i(x,y)$ does not have $+$ as head operator. Equation (2) essentially states that $\|$ is a derived operator in CCS_f modulo bisimilarity. To our mind, this is a natural, initial assumption to make in studying the problem we tackle in the paper.

We now proceed to refine the form of the term $t(x,y)$, in order to guarantee the soundness, modulo bisimilarity, of Equation (2). Intuitively, no term $t_i(x,y)$ can have prefixing as head operator. In fact, if $t(x,y)$ had a summand $\mu.t'(x,y)$, for some $\mu \in \{a, \bar{a}, \tau\}$, then one could easily show that $\mathbf{0}\|\mathbf{0} \not\mathrel{\underline{\leftrightarrow}} t(\mathbf{0},\mathbf{0})$, since $t(\mathbf{0},\mathbf{0})$ could perform a μ-transition, unlike $\mathbf{0}\|\mathbf{0}$. Similarly, $t(x,y)$ cannot have a variable as a summand, for otherwise we would have $a\|\tau \not\mathrel{\underline{\leftrightarrow}} t(a,\tau)$. Indeed, assume, without loss of generality, that $t(x,y)$ has a summand x. Then, $t(a,\tau) \xrightarrow{a} \mathbf{0}$, whereas $a\|\tau$ cannot terminate in one step. We can therefore assume that, for each $i \in I$, $t_i(x,y) = f(t_i^1(x,y), t_i^2(x,y))$ for some CCS_f terms $t_i^j(x,y)$, with $j \in \{1,2\}$. To further narrow down the options on the form that the subterms $t_i^j(x,y)$ might have, we would need to make some assumptions on the behaviour of the operator f. For the sake of generality, we assume that the terms $t_i^j(x,y)$ are in the simplest form, namely they are variables in $\{x,y\}$. Such an assumption is reasonable because to allow prefixing and/or nested occurrences of f-terms in the scope of the terms $t_i(x,y)$ we would need to define (at least partially) the operational semantics of f, thus making our results less general as, roughly speaking, we would need to study one possible auxiliary operator at a time (the one identified by the considered set of de Simone rules). Moreover, if we look at how parallel composition is expressed equationally as a derived operator in terms of Hennessy's merge or Bergstra and Klop's left and communication merge or as in [2], viz. via the equations

$$x \parallel y \approx (x \mathbin{|\!\!/} y) + (y \mathbin{|\!\!/} x)$$

$$x \parallel y \approx (x \mathbin{\|\!\!\_} y) + (y \mathbin{\|\!\!\_} x) + (x \mid y) \qquad x \parallel y \approx (x \mathbin{\|\!\!\_} y) + (x \mathbin{\_\!\!\|} y) + (x \mid y) \ ,$$

we see the emergence of a pattern: the parallel composition operator is always expressed in terms of sums of terms built from the auxiliary operators and variables.

Therefore, from now on we will make the following:

▶ **Assumption 2.** For some $J \subseteq \{x,y\}^2$, the equation

$$x \parallel y \approx \sum \{f(z_1, z_2) \mid (z_1, z_2) \in J\} \tag{3}$$

holds modulo bisimilarity. We shall use t_J to denote the right-hand side of the above equation and use $t_J(p,q)$ to stand for the process $\sigma[x \mapsto p, y \mapsto q](t_J)$, for any closed substitution σ.

Using our assumptions, we further investigate the relation between operator f and parallel composition, obtaining a refined form for Equation (3) (Proposition 7 below).

▶ **Lemma 5.** *Assume that Assumptions 1 and 2 hold. Then:*

1. *The index set J on the right-hand side of (3) is non-empty.*
2. *The set of transition rules for f is non-empty.*
3. *Each transition rule for f has some premise.*
4. *The terms $f(x,x)$ and $f(y,y)$ are not summands of t_J.*

As a consequence, we may infer that the index set J in the term t_J is either one of the singletons $\{(x,y)\}$ or $\{(y,x)\}$, or it is the set $\{(x,y),(y,x)\}$. Due to Moller's results to the effect that bisimilarity has no finite ground-complete axiomatisation over CCS [19, 21], the former option can be discarded, as shown in the following:

▶ **Proposition 6.** *If J is a singleton, then CCS_f admits no finite equational axiomatisation modulo bisimilarity.*

As a consequence, we can restate our Assumption 2 in the following simplified form:

▶ **Proposition 7.** *Equation* (3) *can be refined to the form:*

$$x \parallel y \approx f(x,y) + f(y,x) \quad . \tag{4}$$

Moreover, in the light of Moller's results in [19, 21], we can restrict ourselves to considering only operators f such that $x \parallel y \approx f(x,y)$ does not hold modulo bisimilarity.

For later use, we note a useful consequence of the soundness of Equation (4) modulo bisimilarity.

▶ **Lemma 8.** *Assume that Equation (4) holds modulo $\underline{\leftrightarrow}$. Then $depth(p)$ is finite for each closed CCS_f term p.*

4 The operational semantics of f

In order to obtain the desired results, we shall, first of all, understand what rules f may and must have in order for Equation (4) to hold modulo bisimilarity (Proposition 12 below). We begin this analysis by restricting the possible forms the SOS rules for f *may* take.

▶ **Lemma 9.** *Suppose that f meets Assumption 1, and that Equation* (4) *is sound modulo bisimilarity. Let ρ be a de Simone rule for f with μ as label. Then:*

1. *If $\mu = \tau$ then the set of premises $\{x_i \xrightarrow{\mu_i} y_i \mid i \in I\}$ of ρ can only have one of the following possible forms:*
   - *$\{x_i \xrightarrow{\tau} y_i\}$ for some $i \in \{1, 2\}$, or*
   - *$\{x_1 \xrightarrow{\alpha} y_1, x_2 \xrightarrow{\bar{\alpha}} y_2\}$ for some $\alpha \in \{a, \bar{a}\}$.*
2. *If $\mu = \alpha$ for some $\alpha \in \{a, \bar{a}\}$, then the set of premises $\{x_i \xrightarrow{\mu_i} y_i \mid i \in I\}$ can only have the form $\{x_i \xrightarrow{\alpha} y_i\}$ for some $i \in \{1, 2\}$.*

The previous lemma limits the form of the premises that rules for f *may* have in order for Equation (4) to hold modulo bisimilarity. We now characterise the rules that f *must* have in order for it to satisfy that equation.

Firstly, we deal with *synchronisation*.

▶ **Lemma 10.** *Assume that Equation (4) holds modulo bisimilarity. Then the operator f must have a rule of the form*

$$\frac{x_1 \xrightarrow{\alpha} y_1 \quad x_2 \xrightarrow{\bar{\alpha}} y_2}{f(x_1, x_2) \xrightarrow{\tau} t(y_1, y_2)} \tag{5}$$

for some $\alpha \in \{a, \bar{a}\}$ and term t. Moreover, for each rule for f of the above form the term $t(x,y)$ is bisimilar to $x \parallel y$.

Henceforth we assume, without loss of generality that the target of a rule of the form (5) is $y_1 \| y_2$. We introduce the unary predicates $S^f_{a,\bar{a}}$ and $S^f_{\bar{a},a}$ to identify which rules of type (5) are available for f. In detail, $S^f_{a,\bar{a}}$ holds if f has a rule of type (5) with premises $x_1 \xrightarrow{a} y_1$ and $x_2 \xrightarrow{\bar{a}} y_2$. $S^f_{\bar{a},a}$ holds in the symmetric case.

We consider now the *interleaving* behaviour in the rules for f. In order to properly characterise the rules for f as done in the previous Lemma 10, we consider an additional simplifying assumption on the form that the targets of the rules for f might have.

▶ **Assumption 3.** If t is the target of a rule for f, then t is either a variable or a term obtained by applying a single CCS_f operator to the variables of the rule, according to the constraints of the de Simone format.

▶ **Lemma 11.** *Let $\mu \in \{a, \bar{a}, \tau\}$. Then the operator f must have a rule of the form*

$$\frac{x_1 \xrightarrow{\mu} y_1}{f(x_1, x_2) \xrightarrow{\mu} t(y_1, x_2)} \tag{6}$$

or a rule of the form

$$\frac{x_2 \xrightarrow{\mu} y_2}{f(x_1, x_2) \xrightarrow{\mu} t(x_1, y_2)} \tag{7}$$

for some term t. Moreover, under Assumption 3, for each rule for f of the above forms the term $t(x, y)$ is bisimilar to $x \parallel y$.

Henceforth we assume, without loss of generality, that the target of a rule of the form (6) is $y_1 \| x_2$ and the target of a rule of the form (7) is $x_1 \| y_2$.

For each $\mu \in \{a, \bar{a}, \tau\}$, we introduce two unary predicates, L^f_μ and R^f_μ, that allow us to identify which rules with label μ are available for f. In detail,

- L^f_μ holds if f has a rule of the form (6) with label μ;
- R^f_μ holds if f has a rule of the form (7) with label μ.

We write $L^f_\mu \wedge R^f_\mu$ to denote that f has both a rule of the form (6) and one of the form (7) with label μ. We stress that, for each action μ, the validity of predicate L^f_μ does not prevent R^f_μ from holding, and vice versa. Throughout the paper, in case *only one* of the two predicates holds, we will clearly state it.

Summing up, we have obtained that:

▶ **Proposition 12.** *If f meets Assumptions 1 and 3 and Equation (4) is sound modulo bisimilarity, then f must satisfy $S^f_{\alpha,\bar{\alpha}}$ for at least one $\alpha \in \{a, \bar{a}\}$, and, for each $\mu \in \{a, \bar{a}, \mu\}$, at least one of L^f_μ and R^f_μ.*

The next proposition states that this is enough to obtain the soundness of Equation (4).

▶ **Proposition 13.** *Assume that all of the rules for f have the form (5), (6), or (7). If $S^f_{\alpha,\bar{\alpha}}$ holds for at least one $\alpha \in \{a, \bar{a}\}$, and, for each $\mu \in \{a, \bar{a}, \tau\}$, at least one of L^f_μ and R^f_μ holds, then Equation (4) is sound modulo bisimilarity.*

When the set of actions is $\{a, \bar{a}, \tau\}$, there are 81 operators that satisfy the constraints in Propositions 12 and 13, including parallel composition and Hennessy's merge. In general, when the set of actions has $2n + 1$ elements, there are 3^{3n+1} possible operators meeting those constraints.

5 The main theorem and its proof strategy

Our order of business will now be to use the information collected so far to prove our main result, namely the following theorem:

▶ **Theorem 14.** *Assume that f satisfies Assumptions 1 and 3, and that Equation (4) holds modulo bisimilarity. Then bisimilarity admits no finite equational axiomatisation over CCS_f.*

In this section, we discuss the general reasoning behind the proof of Theorem 14. In light of Propositions 12 and 13, to prove Theorem 14 we will proceed by a case analysis over the possible sets of allowed SOS rules for operator f. In each case, our proof method will follow the same general schema, which has its roots in Moller's arguments to the effect that bisimilarity is not finitely based over CCS (see, e.g., [4, 19, 20, 21]), and that we present here at an informal level.

The main idea is to identify a *witness property of the negative result*. This is a specific property of CCS$_f$ terms, say W_n for $n \geq 0$, that, when n is *large enough*, is preserved by provability from finite axiom systems. Roughly, this means that if $\mathcal{E}$ is a finite set of axioms that are sound modulo bisimilarity, the equation $p \approx q$ is provable from $\mathcal{E}$, and n is greater than the size of all the terms in the equations in $\mathcal{E}$, then either both p and q satisfy W_n, or none of them does. Then, we exhibit an infinite family of valid equations, say e_n, called accordingly *witness family of equations for the negative result*, in which W_n is not preserved, namely it is satisfied only by one side of each equation. Thus, Theorem 14 specialises to:

▶ **Theorem 15.** *Suppose that Assumptions 1–3 are met. Let $\mathcal{E}$ be a finite axiom system over CCS$_f$ that is sound modulo bisimilarity. Then there is an infinite family e_n, $n \geq 0$, of sound equations such that $\mathcal{E}$ does not prove the equation e_n, for each n that is larger than the size of each term in the equations in $\mathcal{E}$.*

In this paper, the property W_n corresponds to having a summand that is bisimilar to a specific process. In detail:

1. We identify, for each case, a family of processes $f(\mu, p_n)$, for $n \geq 0$, and the choices of μ and p_n are tailored to the particular set of SOS rules allowed for f. Moreover, process p_n will have size at least n, for each $n \geq 0$. Sometimes, we shall refer to the processes $f(\mu, p_n)$ as the *witness processes*.
2. We prove that by choosing n *large enough*, given a finite set of valid equations $\mathcal{E}$ and processes $p, q \mathrel{\underline{\leftrightarrow}} f(\mu, p_n)$, if $\mathcal{E} \vdash p \approx q$ and p has a summand bisimilar to $f(\mu, p_n)$, then also q has a summand bisimilar to $f(\mu, p_n)$. Informally, we will choose n greater than the size of all the terms in the equations in $\mathcal{E}$, so that we are guaranteed that the behaviour of the summand bisimilar to $f(\mu, p_n)$ is due to a closed substitution instance of a variable.
3. We provide an infinite family of valid equations e_n in which one side has a summand bisimilar to $f(\mu, p_n)$, but the other side does not. In light of item 2, this implies that such a family of equations cannot be derived from any finite collection of valid equations over CCS$_f$, modulo bisimilarity, thus proving Theorem 15.

To narrow down the combinatorial analysis over the allowed sets of SOS rules for f we examine first the *distributivity properties*, modulo $\underline{\leftrightarrow}$, of the operator f over summation.

First of all, we notice that f cannot distribute over summation in both arguments. This is a consequence of our previous analysis of the operational rules that such an operator f may and must have in order for Equation (4) to hold. However, it can also be shown in a purely algebraic manner.

▶ **Lemma 16.** *A binary operator satisfying Equation (4) cannot distribute over $+$ in both arguments.*

Hence, we can limit ourselves to considering binary operators satisfying our constraints that, modulo bisimilarity, distribute over $+$ in one argument or in none.

We consider these two possibilities in turn.

Distributivity in one argument. Due to our Assumptions 1–3, we can exploit a result from [2] to characterise the rules for an operator f that distributes over summation in one of its arguments. More specifically, [2, Lemma 4.3] gives a condition on the rules for a *smooth operator* g in a GSOS system that includes the $+$ operator in its signature, which guarantees that g distributes over summation in one of its arguments. (The rules defining the semantics of smooth operators are a generalisation of those in de Simone format.) Here we show that, for operator f, the condition in [2, Lemma 4.3] is both necessary and sufficient for distributivity of f in one of its two arguments.

▶ **Lemma 17.** *Let $i \in \{1,2\}$. Modulo bisimilarity, operator f distributes over summation in its i-th argument if and only if each rule for f has a premise $x_i \xrightarrow{\mu_i} y_i$, for some μ_i.*

By Proposition 12, Lemma 17 implies that, when f is distributive in one argument, either L^f_μ holds for all $\mu \in \{a, \bar{a}, \tau\}$ or R^f_μ holds for all $\mu \in \{a, \bar{a}, \tau\}$, and $S_{\alpha,\bar{\alpha}}$ holds for at least one $\alpha \in \{a, \bar{a}\}$. Notice that if L^f_μ holds for each action μ and both $S^f_{a,\bar{a}}$ and $S^f_{\bar{a},a}$ hold, then f behaves as Hennessy's merge $|\!\!/$ [14], and our Theorem 15 specialises to [4, Theorem 22]. Hence we assume, without loss of generality, that $S^f_{\alpha,\bar{\alpha}}$ holds for only one $\alpha \in \{a, \bar{a}\}$. A similar reasoning applies if R^f_μ holds for each action μ.

In Section 6 we will present the proof of Theorem 15 in the case of an operator f that distributes over summation in its first argument (see Theorem 18).

Distributivity in neither argument. We now consider the case in which f does not distribute over summation in either argument.

Also in this case, we can exploit Lemma 17 to obtain a characterisation of the set of rules allowed for an operator f satisfying the desired constraints. In detail, we infer that there must be $\mu, \nu \in \{a, \bar{a}, \tau\}$, not necessarily distinct, such that L^f_μ and R^f_ν hold. Otherwise, as f must have at least one rule for each action (see Proposition 12), at least one argument would be involved in the premises of each rule, and this would entail distributivity over summation in that argument.

We will split the proof of Theorem 15 for an operator f that, modulo bisimilarity, does not distribute over summation in either argument into three main cases:

1. In Section 7, we consider the case of $L^f_\alpha \wedge R^f_\alpha$ holding, for some $\alpha \in \{a, \bar{a}\}$ (Theorem 19).
2. In Section 8, we deal with the case of f having only one rule for α, only one rule for $\bar{\alpha}$, and such rules are of different forms. As we will see, we will need to distinguish two subcases, according to which predicate $S^f_{\alpha,\bar{\alpha}}$ holds (Theorem 20 and Theorem 21).
3. Finally, in Section 9, we study the case of f having only one rule with label α, only one rule with label $\bar{\alpha}$, and such rules are of the same type (Theorem 22).

6 Negative result: the case $L^f_a, L^f_{\bar{a}}, L^f_\tau$

In this section we discuss the nonexistence of a finite axiomatisation of CCS_f in the case of an operator f that, modulo bisimilarity, distributes over summation in one of its arguments. We expand only the case of f distributing in the first argument. (The case of distributivity in the second argument follows by a straightforward adaptation of the arguments we use in this section.) Hence, in the current setting, we can assume the following set of SOS rules for f:

$$\frac{x_1 \xrightarrow{\mu} y_1}{f(x_1,x_2) \xrightarrow{\mu} y_1 \| x_2} \ \forall\, \mu \in \{a, \bar{a}, \tau\} \qquad \frac{x_1 \xrightarrow{\alpha} y_1 \quad x_2 \xrightarrow{\bar{\alpha}} y_2}{f(x_1,x_2) \xrightarrow{\tau} y_1 \| y_2}$$

namely, only L^f_μ holds for each action μ, and only $S_{\alpha,\bar{\alpha}}$ holds for some $\alpha \in \{a, \bar{a}\}$.

According to the proof strategy sketched in Section 5, we now introduce a particular family of equations on which we will build our negative result. We define

$$p_n = \sum_{i=0}^{n} \bar{\alpha}\alpha^{\leq i} \qquad (n \geq 0)$$

$$e_n\colon \quad f(\alpha, p_n) \approx \alpha p_n + \sum_{i=0}^{n} \tau\alpha^{\leq i} \qquad (n \geq 0) \ .$$

It is not difficult to check that the infinite family of equations e_n is sound modulo bisimilarity.

Our order of business is now to prove the instance of Theorem 15 considering the family of equations e_n above, showing that no finite collection of equations over CCS_f that are sound modulo bisimilarity can prove all of the equations e_n $(n \geq 0)$.

Formally, we prove the following theorem:

▶ **Theorem 18.** *Assume an operator f such that only L^f_μ holds for each action μ and only $S^f_{\alpha,\bar{\alpha}}$ holds. Let $\mathcal{E}$ be a finite axiom system over CCS_f that is sound modulo $\leftrightarrow$, n be larger than the size of each term in the equations in $\mathcal{E}$, and p, q be closed terms such that $p, q \leftrightarrow f(\alpha, p_n)$. If $\mathcal{E} \vdash p \approx q$ and p has a summand bisimilar to $f(\alpha, p_n)$, then so does q.*

Then, since the left-hand side of equation e_n, viz. the term $f(\alpha, p_n)$, has a summand bisimilar to $f(\alpha, p_n)$, whilst the right-hand side, viz. the term $\alpha p_n + \sum_{i=0}^{n} \tau\alpha^{\leq i}$, does not, we can conclude that the infinite collection of equations $\{e_n \mid n \geq 0\}$ is the desired witness family. Theorem 15 is then proved for the considered class of auxiliary binary operators.

7 Negative result: the case $L^f_\alpha \wedge R^f_\alpha$

In this section we investigate the first case, out of three, related to an operator f that does not distribute, modulo bisimilarity, over summation in either of its arguments.

We choose $\alpha \in \{a, \bar{a}\}$ and we assume that the set of rules for f includes

$$\frac{x_1 \xrightarrow{\alpha} y_1}{f(x_1, x_2) \xrightarrow{\alpha} y_1 \| x_2} \qquad \frac{x_2 \xrightarrow{\alpha} y_2}{f(x_1, x_2) \xrightarrow{\alpha} x_1 \| y_2} \ ,$$

namely, predicate $L^f_\alpha \wedge R^f_\alpha$ holds for f.

We stress that the validity of the negative result we prove in this section does not depend on which types of rules with labels $\bar{\alpha}$ and τ are available for f. Moreover, the case of an operator for which $L^f_{\bar{\alpha}} \wedge R^f_{\bar{\alpha}}$ holds can be easily obtained from the one we are considering, and it is therefore omitted.

We now introduce the infinite family of valid equations, modulo bisimilarity, that will allow us to obtain the negative result in the case at hand. We define

$$q_n = \sum_{i=0}^{n} \alpha\bar{\alpha}^{\leq i} \qquad (n \geq 0)$$

$$e_n\colon \quad f(\alpha, q_n) \approx \alpha q_n + \sum_{i=0}^{n} \alpha(\alpha \| \bar{\alpha}^{\leq i}) \qquad (n \geq 0) \ .$$

Following the proof strategy from Section 5, we aim to show that, when n is *large enough*, the witness property of having a summand bisimilar to $f(\alpha, q_n)$ is preserved by derivations from a finite, sound axiom system $\mathcal{E}$, as stated in the following theorem:

▶ **Theorem 19.** *Assume an operator f such that $L^f_\alpha \wedge R^f_\alpha$ holds. Let $\mathcal{E}$ be a finite axiom system over CCS_f that is sound modulo $\underline{\leftrightarrow}$, n be larger than the size of each term in the equations in $\mathcal{E}$, and p, q be closed terms such that $p, q \,\underline{\leftrightarrow}\, f(\alpha, q_n)$. If $\mathcal{E} \vdash p \approx q$ and p has a summand bisimilar to $f(\alpha, q_n)$, then so does q.*

Then, we can conclude that the infinite collection of equations $\{e_n \mid n \geq 0\}$ is the desired witness family. In fact, the left-hand side of equation e_n, viz. the term $f(\alpha, q_n)$, has a summand bisimilar to $f(\alpha, q_n)$, whilst the right-hand side, viz. the term $\alpha q_n + \sum_{i=0}^{n} \alpha(\alpha \| \bar{\alpha}^{\leq i})$, does not. This concludes the proof of Theorem 15 in this case.

8 Negative result: the case L^f_α, $R^f_{\bar{\alpha}}$

In this section we deal with the second case related to an operator f that does not distribute over summation in either argument. This time, given $\alpha \in \{a, \bar{a}\}$, we assume that operator f has only one rule with label α and only one rule with label $\bar{\alpha}$, and moreover we assume such rules to be of different types. In detail, we expand the case in which for action α only the predicate L^f_α holds, and for action $\bar{\alpha}$ only $R^f_{\bar{\alpha}}$ holds, namely f has rules:

$$\frac{x_1 \xrightarrow{\alpha} y_1}{f(x_1, x_2) \xrightarrow{\alpha} y_1 \| x_2} \qquad \frac{x_2 \xrightarrow{\bar{\alpha}} y_2}{f(x_1, x_2) \xrightarrow{\bar{\alpha}} x_1 \| y_2} \ .$$

Once again, the proof for the symmetric case with $L^f_{\bar{\alpha}}$ and R^f_α holding is omitted.

To obtain the proof of the negative result, we consider the same family of witness processes $f(\alpha, p_n)$ from Section 6. However, differently from the previous case, the definition of the witness family of equations depends on which rules of type (5) are available for f. More precisely, we need to split the proof of the negative result into two cases, according to whether the rules for f allow α and p_n to synchronise or not.

Case 1: Possibility of synchronisation. Assume first that $S^f_{\alpha,\bar{\alpha}}$ holds, so that the rule

$$\frac{x_1 \xrightarrow{\alpha} y_1 \quad x_2 \xrightarrow{\bar{\alpha}} y_2}{f(x_1, x_2) \xrightarrow{\tau} y_1 \| y_2}$$

allows for synchronisation between α and p_n. In this setting, the infinite family of equations

$$e_n\colon \quad f(\alpha, p_n) \approx \alpha p_n + \sum_{i=0}^{n} \bar{\alpha}(\alpha \| \alpha^{\leq i}) + \sum_{i=0}^{n} \tau \alpha^{\leq i} \qquad (n \geq 0)$$

is sound modulo bisimilarity and it constitutes a family of witness equations.

▶ **Theorem 20.** *Assume an operator f such that only L^f_α holds for α, only $R^f_{\bar{\alpha}}$ holds for $\bar{\alpha}$, and $S^f_{\alpha,\bar{\alpha}}$ holds. Let $\mathcal{E}$ be a finite axiom system over CCS_f that is sound modulo $\underline{\leftrightarrow}$, n be larger than the size of each term in the equations in $\mathcal{E}$, and p, q be closed terms such that $p, q \,\underline{\leftrightarrow}\, f(\alpha, p_n)$. If $\mathcal{E} \vdash p \approx q$ and p has a summand bisimilar to $f(\alpha, p_n)$, then so does q.*

This proves Theorem 15 in the considered setting, as the left-hand side of equation e_n, viz. the term $f(\alpha, p_n)$, has a summand bisimilar to $f(\alpha, p_n)$, whilst the right-hand side, viz. the term $\alpha p_n + \sum_{i=0}^{n} \bar{\alpha}(\alpha \| \bar{\alpha}^{\leq i}) + \sum_{i=0}^{n} \tau \alpha^{\leq i}$, does not.

Case 2: No synchronisation. Assume now that the synchronisation between α and p_n is prevented, namely only $S^f_{\bar{\alpha},\alpha}$ holds. Then, the witness family of equations changes as follows:

$$e_n\colon \quad f(\alpha, p_n) \approx \alpha p_n + \sum_{i=0}^{n} \bar{\alpha}(\alpha \| \alpha^{\leq i}) \qquad (n \geq 0) \ .$$

Our order of business is then to prove the following:

▶ **Theorem 21.** *Assume an operator f such that only L^f_α holds for α, only $R^f_{\bar{\alpha}}$ holds for $\bar{\alpha}$, and only $S^f_{\bar{\alpha},\alpha}$ holds. Let $\mathcal{E}$ be a finite axiom system over CCS_f that is sound modulo $\underline{\leftrightarrow}$, n be larger than the size of each term in the equations in $\mathcal{E}$, and p, q be closed terms such that $p, q \; \underline{\leftrightarrow} \; f(\alpha, p_n)$. If $\mathcal{E} \vdash p \approx q$ and p has a summand bisimilar to $f(\alpha, p_n)$, then so does q.*

Once again, the validity of Theorem 15 follows by noticing that the left-hand side of equation e_n, viz. the term $f(\alpha, p_n)$, has a summand bisimilar to $f(\alpha, p_n)$, whilst the right-hand side, viz. the term $\alpha p_n + \sum_{i=0}^{n} \bar{\alpha}(\alpha \| \bar{\alpha}^{\leq i})$, does not.

9 Negative result: the case L^f_τ

This section considers the last case in our analysis, namely that of an operator f that does not distribute, modulo bisimilarity, over summation in either argument and that has the same rule type for actions $\alpha, \bar{\alpha}$. Here, we present solely the case in which L^f_τ holds, and only $R^f_\alpha, R^f_{\bar{\alpha}}$ hold for $\alpha, \bar{\alpha}$, namely f has rules:

$$\frac{x_1 \xrightarrow{\tau} y_1}{f(x_1, x_2) \xrightarrow{\tau} y_1 \| x_2} \qquad \frac{x_2 \xrightarrow{\alpha} y_2}{f(x_1, x_2) \xrightarrow{\alpha} x_1 \| y_2} \qquad \frac{x_2 \xrightarrow{\bar{\alpha}} y_2}{f(x_1, x_2) \xrightarrow{\bar{\alpha}} x_1 \| y_2} .$$

The symmetric case can be obtained from this one in a straightforward manner.

Interestingly, the validity of the negative result we consider in this section is independent of which rules of type (5) are available for f, and of the validity of the predicate R^f_τ.

Consider the family of equations defined by:

$$e_n\colon \quad f(\tau, q_n) \approx \tau q_n + \sum_{i=0}^{n} \alpha(\tau \| \bar{\alpha}^{\leq i}) \qquad (n \geq 0)$$

where the processes q_n are the same used in Section 7. Theorem 22 below proves that the collection of equations e_n, $n \geq 0$, is a witness family of equations for our negative result.

▶ **Theorem 22.** *Assume an operator f such that L^f_τ holds and only R^f_α and $R^f_{\bar{\alpha}}$ hold for actions α and $\bar{\alpha}$. Let $\mathcal{E}$ be a finite axiom system over CCS_f that is sound modulo $\underline{\leftrightarrow}$, n be larger than the size of each term in the equations in $\mathcal{E}$, and p, q be closed terms such that $p, q \; \underline{\leftrightarrow} \; f(\tau, q_n)$. If $\mathcal{E} \vdash p \approx q$ and p has a summand bisimilar to $f(\tau, q_n)$, then so does q.*

As the left-hand side of equation e_n, viz. the term $f(\tau, q_n)$, has a summand bisimilar to $f(\tau, q_n)$, whilst the right-hand side, viz. the term $\tau q_n + \sum_{i=0}^{n} \alpha(\tau \| \bar{\alpha}^{\leq i})$, does not, we can conclude that the collection of infinitely many equations e_n $(n \geq 0)$ is the desired witness family. This concludes the proof of Theorem 15 for this case and our proof of Theorem 14.

10 Conclusions

In this paper, we have shown that, under a number of reasonable assumptions, we cannot use a single binary auxiliary operator f, whose semantics is defined via inference rules in the de Simone format, to obtain a finite axiomatisation of bisimilarity over the recursion, restriction and relabelling free fragment of CCS. Our result constitutes a first step towards a definitive justification of the canonical standing of the left and communication merge operators by Bergstra and Klop. We envisage the following ways in which we might generalise the contribution presented in this study. Firstly, we will try to get rid of Assumptions 2 and 3. Next, it is natural to relax Assumption 1 by considering the GSOS format [12] in place of the de Simone format. However, as shown by the heavy amount of technical results necessary to prove our main result even in our simplified setting, we believe that this generalisation cannot be obtained in a straightforward manner and that it will require the introduction of new techniques. It would also be very interesting to explore whether some version of problem (P) can be solved using existing results from equational logic and universal algebra.

References

1 Luca Aceto. Some of my favourite results in classic process algebra. *Bulletin of the EATCS*, 81:90–108, 2003.

2 Luca Aceto, Bard Bloom, and Frits W. Vaandrager. Turning SOS rules into equations. *Inf. Comput.*, 111(1):1–52, 1994. doi:10.1006/inco.1994.1040.

3 Luca Aceto, Valentina Castiglioni, Wan Fokkink, Anna Ingólfsdóttir, and Bas Luttik. Are two binary operators necessary to finitely axiomatise parallel composition? *CoRR*, abs/2010.01943, 2020. URL: http://arxiv.org/abs/2010.01943.

4 Luca Aceto, Wan Fokkink, Anna Ingólfsdóttir, and Bas Luttik. CCS with Hennessy's merge has no finite-equational axiomatization. *Theor. Comput. Sci.*, 330(3):377–405, 2005. doi:10.1016/j.tcs.2004.10.003.

5 Luca Aceto, Wan Fokkink, Anna Ingólfsdóttir, and Bas Luttik. Finite equational bases in process algebra: Results and open questions. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 338–367. Springer, 2005. doi:10.1007/11601548_18.

6 Luca Aceto, Wan Fokkink, Anna Ingólfsdóttir, and Bas Luttik. A finite equational base for CCS with left merge and communication merge. *ACM Trans. Comput. Log.*, 10(1):6:1–6:26, 2009. doi:10.1145/1459010.1459016.

7 Luca Aceto, Wan Fokkink, and Chris Verhoef. Structural operational semantics. In *Handbook of Process Algebra*, pages 197–292. North-Holland / Elsevier, 2001. doi:10.1016/b978-044482830-9/50021-7.

8 Luca Aceto, Anna Ingólfsdóttir, Bas Luttik, and Paul van Tilburg. Finite equational bases for fragments of CCS with restriction and relabelling. In *Proceedings of IFIP TCS 2008*, volume 273 of *IFIP*, pages 317–332, 2008. doi:10.1007/978-0-387-09680-3_22.

9 Jan A. Bergstra and Jan Willem Klop. The algebra of recursively defined processes and the algebra of regular processes. In *Proceedings of ICALP 2011*, volume 172 of *Lecture Notes in Computer Science*, pages 82–94, 1984. doi:10.1007/3-540-13345-3_7.

10 Jan A. Bergstra and Jan Willem Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984. doi:10.1016/S0019-9958(84)80025-X.

11 Jan A. Bergstra and Jan Willem Klop. Algebra of communicating processes with abstraction. *Theor. Comput. Sci.*, 37:77–121, 1985. doi:10.1016/0304-3975(85)90088-X.

12 Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995. doi:10.1145/200836.200876.

13 Robert de Simone. Higher-level synchronising devices in meije-SCCS. *Theor. Comput. Sci.*, 37:245–267, 1985. doi:10.1016/0304-3975(85)90093-3.

14 Matthew Hennessy. Axiomatising finite concurrent processes. *SIAM J. Comput.*, 17(5):997–1017, 1988. doi:10.1137/0217063.

15 Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985. doi:10.1145/2455.2460.

16 Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976. doi:10.1145/360248.360251.

17 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. doi:10.1007/3-540-10235-3.

18 Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

19 Faron Moller. *Axioms for Concurrency*. PhD thesis, Department of Computer Science, University of Edinburgh, July 1989. Report CST-59-89. Also published as ECS-LFCS-89-84.

20 Faron Moller. The importance of the left merge operator in process algebras. In *Proceedings of ICALP '90*, volume 443 of *Lecture Notes in Computer Science*, pages 752–764, 1990. doi:10.1007/BFb0032072.

21 Faron Moller. The nonexistence of finite axiomatisations for CCS congruences. In *Proceedings of LICS '90*, pages 142–153, 1990. doi:10.1109/LICS.1990.113741.

22 David M. R. Park. Concurrency and automata on infinite sequences. In *Proceedings of GI-Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, 1981. doi:10.1007/BFb0017309.

23 Gordon D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

24 Walter Taylor. Equational logic. In *Contributions to Universal Algebra*, pages 465–501. North-Holland, 1977. doi:10.1016/B978-0-7204-0725-9.50040-X.

A Quasi-Polynomial Black-Box Algorithm for Fixed Point Evaluation

André Arnold
Independent Researcher, Talence, France
aa-labri@sfr.fr

Damian Niwiński
Institute of Informatics, University of Warsaw, Poland
niwinski@mimuw.edu.pl

Paweł Parys
Institute of Informatics, University of Warsaw, Poland
parys@mimuw.edu.pl

Abstract

We consider nested fixed-point expressions like $\mu z.\nu y.\mu x.f(x,y,z)$ evaluated over a finite lattice, and ask how many queries to a function f are needed to find the value. The previous upper bounds for a monotone function f of arity d over the lattice $\{0,1\}^n$ were of the order $n^{\mathcal{O}(d)}$, whereas a lower bound of $\Omega\left(\frac{n^2}{\lg n}\right)$ is known in case when at least one alternation between the least (μ) and the greatest (ν) fixed point occurs in the expression. Following a recent development for parity games, we show here that a quasi-polynomial number of queries is sufficient, namely $n^{\lg(d/\lg n)+\mathcal{O}(1)}$. The algorithm is an abstract version of several algorithms proposed recently by a number of authors, which involve (implicitly or explicitly) the structure of a universal tree. We then show a quasi-polynomial lower bound for the number of queries used by the algorithms in consideration.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Mu-calculus, Parity games, Quasi-polynomial time, Black-box algorithm

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.9

Funding *Paweł Parys*: Author supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

1 Introduction

Computing fixed points over a finite lattice is a fundamental problem whose algorithmic nature is not yet completely understood. The problem can be stated as evaluation of an expression

$$\theta_d x_d.\theta_{d-1}x_{d-1}.\cdots.\theta_2 x_2.\theta_1 x_1.f\left(x_1,x_2,\dots,x_d\right), \tag{1}$$

where f is a monotonic mapping $f\colon (\{0,1\}^n)^d \to \{0,1\}^n$ for some $n \geq 1$, and where $\theta_1,\dots,\theta_d \in \{\mu,\nu\}$ with μ and ν standing for the least and the greatest fixed point, respectively. If every output bit of the function f is a logical OR or a logical AND of some input bits, the problem is well-known to be equivalent to solving parity games [10, 11] (see, e.g., Arnold and Niwiński [1, Section 4] for an exact statement of this equivalence), and in this form it has attracted much attention since at least 20 years. An abstract formulation was previously considered, e.g., by Browne, Clarke, Jha, Long, and Marrero [4], who made one of the first complexity improvements. These authors also noticed that several algorithms make use only of the structure of fixed points, treating the basic operations as black boxes, and suggested a complexity measure, which in our setting boils down to the following:

▶ **Problem 1.1.** *How many queries to the function f are needed to evaluate Expression (1)?*

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 9; pp. 9:1–9:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The problem has been taken by Parys [24], who showed in particular that at least $\Omega\left(\frac{n^2}{\lg n}\right)$ queries are needed in case when at least one alternation between μ and ν occurs in the expression. One might have expected that this lower bound extends to roughly n^d in general case, but we will see below that it is not the case.

The breakthrough result by Calude, Jain, Khoussainov, Li, and Stephan [5] exhibiting a quasi-polynomial time algorithm for parity games has triggered an intensive flow of research [12, 13, 14, 18, 19, 21, 22, 25], aiming in potential improvement, but also in better understanding of the new complexity situation. The new algorithm has a direct consequence on fixed-point computation: it applies to Expression (1) if the function f is given by a vector of Boolean terms. While these terms can be in general exponential in n and d, Hausmann and Schröder [14] showed that the blow-up can be controlled: Expression (1) can be evaluated in quasi-polynomial time provided that the function f itself can be evaluated in quasi-polynomial time.

In the present paper we focus on the black-box model and evaluate Expression (1) using a quasi-polynomial number of queries to the function f. The algorithm is an abstract version of some recent algorithms for parity games. Our starting point was an algorithm proposed by Parys [25], which is a standard McNaughton–Zielonka's algorithm [23, 29] enhanced by some additional control over recursion. But we also exploit the subsequent improvement by Lehtinen, Schewe, and Wojtczak [22], and a generalisation by Jurdziński and Morvan [19], where the control mechanism uses *universal trees* – a key tool in the modern analysis of parity games [8].

The number of queries to the function f is bounded by $n^{2\cdot\lg(d/\lg n)+\mathcal{O}(1)}$ (in this paper lg stands for the binary logarithm). It can be further improved to $n^{\lg(d/\lg n)+\mathcal{O}(1)}$ (i.e., almost quadratically) by reorganising the algorithm in asymmetric manner, which is a celebrated trick in fixed-point computation invented by Seidl [26]. While the symmetric version of our algorithm resembles the aforementioned recursive algorithms solving parity games [19, 22, 25], the asymmetric version is close to the earlier quasi-polynomial algorithms solving parity games [5, 12, 18, 21]. In consequence, we see a direct link between the two families of algorithms for parity games, which is hardly visible until the μ-calculus perspective is adopted. Finally we show a kind of a lower bound for the class of algorithms in consideration. While this is not an absolute lower bound for the black-box complexity like, e.g., in Parys [24], we show that any algorithm of the considered form must involve a universal tree and therefore, by the result of Czerwiński *et al.* [8], it uses a quasi-polynomial number of queries.

Related work. As mentioned above, we build on recent algorithms for parity games [19, 22, 25]. All these algorithms exploit the concept of *dominion* introduced by Jurdziński, Paterson, and Zwick [20], and our initial step in this paper is a fixed-point interpretation of dominions. The key result on decompositions of dominions is similar to the result on attractor decompositions of dominions considered by Jurdziński and Morvan [19].

The relation of the new techniques for parity games to the μ-calculus has been addressed by Lehtinen [21], who showed, in particular, a new upper bound on the alternation depth of a fixed-point formula. Hausmann and Schröder in the aforementioned work [14] invent a quasi-polynomial time algorithm for computing fixed points of monotone set-valued functions.[1] They did not considered black-box model, but it can be seen that their algorithm (adapted to Expression (1)) performs $n \cdot d^{\lceil \lg n \rceil + 2}$ queries to the function f, which is similar to our

[1] After the submission of our paper, Hausmann and Schröder released a new version of their work [15], where they develop a unified method of fixed-point evaluation based on universal graphs [7].

algorithm in its asymmetric version. These authors ask in the conclusion whether this method can also incorporate the algorithm by Parys [25]. For parity games, this question is essentially answered by a meta-algorithm of Jurdziński and Morvan [19], that captures all algorithms known so far including [22, 25]. In our work (that we started not knowing the work of Hausmann and Schröder [14, 15]) we develop a quasi-polynomial method directly for fixed-point evaluation, and additionally show its limitation.

The concept of *symbolic algorithms* considered for parity games by Chatterjee, Dvořák, Henzinger, and Svozil [6], and also by Jurdziński and Morvan [19], is related to Problem 1.1 if the function f is induced by the binary relation of game moves. Then Expression (1) represents the winning region in a parity game [10] (see also [1]), and any black-box algorithm solving Problem 1.1 can be adapted to a symbolic algorithm for parity games.

The complexity of solving parity games is tantalisingly close to polynomial time. As the problem is in $\mathsf{NP} \cap \mathsf{co}\text{-}\mathsf{NP}$ (even in $\mathsf{UP} \cap \mathsf{co}\text{-}\mathsf{UP}$ [16]), one can hardly expect a lower bound above the P-completeness, which holds already for reachability games [28]. The research in this direction focuses on specific classes of algorithms. The aforementioned (almost) quadratic lower bound by Parys [24] concerns the number of queries used by a black-box algorithm. Recently Czerwiński *et al.* [8] estimated the size of universal trees, which are behind the algorithms exhibiting a *separation scheme* first pinpointed by Bojańczyk and Czerwiński [3]. This gives evidence that the quasi-polynomial complexity of the original algorithm [5] as well as the follow-ups [12, 13, 18, 21] is tight. There has been some hope that the newest approach based on controlling recursion in the McNaughton–Zielonka algorithm [19, 22, 25] may avoid this barrier, but our present results give evidence that it is not the case.

2 Basic concepts

Fixed points. By the celebrated theorem of Knaster and Tarski, if $f\colon L \to L$ is a monotone function over a complete lattice $\langle L, \leq \rangle$, then it has the least (μ) and the greatest (ν) fixed points satisfying the formulae

$$\mu x.f(x) = \inf\{a \mid f(a) \leq a\} \qquad \text{and} \qquad \nu x.f(x) = \sup\{a \mid a \leq f(a)\}\,, \tag{2}$$

respectively. For a monotone function of several arguments, we can apply fixed-point operators successively; for example, $\nu y.\mu x.g(x,y)$ is the greatest fixed point of the mapping $y \mapsto \mu x.g(x,y)$, etc. This gives the semantics of Expression (1).

As it is easy to see that

$$\theta y.\theta x.g(x,y) = \theta x.g(x,x), \tag{3}$$

for $\theta \in \{\mu, \nu\}$, we can without loss of generality assume that the μ and ν operators in Expression (1) alternate.

In this paper, as L we take a finite power $\mathbb{B}^n$ of the Boolean lattice $\mathbb{B} = \{0,1\}$ with the componentwise order denoted by $\leq$. We denote the least and the greatest element of $\mathbb{B}^n$ by $\mathbf{0}$ and $\mathbf{1}$, respectively (assuming that n is clear from the context). We use a semiring notation for join and meet, that is, for $A, B \in \mathbb{B}^n$ we write

$$A + B \overset{\text{def}}{=} \sup(A,B) \qquad \text{and} \qquad A * B \overset{\text{def}}{=} \inf(A,B).$$

Moreover, for $f\colon (\mathbb{B}^n)^d \to \mathbb{B}^n$ and for $A \in \mathbb{B}^n$, let $f^{\upharpoonright A}\colon (\mathbb{B}^n)^{d-1} \to \mathbb{B}^n$ be the mapping defined by

$$f^{\upharpoonright A}(x_1, \dots, x_{d-1}) \overset{\text{def}}{=} f(x_1, \dots, x_{d-1}, A).$$

We refer the reader to Arnold and Niwiński [1] for basic properties of fixed points.

Restrictions. If $A \leq B$, and $f\colon (\mathbb{B}^n)^d \to \mathbb{B}^n$ is monotone in each argument, we let

$$f_{A,B}(x_1, \ldots, x_d) \stackrel{\text{def}}{=} A + B * f(x_1, \ldots, x_d).$$

Clearly $f_{A,B}$ is monotone as well. Note that $f_{\mathbf{0},\mathbf{1}} = f$.

In this paper we often consider a generalisation of Expression (1) where f is replaced by the restricted function $f_{A,B}$. Namely, for $\Theta = \langle \theta_d, \theta_{d-1}, \ldots, \theta_1 \rangle$, with $\theta_i \in \{\mu, \nu\}$, we form an expression $(\Theta, f, (A, B))$ whose value is

$$\|(\Theta, f, (A, B))\| \stackrel{\text{def}}{=} \theta_d x_d.\theta_{d-1} x_{d-1}.\cdots.\theta_1 x_1.f_{A,B}(x_1, \ldots, x_d).$$

The object $(\Theta, f, (A, B))$, where the length of Θ equals the number of arguments in the monotone function $f\colon \mathbb{B}^n \to \mathbb{B}^n$ and where $A \leq B$, is called a *fixed-point expression over* $\mathbb{B}^n$ (or simply a *fixed-point expression* if n is clear from the context). We write $|\Theta|_\mu$ and $|\Theta|_\nu$ for the number of μ and ν operators in Θ, respectively.

▶ Remark 2.1. The above restriction allows us to "narrow the scope" of a function f to an interval $[A, B] = \{x \mid A \leq x \leq B\}$ that can be identified with $\mathbb{B}^I$, where $I = \{i \in \{1, \ldots, n\} \mid A_i < B_i\}$. It would be perhaps more natural to define it with a function

$$A + B * f(A + x_1 * B, A + x_2 * B, \ldots, A + x_n * B);$$

such a function clearly depends only on the bits x_i with $i \in I$. But one can easily prove that $\theta x.A + B * f(A + B * x) = \theta x.A + B * f(x)$, and consequently (by induction) the two definitions lead to the same fixed points. We have chosen $f_{A,B}$ above for its simplicity.

Because $\|(\Theta, f, (A, B))\|$ is a fixed point of $f_{A,B}$, it lies between the bounds A, B:

▶ **Proposition 2.2.** *For every fixed-point expression* $(\Theta, f, (A, B))$,

$$A \leq \|(\Theta, f, (A, B))\| \leq B.$$

Moreover, the value of $(\Theta, f, (A, B))$ does not depend on the bounds A and B, assuming that it lies between these bounds (see Appendix A for a proof):

▶ **Proposition 2.3.** *If* $A \leq C \leq \|(\Theta, f, (A, B))\| \leq D \leq B$, *then*

$$\|(\Theta, f, (A, B))\| = \|(\Theta, f, (C, D))\|.$$

Duality. The *dual* of $b \in \mathbb{B}$ is $\overline{b} = 1 - b$, and the dual of a vector $x = (x_1, \ldots, x_n) \in \mathbb{B}^n$ is $\overline{x} = (\overline{x_1}, \ldots, \overline{x_n})$. The dual of a function $f\colon (\mathbb{B}^n)^d \to \mathbb{B}^n$ is the function $\widetilde{f}$ defined by $\widetilde{f}(x_1, \ldots, x_d) = \overline{f(\overline{x_1}, \ldots, \overline{x_d})}$. The dual $\widetilde{\theta}$ of $\theta \in \{\nu, \mu\}$ is the other element of $\{\nu, \mu\}$, and the dual of $\Theta = \langle \theta_1, \ldots, \theta_d \rangle$ is $\widetilde{\Theta} \stackrel{\text{def}}{=} \langle \widetilde{\theta_1}, \ldots, \widetilde{\theta_d} \rangle$. The dual of $\mathcal{F} = (\Theta, f, (A, B))$ is $\widetilde{\mathcal{F}} \stackrel{\text{def}}{=} (\widetilde{\Theta}, \widetilde{f}, (\overline{B}, \overline{A}))$. The following is a direct consequence of the definition.

▶ **Proposition 2.4.** *For every fixed-point expression* $\mathcal{F}$ *we have* $\|\widetilde{\mathcal{F}}\| = \overline{\|\mathcal{F}\|}$.

This proposition allows us to perform proofs by duality: it is enough to prove statements for one of the fixed-point operators, μ or ν, and then a proof for the other operator follows by considering the dual expression.

Trees. *Ordered trees* (or simply *trees*) are defined by induction: if $T_1, \ldots, T_k$ are ordered trees, then $\langle T_1, \ldots, T_k \rangle$ is an ordered tree (where possibly $k = 0$, which is the base of the induction). A *node*, a *leaf*, a *child*, a *descendant*, a *parent*, an *ancestor*, etc., are defined as expected; we skip formal definitions. The *width* of a tree T, denoted $|T|$, equals 1 for $T = \langle\rangle$, and $|T_1| + \cdots + |T_k|$ for $T = \langle T_1, \ldots, T_k \rangle$ with $k \geq 1$ (we can identify the width with the number of leaves of a tree). The *height* of a tree T equals 0 for $T = \langle\rangle$, and 1 plus the maximum of heights of $T_1, \dots, T_k$ for $T = \langle T_1, \dots, T_k\rangle$ with $k \geq 1$. We allow concatenation of trees, so that $\langle T_1, \dots, T_k\rangle \cdot \langle T_{k+1}, \dots, T_p\rangle$ amounts to $\langle T_1, \dots, T_k, T_{k+1}, \dots, T_p\rangle$, and T^n abbreviates $\underbrace{T \cdot \ldots \cdot T}_{n}$.

A tree is *equitable* if all its branches have the same length; more formally, $T = \langle T_1, \dots, T_k\rangle$ is equitable if all $T_1, \dots, T_k$ are equitable and have the same height. In the sequel, we almost exclusively consider equitable trees. The *level* of a node in an equitable tree of height h is its distance from the leaves (in particular leaves are at level 0, and the root is at level h).

Intuitively, a tree T *embeds* in a tree U if T can be obtained from U by pruning some subtrees. More formally, $T = \langle T_1, \dots, T_k\rangle$ embeds in $U = \langle U_1, \dots, U_p\rangle$ if there exist indices $j_1, \dots, j_k$ such that $1 \leq j_1 < \cdots < j_k \leq p$ and T_i embeds in U_{j_i} for all $i \in \{1, \dots, k\}$. (Thus $\langle\rangle$ embeds in every tree.)

A tree U is (n, h)*-universal* if it is equitable, has height h, and every (equitable) tree T of height at most h and width at most n embeds in U. In this statement it does not matter whether or not we require that T is equitable, because every tree embeds in some equitable tree of the same height and width. We know three families of (n, h)-universal trees:

$$\begin{aligned}
&C_{n,0} = P_{n,0} = S_{n,0} = S_{0,h} = \langle\rangle, && \\
&C_{n,h} = \langle C_{n,h-1}\rangle^n && \text{for } h \geq 1, \\
&P_{n,h} = \langle P_{\lfloor n/2\rfloor,h-1}\rangle^{\lfloor n/2\rfloor} \cdot \langle P_{n,h-1}\rangle \cdot \langle P_{\lfloor n/2\rfloor,h-1}\rangle^{\lfloor n/2\rfloor} && \text{for } h \geq 1, \\
&S_{n,h} = S_{\lfloor n/2\rfloor,h} \cdot \langle S_{n,h-1}\rangle \cdot S_{\lfloor n/2\rfloor,h} && \text{for } n, h \geq 1.
\end{aligned}$$

▶ **Proposition 2.5** ([19, Proposition 3.2])**.** *Trees $C_{n,h}$, $P_{n,h}$, and $S_{n,h}$ are (n, h)-universal.*

Trees $P_{n,h}$ and $S_{n,h}$ are of quasi-polynomial width; more precisely, they have, respectively, $n^{\lg n+\lg(h/\lg n)+O(1)}$ and $n^{\lg(h/\lg n)+O(1)}$ leaves [18, 25].

3 Algorithm

3.1 Symmetric version

In this section we present an algorithm that computes the value of a fixed-point expression $\mathcal{F}$. To this end, we define a value $\|\mathcal{F}\|_{U,V}$ parameterised by two trees U, V. This value can be computed using $|U| \cdot |V|$ queries to f (cf. Lemma 3.2). Simultaneously, if U and V are universal, this value actually equals $\|\mathcal{F}\|$ (cf. Lemma 3.1). Later, we also prove that this is a necessary condition: the above equality holds only when the trees U and V are universal (cf. Theorem 5.1).

Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed-point expression, and let U and V be equitable trees of height, respectively, $|\Theta|_\mu$ and $|\Theta|_\nu$. We define a value $\|\mathcal{F}\|_{U,V}$ by induction on the length of Θ:

(1) if $\Theta = \langle\rangle$, as $\|\mathcal{F}\|_{U,V}$ we take $\|\mathcal{F}\|$, that is, $A + B * f()$;

(2) if $\Theta = \langle\nu\rangle \cdot \Theta'$ and $V = \langle V_1, \dots, V_p\rangle$, then we take $B_0 = B$, and

$$B_j = \|(\Theta', f^{\upharpoonright B_{j-1}}, (A, B_{j-1}))\|_{U,V_j}$$

for $j \in \{1, \dots, p\}$, and $\|\mathcal{F}\|_{U,V} = B_p$;

(3) if $\Theta = \langle\mu\rangle \cdot \Theta'$ and $U = \langle U_1, \dots, U_p\rangle$, then we take $A_0 = A$, and

$$A_j = \|(\Theta', f^{\upharpoonright A_{j-1}}, (A_{j-1}, B))\|_{U_j,V}$$

for $j \in \{1, \dots, p\}$, and $\|\mathcal{F}\|_{U,V} = A_p$.

▶ **Lemma 3.1.** *Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed point expression, and let U and V be $(n, |\Theta|_\mu)$- and $(n, |\Theta|_\nu)$-universal trees, respectively. Then $\|\mathcal{F}\|_{U,V} = \|\mathcal{F}\|$.*

The above lemma is proven in Section 4. We can easily compute $\|\mathcal{F}\|_{U,V}$, following directly its definition:

▶ **Lemma 3.2.** *Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed-point expression, and let U and V be equitable trees of height, respectively, $|\Theta|_\mu$ and $|\Theta|_\nu$. There is an algorithm that computes $\|\mathcal{F}\|_{U,V}$ using $|U| \cdot |V|$ queries to the function f.*

Proof. The proof goes by induction on the length of Θ. A single query is asked in Case (1) of the definition of $\|\mathcal{F}\|_{U,V}$, when all parameters of the original function f have been instantiated, so the claim is true. In Case (2), when $V = \langle V_1, \dots, V_p \rangle$ (where $p \geq 1$, because V has height $|\Theta|_\mu \geq 1$), the number of queries needed to compute $\|\mathcal{F}\|_{U,V}$ amounts to the sum of the analogous numbers for $\|(\Theta', f^{\upharpoonright B_{j-1}}, (A, B_{j-1}))\|_{U,V_j}$, which, by the induction hypothesis, equals

$$|U| \cdot |V_1| + \dots + |U| \cdot |V_p| = |U| \cdot |V|.$$

Case (3) is similar. ◀

▶ Remark 3.3. The above direct "naive" algorithm computing $\|\mathcal{F}\|_{U,V}$ can be made "adaptive": whenever in Case (2) we have that $B_{j-1} = B_{j-2}$ and $V_j = V_{j-1}$, we do not need to compute B_j as $\|(\Theta', f^{\upharpoonright B_{j-1}}, (A, B_{j-1}))\|_{U,V_j}$, but we can simply take $B_j = B_{j-1}$, saving some number of queries to the function f; likewise in Case (3).

Lemma 3.1 combined with Lemma 3.2 and with the estimation of the width of the universal tree $S_{n,h}$ recalled after Proposition 2.5, yields a possible answer to Problem 1.1: $n^{2 \cdot \lg(d/\lg n)+O(1)}$ queries to the function f are enough to evaluate Expression (1). In the next section we improve this complexity (almost) quadratically.

The above algorithm can be seen as a translation of the generic recursive algorithm of Jurdziński and Morvan [19] solving parity games (which is also parameterised by two trees assumed to be universal) to the setting of μ-calculus. There is one difference: the algorithm for parity games includes computation of attractors, which is absent here. The above algorithm used with trees $S_{n,h}$ resembles the recursive algorithm of Lehtinen, Schewe, and Wojtczak [22], and the adaptive version of the algorithm (cf. Remark 3.3) used with trees $P_{n,h}$ resembles the recursive algorithm of Parys [25]. The adaptive version of the algorithm used with the complete trees $C_{n,h}$ gives a version of the naive-iteration algorithm that works in polynomial time assuming that n is fixed (cf. Parys [24]).

3.2 Asymmetric version

We now modify the algorithm from the previous section using an idea of Seidl [26]. As a first step, we define yet another value, $\|\mathcal{F}\|_V$, parameterised by a single tree V. In its definition, we proceed in an asymmetric way: on the μ side we simply compute all fixed points, and on the ν side we follow a structure of the tree V. It turns out that $\|\mathcal{F}\|_V$ equals $\|\mathcal{F}\|$ if V is universal (cf. Lemma 3.4 below). There is however no direct analogue to Lemma 3.2, and keeping the number of queries low requires some care. We explore the fact that nested applications of fixed points of the same kind (μ in this case) can be reduced to a single application over a vector of variables in a system of equations, which is precisely the idea behind Seidl's algorithm [26]. Thus, as a second step of our algorithm, we replace our recursive definition of $\|\mathcal{F}\|_V$ by an equivalent system of least fixed-point equations. This system has size proportional to $|V|$, and thus can be solved using such a number of queries to the function f (see Lemma 3.6 below), yielding the value $\|\mathcal{F}\|$.

Algorithm 1

```
procedure GENERATE($x, \Theta, \mathbf{f}, \mathbf{B}, V$)
begin
    if $\Theta = \langle\rangle$ then
        output "$x = \mathbf{B} * \mathbf{f}$";
    if $\Theta = \langle\nu\rangle \cdot \Theta'$ and $V = \langle V_1, \dots, V_p\rangle$ then begin
        $\mathbf{B}_0 = \mathbf{B}$;
        $\mathbf{B}_p = x$;
        for $j = 1$ to $p - 1$ do
            $\mathbf{B}_j$ = FRESHVARIABLE();
        for $j = 1$ to $p$ do
            GENERATE($\mathbf{B}_j, \Theta', \mathbf{f}^{\Rsh \mathbf{B}_{j-1}}, \mathbf{B}_{j-1}, V_j$);
    end;
    if $\Theta = \langle\mu\rangle \cdot \Theta'$ then
        GENERATE($x, \Theta', \mathbf{f}^{\Rsh x}, \mathbf{B}, V$);
end;
$x_{res}$ = FRESHVARIABLE();
GENERATE($x_{res}, \Theta, \mathsf{f}(?, \dots, ?), \mathsf{x}_0, V$);
```

Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed-point expression, and let V be an equitable tree of height $|\Theta|_\nu$. We define $\|\mathcal{F}\|_V$ by induction on the length of Θ:

(1) if $\Theta = \langle\rangle$, as $\|\mathcal{F}\|_V$ we take $\|\mathcal{F}\|$, that is, $A + B * f()$;

(2) if $\Theta = \langle\nu\rangle \cdot \Theta'$ and $V = \langle V_1, \dots, V_p\rangle$, then we take $B_0 = B$, and

$$B_j = \|(\Theta', f^{\Rsh B_{j-1}}, (A, B_{j-1}))\|_{V_j}$$

for $j \in \{1, \dots, p\}$, and $\|\mathcal{F}\|_V = B_p$;

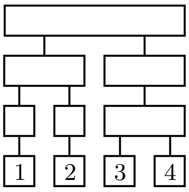
(3) if $\Theta = \langle\mu\rangle \cdot \Theta'$, then we take $\|\mathcal{F}\|_V = \mu x.\|(\Theta', f^{\Rsh x}, (A, B))\|_V$ (i.e., the least fixed point of the mapping $x \mapsto \|(\Theta', f^{\Rsh x}, (A, B))\|_V$).

When we iterate a function n times, starting from the least element (as in the definition of $\|\mathcal{F}\|_{C_{n,h},V}$), we reach the least fixed point (appearing in the definition of $\|\mathcal{F}\|_V$). It is thus not difficult to prove the following lemma, saying that instead of computing $\|\mathcal{F}\|$ we can compute $\|\mathcal{F}\|_V$ for some universal tree V (see Appendix B for more details).

▶ **Lemma 3.4.** *Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed-point expression, and let V be an equitable tree of height $|\Theta|_\nu$. Then $\|\mathcal{F}\|_V = \|\mathcal{F}\|_{C_{n,|\Theta|_\mu},V}$. In particular, if V is $(n, |\Theta|_\nu)$-universal, then (by Proposition 2.5 and Lemma 3.1) $\|\mathcal{F}\|_V = \|\mathcal{F}\|$.*

Next, we consider a system of equations corresponding to the definition of $\|\mathcal{F}\|_V$. For simplicity, we assume here that $A = \mathbf{0}$ and $B = \mathbf{1}$, that is, we consider $\mathcal{F} = (\Theta, f, (\mathbf{0}, \mathbf{1}))$. Let $\mathcal{V}$ be a set of variables, and let $\mathcal{V}_\mathbf{1} = \mathcal{V} \uplus \{\mathsf{x}_0\}$ contain additionally a variable (constant) x_0 that is always valuated to the element $\mathbf{1}$ of $\mathbb{B}^n$. Our equations will be of the form $x = y_0 * \mathsf{f}(y_1, \dots, y_d)$, where $x \in \mathcal{V}$, $y_0, \dots, y_d \in \mathcal{V}_\mathbf{1}$, and f is a constant denoting the considered function.

The system of equations is generated by Algorithm 1. In this algorithm, $\mathbf{f}$ is an expression of the form $\mathsf{f}(?, \dots, ?, y_{k+1}, \dots, y_d)$ for some variables $y_{k+1}, \dots, y_d \in \mathcal{V}_\mathbf{1}$. Moreover, $\mathbf{f}^{\Rsh z}$ for $z \in \mathcal{V}_\mathbf{1}$ denotes $\mathsf{f}(?, \dots, ?, z, y_{k+1}, \dots, y_d)$ (we substitute z for the last question mark). The parameter $\mathbf{B}$ of the procedure GENERATE is an element of $\mathcal{V}_\mathbf{1}$. Note that the algorithm is



$$\begin{aligned}
\mathsf{x}_1 &= \mathsf{x}_0 * \mathsf{f}(\mathsf{x}_1, \mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_0, \mathsf{x}_2, \mathsf{x}_0, \mathsf{x}_4),\\
\mathsf{x}_2 &= \mathsf{x}_1 * \mathsf{f}(\mathsf{x}_2, \mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_0, \mathsf{x}_4),\\
\mathsf{x}_3 &= \mathsf{x}_2 * \mathsf{f}(\mathsf{x}_3, \mathsf{x}_2, \mathsf{x}_4, \mathsf{x}_2, \mathsf{x}_4, \mathsf{x}_2, \mathsf{x}_4),\\
\mathsf{x}_4 &= \mathsf{x}_3 * \mathsf{f}(\mathsf{x}_4, \mathsf{x}_3, \mathsf{x}_4, \mathsf{x}_2, \mathsf{x}_4, \mathsf{x}_2, \mathsf{x}_4).
\end{aligned}$$

Figure 1 A tree V (left), and the system of equations generated for $\|(\langle\mu\rangle \cdot \langle\nu,\mu\rangle^3, f, (\mathbf{0},\mathbf{1}))\|_V$ (right). Variable x_4 stores the result. Observe a correspondence between variables and leaves of V.

syntactical: it depends on the tree V and on the sequence Θ, but not on any particular interpretation of f. Notice that when the algorithm enters line 4 (i.e., it is going to output an equation), $\mathbf{f}$ contains no ?'s. See also Appendix C for another definition of the system.

For an example of a generated system of equations, see Figure 1.

Now, given $\mathcal{F} = (\Theta, f, (A, B))$ and V like in the definition of $\|\mathcal{F}\|_V$, where additionally $A = \mathbf{0}$ and $B = \mathbf{1}$, we can interpret and solve the resulting system in the lattice $\mathbb{B}^n$ with f interpreted by the function f. Thanks to a direct correspondence between the system and the recursive definition of $\|\mathcal{F}\|_V$, we obtain the following lemma (see Appendix D for a tedious but straightforward proof).

▶ **Lemma 3.5.** *Let $\mathcal{F} = (\Theta, f, (\mathbf{0},\mathbf{1}))$ be a fixed-point expression, and let V be an equitable tree of height $|\Theta|_\nu$. The value assigned to the variable x_{res} in the least solution of the system of equations generated by Algorithm 1 equals $\|\mathcal{F}\|_V$.*

We are now ready to state an analogue to Lemma 3.2.

▶ **Lemma 3.6.** *Let $\mathcal{F} = (\Theta, f, (\mathbf{0},\mathbf{1}))$ be a fixed-point expression, and let V be an equitable tree of height $|\Theta|_\nu$. There is an algorithm that computes $\|\mathcal{F}\|_V$ using between $|V|$ and $|V| \cdot (1+nd)$ queries to the function f.*

Proof. By Lemma 3.5, it is enough to generate a system of equations using Algorithm 1, and then to find the least solution of this system, having $|V|$ equations. The least solution can be found by a standard worklist algorithm (cf. [26, Proposition 2]). In this algorithm, we keep updating an underapproximation of the least solution, stage by stage, until reaching a fixed point. In every stage, we re-evaluate right sides only of those equations in which at least one variable was modified in the previous stage, and we store results in variables appearing on the left side (before the first stage we set x_0 to $\mathbf{1}$, other variables to $\mathbf{0}$, and we treat all of them as modified). Each among d arguments of $|V|$ equations can be updated (increased) at most n times, yielding at most $|V| \cdot (1 + nd)$ (and at least $|V|$) queries to the function f. ◀

For $V = S_{n,|\Theta|_\nu}$, the number of queries becomes $n^{\lg(d/\lg n)+O(1)}$, so (almost) quadratically better than for the algorithm of Section 3.1, and, to our knowledge, the best so far.

▶ **Corollary 3.7.** *There is an algorithm to evaluate the Expression (1) using at most $n^{\lg(d/\lg n)+O(1)}$ queries.*

The reduction from an expression using nested μ and ν fixed-point operators to a system of equations involving only the least fixed point can be compared to a reduction from parity games to reachability (or safety) games. As explained in Czerwiński *et al.* [8], such a reduction stands behind the "iterative" quasi-polynomial algorithms [5, 12, 18, 21], but (for universal trees of exponential width) is present also in earlier results [2, 17].

3.3 Time and space complexity

Although our main interest is in the number of queries to the function f performed by our algorithms, let us also analyse their time and space complexity. Let t_f be the time needed to answer a single query (i.e., to compute the value of f for given arguments), and, as previously, let n and d denote, respectively, the height of the considered lattice and the arity of f.

The symmetric version of our algorithm spends time $t_f \cdot n^{2\cdot\lg(d/\lg n)+O(1)}$ on computing values of the function f, and beside of that performs some recursive calls following the structure of universal trees $S_{n,h}$ (of course there is no need to actually construct these trees). Formally, the number of recursive calls depends on the number of nodes of the two trees, not on the number of leaves. However, in every tree, the number of nodes with at least two children is smaller than the number of leaves, and one can easily improve the algorithm so that it "skips" nodes with a single child. Thus, the time spent on performing recursive calls can be ignored. The memory usage is $O(n \cdot d)$ (the depth of the recursion is d, and on every level it is enough to store a constant number of elements of $\mathbb{B}^n$).

In the asymmetric version, the system of equations can be solved (using the method described in the proof of Lemma 3.6) in time proportional to the number of queries. Moreover, we do not need to actually generate the system; we can instead describe it explicitly based on the considered tree (see Appendix C for details). Such a description allows to navigate in the system (e.g. to find all equations containing a given variable) with a constant overhead. Thus, due to Corollary 3.7, the running time is $t_f \cdot n^{\lg(d/\lg n)+O(1)}$. To compare, the algorithm of Hausmann and Schröder [14] needs a factor $O(n \cdot d^{\lceil \lg n\rceil+2})$ per query, not $O(1)$.

Concerning the memory usage, in a straightforward implementation we store a value for every variable while solving the system of equations. We can do better, however. Indeed, we can observe that if we write the i-th equation as

$$\mathsf{x}_i = \mathsf{x}_{k(i,0)} * \mathsf{f}(\mathsf{x}_{k(i,1)}, \ldots, \mathsf{x}_{k(i,d)}),$$

then for all $j \in \{0, \ldots, d\}$ we have $k(i,j) \leq k(i',j)$ whenever $i \leq i'$ (this is best visible while looking at the explicit description of the system, introduced in Appendix C). Thanks to this monotonicity of the system, and monotonicity of f, values assigned to the variables after every stage of the algorithm satisfy $\mathsf{x}_i \geq \mathsf{x}_{i'}$ whenever $i \leq i'$ (this is satisfied before the first stage, and is then preserved). Such a valuation can be stored very succinctly, using n numbers: for every bit of $\mathbb{B}^n$ it is enough to remember the number of the last variable in which this bit is set to 1. Going further: in order to remember which variables were modified in the last stage, it is enough to remember two last valuations of the variables. It is not difficult to adapt the algorithm to such a representation of valuations. Thus, the memory usage can be reduced to $O(n)$, modulo a polylogarithmic factor.

4 Correctness of the algorithms

In this section we prove that if U and V are universal trees, then $\|\mathcal{F}\| = \|\mathcal{F}\|_{U,V}$ (Lemma 3.1). As a first step, we introduce *sup-dominions* and their *decompositions*, and we prove some properties of these notions. Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed-point expression. A value $D \in \mathbb{B}^n$ such that $A \leq D \leq B$ is a *sup-dominion* for $\mathcal{F}$ if $D = \|(\Theta, f, (A, D))\|$.

For readers familiar with game-theoretic dominions considered in prior work [19, 20, 22, 25], the following analogy may be useful: a sup-dominion for $(\Theta, f, (A, B))$ is an area D where player Even can force the play either to reach A or to ensure the parity condition while not leaving D. One can also define inf-dominions (by requiring $D = \|(\Theta, f, (D, B))\|$), describing the complement of an analogous area for player Odd; however, we conduct our correctness proof using only sup-dominions, and then arguing by duality.

We first note some useful properties analogous to the properties of game dominions noted in prior work [19, 22, 25], stemming from Proposition 2.3.

▶ **Lemma 4.1.** *For every fixed-point expression $\mathcal{F}$, the value $\|\mathcal{F}\|$ is a sup-dominion for $\mathcal{F}$.*

▶ **Lemma 4.2.** *If D is a sup-dominion for a fixed-point expression $(\Theta, f, (A, B))$, and $A \leq A' \leq D$, then D is also a sup-dominion for $(\Theta, f, (A', B))$.*

Lemmata 4.1 and 4.2 are immediate consequences of definitions and of Proposition 2.3.

▶ **Lemma 4.3.** *Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed-point expression with $\Theta = \langle\theta\rangle \cdot \Theta'$. If D is a sup-dominion for $\mathcal{F}$, then D is also a sup-dominion for $(\Theta', f^{\Rsh D}, (A, D))$.*

Proof. We have by definition $D = \|(\Theta, f, (A, D))\| = \theta x.\|(\Theta', f^{\Rsh x}, (A, D))\|$. This means that D is a fixed point of the mapping $x \mapsto \|(\Theta', f^{\Rsh x}, (A, D))\|$, that is, $D = \|(\Theta', f^{\Rsh D}, (A, D))\|$, as required. ◀

▶ **Lemma 4.4.** *Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed-point expression with $\Theta = \langle\mu\rangle \cdot \Theta'$. For every sup-dominion D for $\mathcal{F}$ such that $A < D$ there exists a sup-dominion D' for $(\Theta', f^{\Rsh A}, (A, B))$ such that $A < D' \leq D$.*

Proof. Let $g(x) = \|(\Theta', f^{\Rsh x}, (A, D))\|$ so that $D = \|(\Theta, f, (A, D))\| = \mu x.g(x)$. We have, by monotonicity, $D = g(D) \geq g(A) = \|(\Theta', f^{\Rsh A}, (A, D))\|$. Hence, from Proposition 2.3 we obtain $g(A) = \|(\Theta', f^{\Rsh A}, (A, g(A)))\|$. That is, $g(A)$ is a sup-dominion for $(\Theta', f^{\Rsh A}, (A, B))$, which moreover satisfies $A \leq g(A) \leq D$ (where $A \leq g(A)$ is by Proposition 2.2). If $g(A) = A$, we would have $D = \mu x.g(x) \leq A$, a contradiction. Hence $g(A) > A$, and $D' = g(A)$ satisfies the claim. ◀

Next, we define the crucial concept of the paper: dominion decompositions. Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed-point expression. The definition is by induction on the length of Θ. A pair $(D, \mathcal{H})$ is a *sup-dominion decomposition* for $\mathcal{F}$ if D is a sup-dominion for $\mathcal{F}$ and

- if $\Theta = \langle\rangle$, then $\mathcal{H} = \langle\rangle$;
- if $\Theta = \langle\nu\rangle \cdot \Theta'$, then $(D, \mathcal{H})$ is a sup-dominion decomposition for $(\Theta', f^{\Rsh D}, (A, D))$;
- if $\Theta = \langle\mu\rangle \cdot \Theta'$, then $\mathcal{H} = \langle(D_1, \mathcal{H}_1), \dots, (D_k, \mathcal{H}_k)\rangle$, where, assuming $D_0 = A$, for every $i \in \{1, \dots, k\}$ the pair $(D_i, \mathcal{H}_i)$ is a sup-dominion decomposition for $(\Theta', f^{\Rsh D_{i-1}}, (D_{i-1}, B))$, and $D_k = D$.

To a sup-dominion decomposition $(D, \mathcal{H})$ we can assign a tree $T_{\mathcal{H}}$ by forgetting the dominions stored in the decomposition and taking only its "shape": for $\mathcal{H} = \langle(D_1, \mathcal{H}_1), \dots, (D_k, \mathcal{H}_k)\rangle$ we let $T_{\mathcal{H}} = \langle T_{\mathcal{H}_1}, \dots, T_{\mathcal{H}_k}\rangle$.

The following crucial lemma states that every dominion has a decomposition (not necessarily unique). Here by $|D - A|_1$ (for $D \geq A$) we denote the number of bits that are 1 in D but 0 in A.

▶ **Lemma 4.5.** *Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed-point expression. For every sup-dominion D for $\mathcal{F}$ such that $A < D$ there exists a sup-dominion decomposition $(D, \mathcal{H})$ for $\mathcal{F}$ such that $T_{\mathcal{H}}$ is an equitable tree of height $|\Theta|_\mu$ and of width at most $|D - A|_1$.*

Proof. The proof is by induction on the length of Θ. For $\Theta = \langle\rangle$ we just take $\mathcal{H} = \langle\rangle$ (notice that $T_{\mathcal{H}}$ has width $1 \leq |D - A|_1$, because $A < D$). For $\Theta = \langle\nu\rangle \cdot \Theta'$, by Lemma 4.3 D is also a sup-dominion for $(\Theta', f^{\Rsh D}, (A, D))$, and thus the sup-dominion decomposition exists by the induction hypothesis.

Finally, suppose that $\Theta = \langle\mu\rangle \cdot \Theta'$. We first construct a sequence $A = D_0 < D_1 < \dots < D_k \leq D$ such that D_i is a sup-dominion for $(\Theta', f^{\Rsh D_{i-1}}, (D_{i-1}, B))$, for every $i \in \{1, \dots, k\}$. We start with $k = 0$ (and $D_0 = A$). Then, as long as $D_k < D$, we create D_{k+1} as follows. First, because D is a sup-dominion for $\mathcal{F}$ and because $A \leq D_k$, by Lemma 4.2 we have that D is a sup-dominion for $(\Theta, f, (D_k, B))$. Second, by Lemma 4.4 (applied to D and $(\Theta, f,$ $(D_k, B)))$ there exists sup-dominion D_{k+1} for $(\Theta', f^{\Rsh D_k}, (D_k, B))$ such that $D_k < D_{k+1} \leq D$. We can thus attach this D_{k+1} to the sequence. We end the construction when $D_k = D$. To finish the proof, we use the induction hypothesis, which says that every D_i can be extended to a sup-dominion decomposition $(D_i, \mathcal{H}_i)$ for $(\Theta', f^{\Rsh D_{i-1}}, (D_{i-1}, B))$ such that $T_{\mathcal{H}_i}$ is an equitable tree of height $|\Theta|_\mu - 1$ and of width at most $|D_i - D_{i-1}|_1$. Then $\mathcal{H} = \langle (D_1, \mathcal{H}_1), \dots, (D_k, \mathcal{H}_k) \rangle$ satisfies the thesis. ◀

Coming slowly to the proof of Lemma 3.1, let us state some basic properties of the value $\|\mathcal{F}\|_{U,V}$. The first two lemmata can be shown by a straightforward induction on the length of Θ:

▶ **Lemma 4.6.** *For every fixed-point expression $(\Theta, f, (A, B))$ and for all equitable trees U, V of height, respectively, $|\Theta|_\mu$ and $|\Theta|_\nu$,*

$$A \leq \|(\Theta, f, (A, B))\|_{U,V} \leq B.$$

▶ **Lemma 4.7.** *For every fixed-point expression $\mathcal{F} = (\Theta, f, (A, B))$ and for all equitable trees U, V of height, respectively, $|\Theta|_\mu$ and $|\Theta|_\nu$, we have $\|\widetilde{\mathcal{F}}\|_{V,U} = \overline{\|\mathcal{F}\|_{U,V}}$.*

Moreover, again by a straightforward induction, it follows that $\|(\Theta, f, (A, B))\|_{U,V}$ is monotone in f, A, and B. This value is also monotone in U and V, in the following sense:

▶ **Lemma 4.8.** *Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed-point expression, let T, U be equitable trees of height $|\Theta|_\mu$, and let T', V be equitable trees of height $|\Theta|_\nu$. If T embeds in U, and T' embeds in V, then $\|\mathcal{F}\|_{T,V} \leq \|\mathcal{F}\|_{U,V} \leq \|\mathcal{F}\|_{U,T'}$.*

Proof. It is enough to prove the inequality $\|\mathcal{F}\|_{T,V} \leq \|\mathcal{F}\|_{U,V}$; the second inequality follows then by duality (using Lemma 4.7). The proof is by induction on the length of Θ. For empty Θ both values, $\|\mathcal{F}\|_{T,V}$ and $\|\mathcal{F}\|_{U,V}$ are defined as $A + B * f()$; we have equality. For $\Theta = \langle\nu\rangle \cdot \Theta'$, the inductive definitions of $\|\mathcal{F}\|_{T,V}$ and $\|\mathcal{F}\|_{U,V}$ are the same (we descend recursively using the tree V), so it is enough to use the induction hypothesis and monotonicity. Suppose that $\Theta = \langle\mu\rangle \cdot \Theta'$, and $T = \langle T_1, \dots, T_k \rangle$, and $U = \langle U_1, \dots, U_p \rangle$. Then $\|\mathcal{F}\|_{T,V}$ and $\|\mathcal{F}\|_{U,V}$ are defined as A'_k and A_p, respectively, where

$$\begin{aligned}
&A'_0 = A_0 = A, && \\
&A'_i = \|(\Theta', f^{\Rsh A'_{i-1}}, (A'_{i-1}, B))\|_{T_i,V} && \text{for } i \in \{1, \dots, k\}, \text{ and} \\
&A_j = \|(\Theta', f^{\Rsh A_{j-1}}, (A_{j-1}, B))\|_{U_j,V} && \text{for } j \in \{1, \dots, p\}.
\end{aligned}$$

Observe that if T_i embeds in U_j, and $A'_{i-1} \leq A_{j-1}$, then by monotonicity and by the induction hypothesis we obtain that $A'_i \leq A_j$. Moreover, always $A_{j-1} \leq A_j$, by Lemma 4.6 (this way, we can skip subtrees U_j to which no T_i needs to be embedded). Using these inequalities and the definition of embedding we easily obtain the required thesis $A'_k \leq A_p$. ◀

The next lemma is crucial in the proof of Lemma 3.1.

▶ **Lemma 4.9.** *Let $\mathcal{F} = (\Theta, f, (A, B))$ be a fixed-point expression, and let V be an equitable tree of height $|\Theta|_\nu$. If $(D, \mathcal{H})$ is a sup-dominion decomposition for $\mathcal{F}$, then $D \leq \|\mathcal{F}\|_{T_{\mathcal{H}},V}$.*

Proof. The proof is by induction on the length of Θ. If $\Theta = \langle\rangle$, we have (by the definition of a sup-dominion) $D = \|(\Theta, f, (A, D))\| = A + D * f() \leq A + B * f() = \|\mathcal{F}\|_{T_\mathcal{H},V}$.

Suppose that $\Theta = \langle\nu\rangle \cdot \Theta'$ and $V = \langle V_1, \dots, V_p\rangle$. Recall the values B_j from the definition of $\|\mathcal{F}\|_{T_\mathcal{H},V}$ (page 5). We prove by an internal induction on $j \in \{0, \dots, p\}$ that $D \leq B_j$ (this gives the thesis since $\|\mathcal{F}\|_{T_\mathcal{H},V} = B_p$). For $j = 0$ the thesis is clear: $D \leq B = B_0$. Next, for $j > 0$ we prove $D \leq B_j$ assuming $D \leq B_{j-1}$. Recall that $(D, \mathcal{H})$ is a sup-dominion decomposition for $(\Theta', f^{\restriction D}, (A, D))$. Thus, the induction hypothesis implies that $D \leq \|(\Theta', f^{\restriction D}, (A, D))\|_{T_\mathcal{H},V_j}$. Using the assumption $D \leq B_{j-1}$ and monotonicity, we obtain that $D \leq \|(\Theta', f^{\restriction B_{j-1}}, (A, B_{j-1}))\|_{T_\mathcal{H},V_j} = B_j$, as required.

Finally, suppose that $\Theta = \langle\mu\rangle \cdot \Theta'$ and $\mathcal{H} = \langle(D_1, \mathcal{H}_1), \dots, (D_k, \mathcal{H}_k)\rangle$. Recall the values A_j from the definition of $\|\mathcal{F}\|_{T_\mathcal{H},V}$. We prove by an internal induction on $j \in \{0, \dots, k\}$ that $D_j \leq A_j$ (this gives the thesis since $\|\mathcal{F}\|_{T_\mathcal{H},V} = A_k$). For $j = 0$ the thesis is clear: $D_0 = A = A_0$. Next, for $j > 0$ we prove $D_j \leq A_j$ assuming $D_{j-1} \leq A_{j-1}$. Recall that $(D_j, \mathcal{H}_j)$ is a sup-dominion decomposition for $(\Theta', f^{\restriction D_{j-1}}, (D_{j-1}, B))$. Thus, the induction hypothesis implies that $D_j \leq \|(\Theta', f^{\restriction D_{j-1}}, (D_{j-1}, B))\|_{T_{\mathcal{H}_j},V}$. Using the assumption $D_{j-1} \leq A_{j-1}$ and monotonicity, we obtain that $D_j \leq \|(\Theta', f^{\restriction A_{j-1}}, (A_{j-1}, B))\|_{T_{\mathcal{H}_j},V} = A_j$, as required. ◀

Proof of Lemma 3.1. Recall that we prove $\|\mathcal{F}\|_{U,V} = \|\mathcal{F}\|$, where $\mathcal{F} = (\Theta, f, (A, B))$ and U, V are universal. It is actually enough to prove $\|\mathcal{F}\| \leq \|\mathcal{F}\|_{U,V}$, because then the converse inequality follows by duality (using Proposition 2.4 and Lemma 4.7). By Lemma 4.1 $\|\mathcal{F}\|$ is a sup-dominion for $\mathcal{F}$. If $\|\mathcal{F}\| = A$, then clearly $\|\mathcal{F}\| \leq \|\mathcal{F}\|_{U,V}$, by Lemma 4.6. Otherwise $\|\mathcal{F}\| > A$, so by Lemma 4.5 there exists a sup-dominion decomposition $(\|\mathcal{F}\|, \mathcal{H})$ for $\mathcal{F}$ such that $T_\mathcal{H}$ is an equitable tree of height $|\Theta|_\mu$ and width at most $|\|\mathcal{F}\| - A|_1 \leq n$. It follows that $T_\mathcal{H}$ embeds in U, and thus $\|\mathcal{F}\| \leq \|\mathcal{F}\|_{T_\mathcal{H},V} \leq \|\mathcal{F}\|_{U,V}$ by Lemmata 4.8 and 4.9. ◀

5 Lower bound

We now present a lower bound for the number of queries used in our method. To this end, we prove that our algorithms work only if they are driven by universal trees:

▶ **Theorem 5.1.** *Let U, V be equitable trees of height h.*

(1) *If $\|\mathcal{F}\|_{U,V} = \|\mathcal{F}\|$ for all fixed-point expressions over $\mathbb{B}^n$ of the form $\mathcal{F} = (\Theta, f, (\mathbf{0}, \mathbf{1}))$, where $|\Theta|_\mu = |\Theta|_\nu = h$, then U and V are (n, h)-universal.*

(2) *If $\|\mathcal{F}\|_V = \|\mathcal{F}\|$ for all fixed-point expressions as above, then V is (n, h)-universal.*

To estimate the number of queries used by the algorithms of Section 3, we combine Theorem 5.1 with a result by Czerwiński *et al.* [8, Theorem 2.3] saying that any (n, h)-universal tree has at least $\binom{\lfloor \lg n\rfloor + h - 1}{\lfloor \lg n\rfloor}$ leaves (which is at least $n^{\lg(h/\lg n)-1}$ if $h \leq n \lg n$, and at least $\left(\frac{n}{2}\right)^{\lg(h/\lg n)}$ in general). Recall that the number of queries used by our algorithms is proportional to widths of the employed trees (cf. Lemmata 3.2 and 3.6), thus by the above it is also quasi-polynomial.

The rest of this section is devoted to the proof of Theorem 5.1. Observe first that Point (2) of this theorem is a direct consequence of Point (1), because $\|\mathcal{F}\|_V = \|\mathcal{F}\|_{C_{n,h},V}$, by Lemma 3.4. It is thus enough to prove Point (1). Moreover, we can assume that $h \geq 1$, because for height $h = 0$ there exists only one tree $U = V = \langle\rangle$, which is $(n, 0)$-universal.

In order to prove Point (1), fix an equitable tree T of height $h \geq 1$ and of width (exactly) n. The core of our proof is a "difficult example": a fixed-point expression $\mathcal{F}_T$ such that if $\|\mathcal{F}_T\|_{U,V}$ is correct (i.e., equal to $\|\mathcal{F}_T\|$) for some U then T embeds in U, under some mild assumptions saying, roughly, that V is nontrivial. In order to achieve this property, we

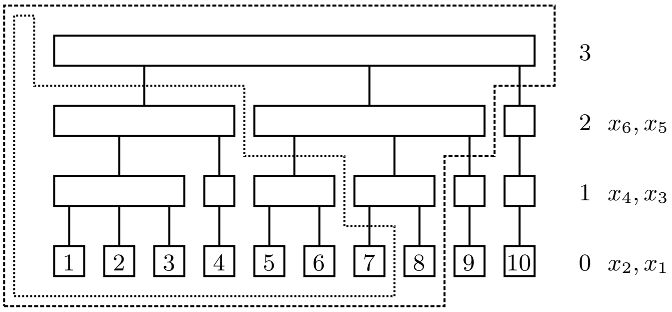


Figure 2 Example equitable tree T of height 3 with numbers in leaves. On the right: layer numbers and corresponding variable names. Dotted and dashed areas represent bits set to 1 in the values $A(v)$ and $B(v)$ for ancestors v of the 8-th leaf. Thus, the 8-th bit of $f_3(x_1, x_2, x_3, x_4, x_5, x_6)$ is 1 if at least the following bits are set to 1: the first 8 bits of x_1, the first 7 bits of x_2, the first 8 bits of x_3, the first 6 bits of x_4, the first 9 bits of x_5, and the first 4 bits of x_6.

construct $\mathcal{F}_T$ that has only one sup-dominion decomposition, which has the shape of the tree T (this claim needs not be proven, but it helps to understand the construction). The construction is to a large degree motivated by the work of Czerwiński *et al.* [8]. Existence of $\mathcal{F}_T$ for every T essentially implies that U is universal, as soon as we deal with the additional assumptions on the tree V, which is done at the very end.

Let us assign numbers $1, \dots, n$ to leaves of T, consecutively from left to right. For every node v we define two values $A(v), B(v) \in \mathbb{B}^n$, as follows:

- if the leftmost leaf descendant of v has number i, then bits number $1, 2, \dots, i-1$ in $A(v)$ are set to 1 (and the remaining bits are set to 0), and
- if the rightmost leaf descendant of v has number j, then bits number $1, 2, \dots, j$ in $B(v)$ are set to 1 (and the remaining bits are set to 0).

Having these values, for every level $\ell \in \{0, \dots, h\}$ we define two functions, $g_\ell^+, g_\ell^- : \mathbb{B}^n \to \mathbb{B}^n$. For every $x \in \mathbb{B}^n$, we consider the rightmost node v at level ℓ such that $A(v) \leq x$ (such a v exists, because $A(v) = \mathbf{0}$ for the extreme-leftmost node v), and we take

$$g_\ell^+(x) = B(v), \qquad \text{and} \qquad g_\ell^-(x) = \begin{cases} \mathbf{1} & \text{if } x = \mathbf{1}, \\ A(v) & \text{otherwise.} \end{cases}$$

Finally, for every level $\ell \in \{0, \dots, h\}$ we define a function $f_\ell \colon (\mathbb{B}^n)^{2\ell} \to \mathbb{B}^n$, by induction on ℓ, as follows:

$$\begin{aligned} &f_0() = \mathbf{1}, && \text{and} \\ &f_\ell(x_1, \dots, x_{2\ell-2}, x_{2\ell-1}, x_{2\ell}) = f_{\ell-1}(x_1, \dots, x_{2\ell-2}) * g_{\ell-1}^-(x_{2\ell-1}) * g_{\ell-1}^+(x_{2\ell}) && \text{for } \ell \geq 1. \end{aligned}$$

The sequence of fixed-point operators corresponding to f_ℓ is $\Theta_\ell = \langle \mu, \nu \rangle^\ell$. As the resulting expression we take $\mathcal{F}_T = (\Theta_h, f_h, (\mathbf{0}, \mathbf{1}))$. See Figure 2 for an example.

Directly from the definition it follows that for every node v at level ℓ,

$$g_\ell^+(A(v)) = B(v), \qquad g_\ell^-(B(v)) = B(v), \qquad g_\ell^-(x) \leq A(v) \text{ if } x < B(v). \tag{4}$$

Another useful property is that for every fixed-point expression of the form $(\Theta, f * C, (A, B))$ we can move the multiplicative constant C from "the function" to "the bound" (anyway, it will be just multiplied by the function):

$$\|(\Theta, f * C, (A, B))\| = \|(\Theta, f, (A, B * C))\|,$$

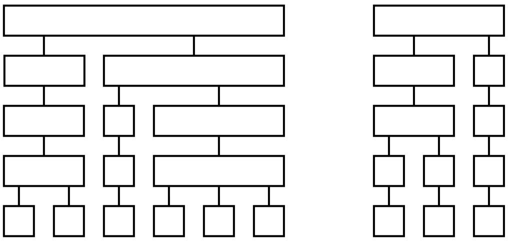


Figure 3 Example tree T (left) and the corresponding comb C_T (right).

or, more specifically (assuming $\ell \geq 1$),

$$\|(\Theta_{\ell-1}, (f_\ell{}^{\Rsh D})^{\Rsh C}, (A, B))\| = \|(\Theta_{\ell-1}, f_{\ell-1}, (A, B * g^-_{\ell-1}(C) * g^+_{\ell-1}(D)))\|. \tag{5}$$

The same holds for the value parameterised by two trees, $\|\cdot\|_{U,V}$.

Analysing the definition of $\mathcal{F}_T$, it is not difficult to prove by induction on ℓ that

$$\|(\Theta_\ell, f_\ell, (\mathbf{0}, B(v)))\| \geq B(v).$$

for every node v of T located at level ℓ (see Appendix E for details). In particular, taking the root of T as v, we obtain the following lemma.

▶ **Lemma 5.2.** $\|\mathcal{F}_T\| = \mathbf{1}$.

Next, we define trees used on the "inf" side. Namely, for every $\ell \in \{0, \dots, h\}$ we define an equitable tree C_ℓ ("comb") of height ℓ:

- $C_0 = \langle\rangle$,
- $C_\ell = \langle C_{\ell-1}\rangle$ for $\ell \geq 1$, if every node at level $\ell - 1$ in T has at most one child, and
- $C_\ell = \langle C_{\ell-1}, S_{\ell-1}\rangle$ for $\ell \geq 1$, if some node at level $\ell - 1$ in T has at least two children, where $S_{\ell-1}$ is the (unique) tree of height $\ell - 1$ having exactly one leaf.

Moreover, we define $C_T = C_h$. See Figure 3 for an illustration. Notice that the information about T is "shifted by one level" in C_T; in particular C_1 always equals $\langle\langle\rangle\rangle$ (leaves of T have no children), and the shape of C_T does not depend on the fact whether or not the root of T has at least two children. Attention: trees C_ℓ and S_ℓ defined here should not be confused with universal trees $C_{n,h}$ and $S_{n,h}$ defined earlier.

It is easy to see by induction on $\ell \in \{1, \dots, h\}$ that the width of C_ℓ plus the number of nodes of T at level $\ell - 1$ is at most $n + 1$. In particular, the width of C_T is at most n.

For a node v of T we write $T(v)$ for the subtree of T starting at v.

▶ **Lemma 5.3.** *Let U be an equitable tree of height $\ell \in \{0, \dots, h\}$, and let v be a node of T located at level ℓ. If $T(v)$ does not embed in U, then $\|(\Theta_\ell, f_\ell, (A(v), B(v)))\|_{U,C_\ell} < B(v)$. If additionally v has at most one child, then $\|(\Theta_\ell, f_\ell, (A(v), B(v)))\|_{U,C_\ell} \leq A(v)$.*

Proof. The proof is by induction on ℓ. For $\ell = 0$ the node v is a leaf, so surely $T(v)$ (i.e., $\langle\rangle$) embeds in U; there is nothing to prove.

Suppose now that $\ell \geq 1$. Let $\Theta'_\ell = \langle\nu\rangle \cdot \langle\mu, \nu\rangle^{l-1}$ and let $U = \langle U_1, \dots, U_p\rangle$. We first prove that for every child c of v and for every $U' \in \{U_1, \dots, U_p\}$,

$$\|(\Theta'_\ell, f_\ell{}^{\Rsh A(c)}, (A(c), B(v)))\|_{U',C_\ell} \leq \begin{cases} B(c) & \text{if } T(c) \text{ embeds in } U', \\ A(c) & \text{otherwise.} \end{cases} \tag{6}$$

Denote $E = \|(\Theta'_\ell, {f_\ell}^{\Rsh A(c)}, (A(c), B(v)))\|_{U',C_\ell}$. Let

$$B_1 = \|(\Theta_{\ell-1}, ({f_\ell}^{\Rsh A(c)})^{\Rsh B(v)}, (A(c), B(v)))\|_{U',C_{\ell-1}} \qquad \text{and} \tag{7}$$

$$B_2 = \|(\Theta_{\ell-1}, ({f_\ell}^{\Rsh A(c)})^{\Rsh B_1}, (A(c), B_1))\|_{U',S_{\ell-1}}. \tag{8}$$

By definition, E equals B_1 when $C_\ell = \langle C_{\ell-1}\rangle$, and equals B_2 when $C_\ell = \langle C_{\ell-1}, S_{\ell-1}\rangle$. Because $g^-_{\ell-1}(B(v)) = B(v) \geq B(c)$ and $g^+_{\ell-1}(A(c)) = B(c)$ (cf. Equalities (4)), we can simplify Equalities (7) and (8) (using also Equality (5)) to

$$B_1 = \|(\Theta_{\ell-1}, f_{\ell-1}, (A(c), B(c)))\|_{U',C_{\ell-1}}, \qquad \text{and} \tag{9}$$

$$B_2 = \|(\Theta_{\ell-1}, f_{\ell-1}, (A(c), B(c) * g^-_{\ell-1}(B_1)))\|_{U',S_{\ell-1}}. \tag{10}$$

We now have three cases.

First, suppose that $T(c)$ embeds in U'. In this case $B_2 \leq B_1 \leq B(c)$ by Lemma 4.6 and Equalities (8) and (9), so $E \leq B(c)$ (no matter whether E equals B_1 or B_2).

Next, suppose that $T(c)$ does not embed in U' and that $C_\ell = \langle C_{\ell-1}\rangle$. By definition of C_ℓ this means that c (and every other node at level $\ell - 1$) has at most one child. Thus, the induction hypothesis implies that $\|(\Theta_{\ell-1}, f_{\ell-1}, (A(c), B(c)))\|_{U',C_{\ell-1}} \leq A(c)$, that is, $E = B_1 \leq A(c)$ (by Equality (9)).

Finally, suppose that $T(c)$ does not embed in U', but $C_\ell = \langle C_{\ell-1}, S_{\ell-1}\rangle$. In this case the induction hypothesis implies that $\|(\Theta_{\ell-1}, f_{\ell-1}, (A(c), B(c)))\|_{U',C_{\ell-1}} < B(c)$, that is, $B_1 < B(c)$ (by Equality (9)). Then $g^-_{\ell-1}(B_1) \leq A(c)$ (cf. Equalities (4)), so $E = B_2 \leq A(c)$ by Lemma 4.6 and Equality (10). This finishes the proof of Inequality (6).

Recall now that $\|(\Theta_\ell, f_\ell, (A(v), B(v)))\|_{U,C_\ell}$ is defined as A_p, where $A_j = \|(\Theta'_\ell, {f_\ell}^{\Rsh A_{j-1}}, (A_{j-1}, B(v)))\|_{U_j,C_\ell}$ for $j \in \{1, \dots, p\}$ and $A_0 = A(v)$. Notice that A_0 equals $A(c)$ for the leftmost child c of v. If $A_{j-1} \leq A(c)$ for some child c of v, by monotonicity we have $A_j \leq \|(\Theta'_\ell, {f_\ell}^{\Rsh A(c)}, (A(c), B(v)))\|_{U_j,C_\ell}$, so we can apply Inequality (6). If $A_{j-1} \leq A(c)$ and $T(c)$ does not embed in U_j, we obtain that $A_j \leq A(c)$ as well. Contrarily, if $A_{j-1} \leq A(c)$ and $T(c)$ embeds in U_j, we obtain that $A_j \leq B(c) = A(c')$, where c' is the right sibling of c. We know that $T(v)$ does not embed in U, so $\|(\Theta_\ell, f_\ell, (A(v), B(v)))\|_{U,C_\ell} = A_p \leq A(c)$ for some child c of v (this is the first child that "could not be embedded"), and $A(c) < B(v)$, as required in the thesis. If moreover v has exactly one child (i.e., c is the only child of v), then $A(c) = A(v)$. ◀

▶ **Corollary 5.4.** *Let U, V be equitable trees of height h. If $\|\mathcal{F}_T\|_{U,V} = \|\mathcal{F}_T\|$ and C_T embeds in V, then T embeds in U. If $\|\widetilde{\mathcal{F}_T}\|_{U,V} = \|\widetilde{\mathcal{F}_T}\|$ and C_T embeds in U, then T embeds in V.*

Proof. For the first part, by Lemmata 5.2 and 4.8 we have $\mathbf{1} = \|\mathcal{F}_T\| = \|\mathcal{F}_T\|_{U,V} \leq \|\mathcal{F}_T\|_{U,C_T}$, that is, $\|\mathcal{F}_T\|_{U,C_T} = \mathbf{1}$. But Lemma 5.3 (where as v we take the root of T) allows $\|\mathcal{F}_T\|_{U,C_T}$ to be $\mathbf{1}$ only if T embeds in U. The second part is a consequence of the first part and of the equalities $\|\widetilde{\mathcal{F}_T}\| = \overline{\|\mathcal{F}_T\|}$ (Proposition 2.4) and $\|\widetilde{\mathcal{F}_T}\|_{U,V} = \overline{\|\mathcal{F}_T\|_{V,U}}$ (Lemma 4.7). ◀

We now finish the proof of (Point (1) of) Theorem 5.1. From this point, the tree T is no longer fixed. Recall that we have two equitable trees U, V of height $h \geq 1$, and we assume that $\|\mathcal{F}\|_{U,V} = \|\mathcal{F}\|$ for all fixed-point expressions of the form $\mathcal{F} = (\Theta, f, (\mathbf{0}, \mathbf{1}))$, where $|\Theta|_\mu = |\Theta|_\nu = h$. The goal is to prove that U and V are (n, h)-universal, that is, every equitable tree T of height h and width at most n can be embedded in U and V.

The difficulty is that in order to prove that a tree T embeds in U, we already need to know that some tree, namely C_T, embeds in V, and in order to prove that a tree T embeds in V, we already need to know that some tree, namely C_T, embeds in U (cf. Corollary 5.4). In order to deal with this circular dependency, we argue that C_T is "simpler" than T. To compare these trees, we introduce a parameter called a stick length.

The *stick length* of an equitable tree T of height h is the greatest number $s \in \{0, \dots, h\}$ such that all nodes at levels $0, 1, \dots, s$ have at most one child. We prove by induction on $h - s$ that every equitable tree T of height h, width at most n, and stick length s embeds in U and in V.

The only tree T of height h and stick length $s = h$ is S_h (a single branch); it clearly embeds in every tree of height h, in particular in U and in V.

For an induction step, consider an equitable tree T of height h, width at most n, and stick length $s < h$. Recall that the corresponding tree C_T has height h and width at most n. It is also easy to see that it has stick length $s + 1$ (if all nodes at some level ℓ in T have at most one child, then all nodes at level $\ell + 1$ in C_T have at most one child). Thus, C_T embeds in U and in V by the induction hypothesis. If T has width n, it embeds in U and in V by Corollary 5.4, as required. If the width is smaller, we consider the tree T' of width n obtained from T by attaching an appropriate number of trees S_{h-1} directly below the root. This tree also has stick length s, height h, and is equitable, so it embeds in U and in V by the above, and hence T embeds as well.

▶ Remark 5.5. Theorem 5.1 can be strengthened in the following way: it is enough to assume that the equalities $\|\mathcal{F}\|_{U,V} = \|\mathcal{F}\|$ and $\|\mathcal{F}\|_U = \|\mathcal{F}\|$ hold for those monotonic functions f, where every output bit is a logical OR or a logical AND of some input bits. Indeed, all functions constructed in the proof are of this kind. This makes a connection with parity games (cf. Introduction).

6 Conclusion

It is well known that a nested fixed point like $\mu x.\nu y.\mu x.f(x, y, z)$ can be computed by a formula

$$\mu^\gamma z.\nu^\beta y.\mu^\alpha x.f(x, y, z),$$

for sufficiently large ordinals α, β, γ, where $\theta^\eta x$ means that only η iterations of a fixed point θx are computed (cf. e.g. [27]). Our algorithms refine this idea in two ways: the iterations follow a more subtle tree structure, and moreover they are combined with narrowing the scope according to the nested $A + B * f$ pattern. In the asymmetric case, the restrictions apply only to one kind of fixed points, but the complexity – somewhat surprisingly – actually improves as we eventually compute a single fixed point (albeit of a larger system).

Our black-box algorithms, when adapted to parity games, match the best complexity known so far, but do not improve it. Our lower bound, in view of Remark 5.5, gives an evidence that in the algorithm by Jurdziński and Morvan [19] the use of (not others but) universal trees is indeed necessary.

This may suggest that any polynomial algorithm for parity games – if it exists – should involve some other structure of the game, which remains to be discovered. On the other hand, it is an intriguing question if the lower bound can be strengthen to an “absolute” lower bound for the number of queries, like the aforementioned (almost) quadratic lower bound [24]. A recently developed theory around the complexity of computing the Nash equilibria [9] warns us that some problems in NP ∩ co-NP may be hard.

References

1 André Arnold and Damian Niwiński. *Rudiments of μ-Calculus.* Studies in Logic and the Foundations of Mathematics. Elsevier, 2001.

2 Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: from parity games to safety games. *RAIRO Theor. Informatics Appl.*, 36(3):261–275, 2002. doi:10.1051/ita:2002013.

3 Mikołaj Bojańczyk and Wojciech Czerwiński. *An automata toolbox.* Informal lecture notes, 2018. URL: `https://www.mimuw.edu.pl/~bojan/upload/reduced-may-25.pdf`.

4 Anca Browne, Edmund M. Clarke, Somesh Jha, David E. Long, and Wilfredo R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theor. Comput. Sci.*, 178(1-2):237–255, 1997. `doi:10.1016/S0304-3975(96)00228-9`.

5 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263. ACM, 2017. `doi:10.1145/3055399.3055409`.

6 Krishnendu Chatterjee, Wolfgang Dvořák, Monika Henzinger, and Alexander Svozil. Quasipolynomial set-based symbolic algorithms for parity games. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, volume 57 of *EPiC Series in Computing*, pages 233–253. EasyChair, 2018. URL: `http://www.easychair.org/publications/paper/L8b1`.

7 Thomas Colcombet and Nathanaël Fijalkow. Universal graphs and good for games automata: New tools for infinite duration games. In Mikołaj Bojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2019. `doi:10.1007/978-3-030-17127-8_1`.

8 Wojciech Czerwiński, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdziński, Ranko Lazić, and Paweł Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2333–2349. SIAM, 2019. `doi:10.1137/1.9781611975482.142`.

9 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *Commun. ACM*, 52(2):89–97, 2009. `doi:10.1145/1461928.1461951`.

10 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. `doi:10.1109/SFCS.1991.185392`.

11 E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model checking for the μ-calculus and its fragments. *Theor. Comput. Sci.*, 258(1-2):491–522, 2001. `doi:10.1016/S0304-3975(00)00034-7`.

12 John Fearnley, Sanjay Jain, Sven Schewe, Frank Stephan, and Dominik Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In Hakan Erdogmus and Klaus Havelund, editors, *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017*, pages 112–121. ACM, 2017. `doi:10.1145/3092282.3092286`.

13 Hugo Gimbert and Rasmus Ibsen-Jensen. A short proof of correctness of the quasi-polynomial time algorithm for parity games. *CoRR*, abs/1702.01953, 2017. `arXiv:1702.01953`.

14 Daniel Hausmann and Lutz Schröder. Computing nested fixpoints in quasipolynomial time. *CoRR*, abs/1907.07020v2, 2019. `arXiv:1907.07020v2`.

15 Daniel Hausmann and Lutz Schröder. Quasipolynomial computation of nested fixpoints. *CoRR*, abs/1907.07020v3, 2020. `arXiv:1907.07020v3`.

16 Marcin Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998. `doi:10.1016/S0020-0190(98)00150-1`.

17 Marcin Jurdziński. Small progress measures for solving parity games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000. doi:10.1007/3-540-46541-3_24.

18 Marcin Jurdziński and Ranko Lazić. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–9. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005092.

19 Marcin Jurdziński and Rémi Morvan. A universal attractor decomposition algorithm for parity games. *CoRR*, abs/2001.04333, 2020. arXiv:2001.04333.

20 Marcin Jurdziński, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 117–123. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109571.

21 Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 639–648. ACM, 2018. doi:10.1145/3209108.3209115.

22 Karoliina Lehtinen, Sven Schewe, and Dominik Wojtczak. Improving the complexity of Parys' recursive algorithm. *CoRR*, abs/1904.11810, 2019. arXiv:1904.11810.

23 Robert McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic*, 65(2):149–184, 1993. doi:10.1016/0168-0072(93)90036-D.

24 Paweł Parys. Some results on complexity of μ-calculus evaluation in the black-box model. *RAIRO - Theor. Inf. and Applic.*, 47(1):97–109, 2013. doi:10.1051/ita/2012030.

25 Paweł Parys. Parity games: Zielonka's algorithm in quasi-polynomial time. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 10:1–10:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.10.

26 Helmut Seidl. Fast and simple nested fixpoints. *Inf. Process. Lett.*, 59(6):303–308, 1996. doi:10.1016/0020-0190(96)00130-5.

27 Igor Walukiewicz. Monadic second order logic on tree-like structures. In Claude Puech and Rüdiger Reischuk, editors, *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings*, volume 1046 of *Lecture Notes in Computer Science*, pages 401–413. Springer, 1996. doi:10.1007/3-540-60922-9_33.

28 Shipei Zhang, Oleg Sokolsky, and Scott A. Smolka. On the parallel complexity of model checking in the modal mu-calculus. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994*, pages 154–163. IEEE Computer Society, 1994. doi:10.1109/LICS.1994.316075.

29 Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

A Proof of Proposition 2.3

▶ **Proposition 2.3.** *If $A \le C \le \|(\Theta, f, (A, B))\| \le D \le B$, then*

$$\|(\Theta, f, (A, B))\| = \|(\Theta, f, (C, D))\|.$$

Proof. The proof is by induction on the length of Θ.

In the base case of $\Theta = \langle\rangle$, the thesis amounts to $A + B * f() = C + D * f()$, assuming that $A \le C \le A + B * f() \le D \le B$. But we have that $B * f() \le A + B * f() \le D$ and

$B * f() \leq f()$, which implies that $B * f() \leq D * f()$. Since moreover $A \leq C$, we obtain that $A + B * f() \leq C + D * f()$. The converse inequality $A + B * f() \geq C + D * f()$ can be proven analogously.

Suppose now that $\Theta = \langle\theta\rangle \cdot \Theta'$. Let $E = \|(\Theta, f, (A, B))\|$. Then $E = \theta x.g(x)$ where $g(x) = \|(\Theta', f^{\Rsh x}, (A, B))\|$. By Proposition 2.2 it follows that $A \leq g(x) \leq B$ for all $x \in \mathbb{B}^n$, which due to the inequalities $A \leq C$ and $D \leq B$ implies that $A \leq C * g(x) \leq g(x) \leq D + g(x) \leq B$. By the induction hypothesis,

$$g(x) = \|(\Theta', f^{\Rsh x}, (A, B))\| = \|(\Theta', f^{\Rsh x}, (C * g(x), D + g(x)))\|.$$

Let $h(x, y) = \|(\Theta', f^{\Rsh x}, (C * g(y), D + g(y)))\|$, so that $h(x, x) = g(x)$. By Equality (3), $E = \theta x.g(x) = \theta y.\theta x.h(x, y)$, so $E = \theta x.h(x, E)$. Recalling that $g(E) = E$, by assumption we have that $C \leq g(E) \leq D$. Unravelling the definition of h in $E = \theta x.h(x, E)$, we obtain that

$$E = \theta x.\|(\Theta', f^{\Rsh x}, (C * g(E), D + g(E)))\| = \theta x.\|(\Theta', f^{\Rsh x}, (C, D))\| = \|(\Theta, f, (C, D))\|,$$

as required. ◀

B Proof of Lemma 3.4

Heading towards a proof of Lemma 3.4, we first note an analogue to Proposition 2.2 and Lemma 4.6, which follows by a straightforward induction:

▶ **Lemma B.1.** *For every fixed-point expression* $(\Theta, f, (A, B))$ *and for every equitable tree* V *of height* $|\Theta|_\nu$*,*

$$A \leq \|(\Theta, f, (A, B))\|_V \leq B.$$

As already mentioned, while proving Lemma 3.4 we make use of the following simple fact.

▶ **Proposition B.2.** *For a monotone mapping* $f \colon \mathbb{B}^n \to \mathbb{B}^n$*,*

$$\mu x.f(x) = \underbrace{f(\dots(f}_{n}(\mathbf{0}))\dots).$$

We now generalize Lemma 3.4 a bit, to a variant suitable for an inductive proof (Lemma 3.4 follows from Lemma B.3 by taking $A' = A$):

▶ **Lemma B.3.** *Let* $(\Theta, f, (A, B))$ *be a fixed-point expression, and let* V *be an equitable tree of height* $|\Theta|_\nu$*. If* $A \leq A' \leq \|(\Theta, f, (A, B))\|_V$*, then*

$$\|(\Theta, f, (A, B))\|_V = \|(\Theta, f, (A', B))\|_{C_{n,|\Theta|_\mu},V}.$$

Proof. Induction on the length of Θ. Let $h = |\Theta|_\mu$. For $\Theta = \langle\rangle$ on the one hand due to $A \leq A'$ we have

$$\|(\Theta, f, (A, B))\|_V = A + B * f() \leq A' + B * f() = \|(\Theta, f, (A', B))\|_{C_{n,h},V},$$

and on the other hand due to $A' \leq \|(\Theta, f, (A, B))\|_V$ we have

$$A' + B * f() \leq A + B * f() + B * f() = A + B * f().$$

Next, suppose that $\Theta = \langle\nu\rangle \cdot \Theta'$ and $V = \langle V_1, \dots, V_p\rangle$. Let $B_0 = B'_0 = B$ and for $j \in \{1, \dots, p\}$, let

$$B_j = \|(\Theta', f^{\Rsh B_{j-1}}, (A, B_{j-1}))\|_{V_j}, \qquad\qquad B'_j = \|(\Theta', f^{\Rsh B'_{j-1}}, (A', B'_{j-1}))\|_{C_{n,h},V_j}.$$

Since $A \le A' \le \|(\Theta, f, (A, B))\|_V = B_p \le B_{p-1} \le \dots \le B_0$ (the inequalities between the B_j's are by Lemma B.1), we have by the induction hypothesis

$$B_j = \|(\Theta', f^{\Rsh B_{j-1}}, (A', B_{j-1}))\|_{C_{n,h},V_j}.$$

Thus, $B_{j-1} = B'_{j-1}$ implies $B_j = B'_j$, and since $B_0 = B'_0 = B$, we have $\|(\Theta, f, (A, B))\|_V = B_p = B'_p = \|(\Theta, f, (A', B))\|_{C_{n,h},V}$.

Finally, suppose that $\Theta = \langle\mu\rangle \cdot \Theta'$. Let us define

$$E_0 = \mathbf{0}, \qquad E_j = \|(\Theta', f^{\Rsh E_{j-1}}, (A, B))\|_V \qquad \text{for } j \in \{1, \dots, n\}, \qquad \text{and} \tag{11}$$
$$A'_0 = A', \qquad A'_j = \|(\Theta', f^{\Rsh A'_{j-1}}, (A'_{j-1}, B))\|_{C_{n,h-1},V} \qquad \text{for } j \in \{1, \dots, n\}. \tag{12}$$

Let $E = E_n$. By Proposition B.2 we have $E = \mu x.\|(\Theta', f^{\Rsh x}, (A, B))\|_V = \|(\Theta, f, (A, B))\|_V$. Simultaneously, by the definition of $\|(\Theta, f, (A', B))\|_{C_{n,h},V}$ we have $\|(\Theta, f, (A', B))\|_{C_{n,h},V} = A'_n$. Moreover, by Lemma 4.6 we have $A \le A' = A'_0 \le A'_1 \le \dots \le A'_n$, that is, $A \le A'_j$ for all $j \in \{0, \dots, n\}$. We now prove by induction on $j \in \{0, \dots, n\}$ that $E_j \le A'_j \le E$ (for $j = n$ we obtain the required equality $E = E_n = A'_n$). For $j = 0$ the inequality boils down to $\mathbf{0} \le A' \le E$, which holds by assumption. Suppose thus that $j \in \{1, \dots, n\}$ and that $E_{j-1} \le A'_{j-1} \le E$. By monotonicity and by the induction hypothesis it follows that

$$E_j \overset{(11)}{\le} \|(\Theta', f^{\Rsh A'_{j-1}}, (A'_{j-1}, B))\|_V = \|(\Theta', f^{\Rsh A'_{j-1}}, (A'_{j-1}, B))\|_{C_{n,h},V} = A'_j.$$

Again by monotonicity and again by the induction hypothesis, since $A \le A'_{j-1} \le E = \|(\Theta', f^{\Rsh E}, (A, B))\|_V$,

$$A'_j \overset{(12)}{\le} \|(\Theta', f^{\Rsh E}, (A'_{j-1}, B))\|_{C_{n,h},V} = \|(\Theta', f^{\Rsh E}, (A, B))\|_V = E. \qquad ◀$$

C An explicit definition of the system of equations

In this section we give an explicit description of the system of equations generated by Algorithm 1 for $\|(\Theta, f, (A, B))\|_V$, where $|\Theta|_\nu = h$. Without loss of generality we may assume that $\Theta = \langle\mu\rangle \cdot \langle\nu, \mu\rangle^h$; to reach such a situation from a general case, one can add additional parameters (quantified by μ) on which f does not depend (and compress neighbouring occurrences of μ using Equality (3)). Suppose that leaves of V are numbered $1, \dots, m$ from left to right (where $m = |V|$). For all $i \in \{1, \dots, m\}$ and $\ell \in \{0, \dots, h\}$, let $\lambda^\ell(i)$ be the least number $j \in \{0, \dots, m-1\}$ such that leaves number $j + 1$ and i have the same ancestor at level ℓ, and let $\rho^\ell(i)$ be the greatest number $j \in \{1, \dots, m\}$ such that leaves number i and j have the same ancestor at level ℓ (in particular, $\lambda^0(i) = i - 1$ and $\rho^0(i) = i$). In other words, if v is the ancestor of the i-th leaf located at level ℓ, then $\lambda^\ell(i)$ is the number of the leftmost leaf descendant of v, decreased by one, and $\rho^\ell(i)$ is the number of the rightmost leaf descendant of v. In the system, we use variables $\mathsf{x}_1, \dots, \mathsf{x}_m$ and additionally the variable x_0 that is valuated to $\mathbf{1}$. For every $i \in \{1, \dots, m\}$ we have an equation

$$\mathsf{x}_i = \mathsf{x}_{\lambda^0(i)} * \mathsf{f}(\mathsf{x}_{\rho^0(i)}, \mathsf{x}_{\lambda^0(i)}, \mathsf{x}_{\rho^1(i)}, \mathsf{x}_{\lambda^1(i)}, \mathsf{x}_{\rho^2(i)}, \mathsf{x}_{\lambda^2(i)}, \dots, \mathsf{x}_{\rho^{h-1}(i)}, \mathsf{x}_{\lambda^{h-1}(i)}, \mathsf{x}_{\rho^h(i)}). \tag{$\star$}$$

It is the variable x_m that stores the result (i.e., $x_{res} = \mathsf{x}_m$).

We now prove that the system generated by Algorithm 1 is indeed of the above form. Originally, the last argument of the procedure Generate is a subtree of the input tree V. Let us consider a modification of this procedure, where the last argument is a node of V, being the root of this subtree (and then, in the procedure, we consider the subtree starting in this node).

Consider now a recursive call

$$\textsc{Generate}(x, \langle\mu\rangle \cdot \langle\nu,\mu\rangle^{\ell}, \mathbf{f}, \mathbf{B}, u) \qquad \text{with } \mathbf{f} = \mathsf{f}(?,\dots,?,y_{2\ell+2},\dots,y_{2h+1})$$

performed during the considered execution of Algorithm 1. It is easy to prove (by induction on the depth of the recursion) that, if i is the number of the rightmost leaf descendant of the node u,

- u is located at level ℓ,
- for every $j \in \{\ell+1,\dots,h\}$, the argument y_{2j} contains the variable $\mathsf{x}_{\lambda^{j-1}(i)}$, and the argument y_{2j+1} contains the variable $\mathsf{x}_{\rho^{j}(i)}$,
- the argument $\mathbf{B}$ contains the variable $\mathsf{x}_{\lambda^{\ell}(i)}$, and
- the argument x contains the variable $\mathsf{x}_{\rho^{\ell}(i)}$, that is, x_i.

From the above claim it follows that equations generated in line 4 of the algorithm are indeed of the form ($\star$).

D Proof of Lemma 3.5

▶ **Lemma 3.5.** *Let $\mathcal{F} = (\Theta, f, (\mathbf{0},\mathbf{1}))$ be a fixed-point expression, and let V be an equitable tree of height $|\Theta|_\nu$. The value assigned to the variable x_{res} in the least solution of the system of equations generated by Algorithm 1 equals $\|\mathcal{F}\|_V$.*

Proof. In order to facilitate an inductive proof of this lemma, we slightly modify Algorithm 1. Namely, we replace line 14 by the following three lines:

```
$x' = \textsc{FreshVariable}()$;
$\textsc{Generate}(x, \Theta', \mathbf{f}^{\Rsh x'}, \mathbf{B}, V)$;
output "$x' = x$";
```

This way, we replace some occurrences of the variable x in the resulting system of equations by a fresh variable x', and we add an equation ensuring that x' equals x. It should be clear that there is a one-to-one correspondence between solutions of the original system of equations and solutions of the new one; in particular the least solution is the same (modulo the difference that in the new system some variables are multiplicated). Thus, from now on we only consider the modified algorithm.

Let d be the length of Θ. We prove a claim concerning any recursive call

$$\textsc{Generate}(x, \widehat{\Theta}, \mathbf{f}, \mathbf{B}, \widehat{V}) \qquad \text{with } \mathbf{f} = \mathsf{f}(?,\dots,?,y_{k+1},\dots,y_d).$$

We assume that k (i.e., the number of ?'s) equals the length of $\widehat{\Theta}$, and $\widehat{V}$ is an equitable tree of height $|\widehat{\Theta}|_\nu$ (we use here symbols $\widehat{\Theta}$ and $\widehat{V}$, not Θ and V, in order to distinguish these arguments from the original sequence Θ and from the original tree V). Moreover, we assume that the variable contained in the argument x is not contained in any of the arguments $y_{k+1},\dots,y_d,\mathbf{B}$. All calls appearing in the modified version of the algorithm satisfy the above two conditions (while the second condition is not satisfied by the original algorithm; this is why we consider the modification). Let $\mathcal{S}$ be the system of equations generated by the above call. A *solution* of $\mathcal{S}$ is a function that maps every variable appearing in $\mathcal{S}$ (including those that do not appear on the left side of any equation) to an element of $\mathbb{B}^n$. Consider also a valuation $\mathit{val}\colon \{y_{k+1},\dots,y_d,\mathbf{B}\}\to\mathbb{B}^n$. Based on the the original function f (of arity d), the valuation induces a new function f_{val} of arity k, in an obvious way. Let $B=\mathit{val}(\mathbf{B})$ and $E=\|(\widehat{\Theta}, f_{\mathit{val}}, (\mathbf{0},B))\|_{\widehat{V}}$.

▷ Claim.

(1) For every solution R of $\mathcal{S}$ that extends val it holds $R(x)\geq E$, and

(2) there exists a solution R of $\mathcal{S}$ that extends val and such that $R(x)=E$.

Equivalently, the claim says that in the least solution of $\mathcal{S}$ among those that extend the valuation val, the variable x is valuated to E. In particular, while considering the main call of the algorithm (i.e., the call in line 17), this claim gives us the thesis of the lemma. It thus remains to prove the claim, which is done by induction on k.

For $k=0$ (i.e., $\widehat{\Theta}=\langle\rangle$) the system $\mathcal{S}$ consists of a single equation, namely $x=\mathbf{B}*\mathsf{f}(y_1,\dots,y_d)$. The valuation val already assigns values to the variables $y_1,\dots,y_d,\mathbf{B}$ (but not to x), so there is a unique solution that extends val; it maps x to $B*f_{\mathit{val}}()$, that is, to E.

Next, suppose that $\widehat{\Theta}=\langle\nu\rangle\cdot\widehat{\Theta}'$. Let $\widehat{V}=\langle\widehat{V}_1,\dots,\widehat{V}_p\rangle$, let $B_0=B$, and for $j\in\{1,\dots,p\}$ let $B_j=\|(\widehat{\Theta}',(f_{\mathit{val}})^{\Rsh B_{j-1}},(\mathbf{0},B_{j-1}))\|_{\widehat{V}_j}$. Then, by definition, $E=B_p$. Moreover, for every $j\in\{1,\dots,p\}$, let $\mathcal{S}_j$ be the system of equations generated by the call $\textsc{Generate}(\mathbf{B}_j,\widehat{\Theta}',\mathbf{f}^{\Rsh\mathbf{B}_{j-1}},\mathbf{B}_{j-1},\widehat{V}_j)$ in line 11 of the algorithm.

Consider now a solution R of $\mathcal{S}$ that extends val. We prove by induction on $j\in\{0,\dots,p\}$ that $R(\mathbf{B}_j)\geq B_j$; for $j=p$ this gives Point (1) of the claim, because $\mathbf{B}_p=x$ and $B_p=E$. For $j=0$ we have $\mathbf{B}_0=\mathbf{B}$, so $R(\mathbf{B}_0)=\mathit{val}(\mathbf{B})=B=B_0$. For the induction step, we prove $R(\mathbf{B}_j)\geq B_j$ assuming $R(\mathbf{B}_{j-1})\geq B_{j-1}$, where $j\in\{1,\dots,p\}$. Because $\mathcal{S}_j$ is a part of $\mathcal{S}$, our solution R (when restricted to variables appearing in $\mathcal{S}_j$) is also a solution of $\mathcal{S}_j$. Moreover, it extends $\mathit{val}[\mathbf{B}_{j-1}\mapsto R(\mathbf{B}_{j-1})]$. Observe that the function of arity $k-1$ induced by the valuation $\mathit{val}[\mathbf{B}_{j-1}\mapsto R(\mathbf{B}_{j-1})]$ (i.e., $f_{\mathit{val}[\mathbf{B}_{j-1}\mapsto R(\mathbf{B}_{j-1})]}$) equals $(f_{\mathit{val}})^{\Rsh R(\mathbf{B}_{j-1})}$ (below, similar equalities are used implicitly). Thus, by the induction hypothesis (used for this valuation) and by monotonicity,

$$R(\mathbf{B}_j)\geq\|(\widehat{\Theta}',(f_{\mathit{val}})^{\Rsh R(\mathbf{B}_{j-1})},(\mathbf{0},R(\mathbf{B}_{j-1})))\|_{\widehat{V}_j}\geq\|(\widehat{\Theta}',(f_{\mathit{val}})^{\Rsh B_{j-1}},(\mathbf{0},B_{j-1}))\|_{\widehat{V}_j}=B_j.$$

In order to prove Point (2) of the claim, for every $j\in\{1,\dots,p\}$ we create by the induction hypothesis a solution R_j to $\mathcal{S}_j$ that extends $\mathit{val}[\mathbf{B}_{j-1}\mapsto B_{j-1}]$ and such that $R_j(\mathbf{B}_j)=B_j$. Observe that the systems $\mathcal{S}_1,\dots,\mathcal{S}_p$ have pairwise disjoint sets of variables, except for the variables $y_{k+1},\dots,y_d$ whose values are fixed by val, and except for variables $\mathbf{B}_1,\dots,\mathbf{B}_{p-1}$ shared by consecutive systems, for which we also fix values in a consistent way. It follows that the solutions $R_1,\dots,R_p$ may be merged into a solution R of the whole system $\mathcal{S}$ (being a union of $\mathcal{S}_1,\dots,\mathcal{S}_p$). This solution extends val, and satisfies $R(x)=R_p(\mathbf{B}_p)=B_p=E$.

We now come to the last case, namely $\widehat{\Theta}=\langle\mu\rangle\cdot\widehat{\Theta}'$. Let $\mathcal{S}'$ be the system of equations generated by the call $\textsc{Generate}(x,\widehat{\Theta}',\mathbf{f}^{\Rsh x'},\mathbf{B},\widehat{V})$ in the modified line 14 of the algorithm; in other words, this is $\mathcal{S}$ without the equation $x'=x$. Recall that, in this case, E is the least fixed point of the mapping $C\mapsto\|(\widehat{\Theta}',(f_{\mathit{val}})^{\Rsh C},(\mathbf{0},B))\|_{\widehat{V}}$. For Point (1) of the claim, consider a solution R of $\mathcal{S}$ that extends val. This is also a solution of $\mathcal{S}'$, and $R(x')=R(x)$. Thus, by the induction hypothesis used for the valuation $\mathit{val}[x'\mapsto R(x)]$,

$$R(x)\geq\|(\widehat{\Theta}',(f_{\mathit{val}})^{\Rsh R(x)},(\mathbf{0},B))\|_{\widehat{V}}.$$

By Equalities (2) this means that $R(x)$ can only be greater than the least fixed point of the aforementioned mapping, that is $R(x)\geq E$. This proves Point (1). For Point (2), the induction hypothesis gives us a solution R of $\mathcal{S}'$ that extends $val[x' \mapsto E]$ and such that $R(x) = \|(\widehat{\Theta}', (f_{val})^{\Rsh E}, (\mathbf{0}, B))\|_{\widehat{V}}$, that is, $R(x) = E$. We also have $R(x') = E$, so R is a solution of $\mathcal{S}$ as well. This finishes the proof of the claim, and thus of the whole lemma. ◀

Lemmata 4.1 and 4.2 are immediate consequences of definitions and of Proposition 2.3.

E Proof of Lemma 5.2

As already mentioned, we obtain Lemma 5.2 by taking the root of T as v in the following lemma.

▶ **Lemma E.1.** *For every node v of T located at level ℓ,*

$$\|(\Theta_\ell, f_\ell, (\mathbf{0}, B(v)))\| \geq B(v).$$

Proof. The proof is by induction on ℓ. Let $E = \|(\Theta_\ell, f_\ell, (\mathbf{0}, B(v)))\|$. For $\ell = 0$ we simply have $E = \mathbf{0} + B(v) * \mathbf{1} = B(v)$.

Suppose now that $\ell \geq 1$. Let $\Theta'_\ell = \langle \nu \rangle \cdot \langle \mu, \nu \rangle^{l-1}$, and let c be the rightmost node at level $\ell - 1$ such that $A(c) \leq E$. We first prove that

$$\|(\Theta'_\ell, {f_\ell}^{\Rsh E}, (\mathbf{0}, B(c)))\| \geq B(c). \tag{13}$$

To this end, observe that

$$\begin{aligned} \|(\Theta_{\ell-1}, ({f_\ell}^{\Rsh E})^{\Rsh B(c)}, (\mathbf{0}, B(c)))\| &\overset{(5)}{=} \|(\Theta_{\ell-1}, f_{\ell-1}, (\mathbf{0}, B(c) * g^{-}_{\ell-1}(B(c)) * g^{+}_{\ell-1}(E)))\| \\ &\overset{(4)}{=} \|(\Theta_{\ell-1}, f_{\ell-1}, (\mathbf{0}, B(c) * B(c) * B(c)))\| \geq B(c), \end{aligned}$$

where the last inequality is by the induction hypothesis. Moreover, by Proposition 2.2 we have a converse inequality. Together, this means that $B(c)$ is a fixed point of the mapping $x \mapsto \|(\Theta_{\ell-1}, ({f_\ell}^{\Rsh E})^{\Rsh x}, (\mathbf{0}, B(c)))\|$. By definition $\|(\Theta'_\ell, {f_\ell}^{\Rsh E}, (\mathbf{0}, B(c)))\|$ is the greatest fixed points of this mapping, so it can be only greater. This finishes the proof of Inequality (13).

If $B(v) \leq A(c)$, the thesis (i.e., $E \geq B(v)$) is true, since $A(c) \leq E$. Otherwise, $B(v) > A(c)$ implies that $B(v) \geq B(c)$ (because c is not higher in the tree than v). Thus, because E is a fixed point and by monotonicity we have that

$$E = \|(\Theta'_\ell, {f_\ell}^{\Rsh E}, (\mathbf{0}, B(v)))\| \geq \|(\Theta'_\ell, {f_\ell}^{\Rsh E}, (\mathbf{0}, B(c)))\| \overset{(13)}{\geq} B(c).$$

If c is the rightmost node at level $\ell - 1$, we have $E \geq B(c) = \mathbf{1} \geq B(v)$ and again we are done. In the remaining case, the node c' one to the right from c satisfies $A(c') = B(c) \leq E$, contrary to the definition of c. ◀

Learning Concepts Described By Weight Aggregation Logic

Steffen van Bergerem
RWTH Aachen University, Germany
vanbergerem@informatik.rwth-aachen.de

Nicole Schweikardt
Humboldt-Universität zu Berlin, Germany
schweikn@informatik.hu-berlin.de

Abstract

We consider weighted structures, which extend ordinary relational structures by assigning weights, i.e. elements from a particular group or ring, to tuples present in the structure. We introduce an extension of first-order logic that allows to aggregate weights of tuples, compare such aggregates, and use them to build more complex formulas. We provide locality properties of fragments of this logic including Feferman-Vaught decompositions and a Gaifman normal form for a fragment called FOW_1, as well as a localisation theorem for a larger fragment called FOWA_1. This fragment can express concepts from various machine learning scenarios. Using the locality properties, we show that concepts definable in FOWA_1 over a weighted background structure of at most polylogarithmic degree are agnostically PAC-learnable in polylogarithmic time after pseudo-linear time preprocessing.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Complexity theory and logic; Computing methodologies → Logical and relational learning; Computing methodologies → Supervised learning

Keywords and phrases first-order definable concept learning, agnostic probably approximately correct learning, classification problems, locality, Feferman-Vaught decomposition, Gaifman normal form, first-order logic with counting, weight aggregation logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.10

Related Version A full version of the paper is available at [23], `https://arxiv.org/abs/2009.10574`.

Funding The second author has been partially supported by the ANR project EQUUS ANR-19-CE48-0019; funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 431183758 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 431183758).

Acknowledgements We thank Martin Grohe and Sandra Kiefer for helpful discussions on the subject.

1 Introduction

In this paper, we study Boolean classification problems. The elements that are to be classified come from a set $\mathcal{X}$, the *instance space*. A *classifier* on $\mathcal{X}$ is a function $c\colon \mathcal{X} \to \{0,1\}$. Given a *training sequence* T of labelled examples $(x_i, b_i) \in \mathcal{X} \times \{0,1\}$, we want to find a classifier, called a *hypothesis*, that can be used to predict the label of elements from $\mathcal{X}$ not given in T. We consider the following well-known frameworks for this setting from computational learning theory.

In Angluin's model of *exact learning* [1], the examples are assumed to be generated using an unknown classifier, the *target concept*, from a known *concept class*. The task is to find a hypothesis that is consistent with the training sequence T, i.e. a function $h\colon \mathcal{X} \to \{0,1\}$ such that $h(x_i) = b_i$ for all i. In Haussler's model of *agnostic probably approximately correct (PAC) learning* [11], a generalisation of Valiant's *PAC learning* model [21], an (unknown) probability

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 10; pp. 10:1–10:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

distribution $\mathcal{D}$ on $\mathcal{X} \times \{0,1\}$ is assumed and training examples are drawn independently from this distribution. The goal is to find a hypothesis that generalises well, i.e. one is interested in algorithms that return with high probability a hypothesis with a small expected error on new instances drawn from the same distribution. For more background on PAC learning, we refer to [12, 19]. We study learning problems in the framework that was introduced by Grohe and Turán [9] and further studied in [3, 6, 7, 22]. There, the instance space $\mathcal{X}$ is a set of tuples from a background structure and classifiers are described using parametric models based on logics.

Our Contribution. We introduce a new logic for describing such classifiers, namely *first-order logic with weight aggregation* (FOWA). It operates on *weighted structures*, which extend ordinary relational structures by assigning weights, i.e. elements from a particular abelian group or ring, to tuples present in the structure. Such weighted structures were recently considered by Toruńczyk [20], who studied the complexity of query evaluation problems for the related logic $\mathrm{FO}[\mathbb{C}]$ and its fragment $\mathrm{FO}_{\mathrm{G}}[\mathbb{C}]$. Our logic FOWA, however, is closer to the syntax and semantics of the first-order logic with counting quantifiers FOC considered in [13]. This connection enables us to achieve locality results for the fragments FOW_1 and FOWA_1 of FOWA similar to those obtained in [14, 8]. Specifically, we achieve Feferman-Vaught decompositions and a Gaifman normal form for FOW_1 as well as a localisation theorem for the more expressive logic FOWA_1. We provide examples illustrating that FOWA_1 can express concepts relevant for various machine learning scenarios. Using the locality properties, we show that concepts definable in FOWA_1 over a weighted background structure of at most polylogarithmic degree are agnostically PAC-learnable in polylogarithmic time after pseudo-linear time preprocessing. This generalises the results that Grohe and Ritzert [7] obtained for first-order logic to the substantially more expressive logic FOWA_1.

The main drawback of the existing logic-based learning results is that they deal with structures and logics that are too weak for describing meaningful classifiers for real-world machine learning problems. In machine learning, input data is often given via numerical values which are contained in or extracted from a more complex structure, such as a relational database (cf., [5, 10, 17, 18]). Hence, to combine these two types of information, we are interested in hybrid structures, which extend relational ones by numerical values. Just as in commonly used relational database systems, to utilise the power of such hybrid structures, the classifiers should be allowed to use different methods to aggregate the numerical values. Our main contribution is the design of a logic that is capable of expressing meaningful machine learning problems and, at the same time, well-behaved enough to have similar locality properties as first-order logic, which enable us to learn the concepts in sublinear time.

Outline. This paper is structured as follows. Section 2 fixes basic notation. Section 3 introduces the logic FOWA and its fragments FOW_1 and FOWA_1, provides examples, and discusses enrichments of the logic with syntactic sugar in order to make it more user-friendly (i.e. easier to parse or construct formulas) without increasing its expressive power. Section 4 provides locality results for the fragments FOW_1 and FOWA_1 that are similar in spirit to the known locality results for first-order logic and the counting logic FOC_1. Section 5 is devoted to our results on agnostic PAC learning. Section 6 combines the results from the previous sections to obtain our main learning theorem for FOWA_1, and concludes the paper with an application scenario and directions for future work. Due to space restrictions, we omit some proofs and proof details in this article; all these details can be found in the preliminary full version of this paper [23].

2 Preliminaries

Standard Notation. We write $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{N}$, and $\mathbb{N}_{\geqslant 1}$ for the sets of reals, rationals, integers, non-negative integers, and positive integers, respectively. For all $m, n \in \mathbb{N}$, we write $[m, n]$ for the set $\{k \in \mathbb{N} : m \leqslant k \leqslant n\}$, and we let $[m] := [1, m]$. For a k-tuple $\bar{x} = (x_1, \ldots, x_k)$, we write $|\bar{x}|$ to denote its *arity* k. By $()$, we denote the empty tuple, i.e. the tuple of arity 0. All graphs are assumed to be undirected. For a graph G, we write $V(G)$ and $E(G)$ to denote its vertex set and edge set, respectively. For $V' \subseteq V(G)$, we write $G[V']$ to denote the subgraph of G induced on V'. We assume familiarity with standard definitions concerning groups and rings (cf., [23]). When referring to an abelian group (or ring), we will usually write $(S, +_S)$ (or $(S, +_S, \cdot_S)$), we denote the neutral element of the group by 0_S, and $-a$ denotes the inverse of an element a in $(S, +_S)$ (and we denote the neutral element of the ring for $(S, \cdot_S)$ by 1_S).

Signatures, Structures, and Neighbourhoods. A *signature* σ is a finite set of relation symbols. Associated with every $R \in \sigma$ is an arity $\operatorname{ar}(R) \in \mathbb{N}$. A *$\sigma$-structure* $\mathcal{A}$ consists of a finite non-empty set A called the *universe* of $\mathcal{A}$ (sometimes denoted $U(\mathcal{A})$), and for each $R \in \sigma$ a relation $R^{\mathcal{A}} \subseteq A^{\operatorname{ar}(R)}$. The *size* of $\mathcal{A}$ is $|\mathcal{A}| := |A|$. Note that, according to these definitions, the universe $A = U(\mathcal{A})$ of each considered structure $\mathcal{A}$ is finite, and all considered signatures σ are *relational* (i.e. they do not contain any constants or function symbols), and may contain relation symbols of arity 0 (the only 0-ary relations over a set A are $\emptyset$ and $\{()\}$).

Let σ' be a signature with $\sigma' \supseteq \sigma$. A *σ'-expansion* of a σ-structure $\mathcal{A}$ is a σ'-structure $\mathcal{B}$ with universe B such that $B = A$ and $R^{\mathcal{B}} = R^{\mathcal{A}}$ for every $R \in \sigma$. If $\mathcal{B}$ is a σ'-expansion of $\mathcal{A}$, then $\mathcal{A}$ is called the *σ-reduct* of $\mathcal{B}$. A *substructure* of a σ-structure $\mathcal{A}$ is a σ-structure $\mathcal{B}$ with a universe $B \subseteq A$ and $R^{\mathcal{B}} \subseteq R^{\mathcal{A}}$ for all $R \in \sigma$. For a σ-structure $\mathcal{A}$ and a non-empty set $B \subseteq A$, we write $\mathcal{A}[B]$ to denote the *induced substructure* of $\mathcal{A}$ on B, i.e. the σ-structure with universe B and $R^{\mathcal{A}[B]} = R^{\mathcal{A}} \cap B^{\operatorname{ar}(R)}$ for every $R \in \sigma$.

The *Gaifman graph* $G_{\mathcal{A}}$ of a σ-structure $\mathcal{A}$ is the graph with vertex set A and an edge between two distinct vertices $a, b \in A$ iff there exists $R \in \sigma$ and a tuple $(a_1, \ldots, a_{\operatorname{ar}(R)}) \in R^{\mathcal{A}}$ such that $a, b \in \{a_1, \ldots, a_{\operatorname{ar}(R)}\}$. The structure $\mathcal{A}$ is *connected* if $G_{\mathcal{A}}$ is connected; the *connected components* of $\mathcal{A}$ are the connected components of $G_{\mathcal{A}}$. The *degree* of $\mathcal{A}$ is the degree of $G_{\mathcal{A}}$, i.e. the maximum number of neighbours of a vertex of $G_{\mathcal{A}}$. The *distance* $\operatorname{dist}^{\mathcal{A}}(a, b)$ between two elements $a, b \in A$ is the minimal number of edges of a path from a to b in $G_{\mathcal{A}}$; if no such path exists, we set $\operatorname{dist}^{\mathcal{A}}(a, b) := \infty$. For a tuple $\bar{a} = (a_1, \ldots, a_k) \in A^k$ and an element $b \in A$, we let $\operatorname{dist}^{\mathcal{A}}(\bar{a}, b) := \min_{i \in [k]} \operatorname{dist}(a_i, b)$, and for a tuple $\bar{b} = (b_1, \ldots, b_\ell)$, we let $\operatorname{dist}(\bar{a}, \bar{b}) := \min_{j \in [\ell]} \operatorname{dist}(\bar{a}, b_j)$.

For every $r \geqslant 0$, the *r-ball of $\bar{a}$ in $\mathcal{A}$* is the set $N_r^{\mathcal{A}}(\bar{a}) = \{b \in A : \operatorname{dist}^{\mathcal{A}}(\bar{a}, b) \leqslant r\}$. The *$r$-neighbourhood of $\bar{a}$ in $\mathcal{A}$* is the structure $\mathcal{N}_r^{\mathcal{A}}(\bar{a}) := \mathcal{A}[N_r^{\mathcal{A}}(\bar{a})]$.

3 Weight Aggregation Logic

This section introduces our new logic, which we call *first-order logic with weight aggregation*. It is inspired by the counting logic FOC and its fragment FOC_1, as introduced in [13, 8], as well as the logic $\mathrm{FO}[\mathbb{C}]$ and its fragment $\mathrm{FO}_{\mathrm{G}}[\mathbb{C}]$, which were recently introduced by Toruńczyk in [20]. Similarly as in [20], we consider *weighted structures*, which extend ordinary relational structures by assigning a weight, i.e. an element of a particular group or ring, to tuples present in the structure. The syntax and semantics of our logic, however, are closer in spirit to the syntax and semantics of the logic FOC_1, since this will enable us to achieve locality results similar to those obtained in [14, 8].

Weighted Structures. Let σ be a signature. Let $\mathbb{S}$ be a collection of rings and/or abelian groups. Let $\mathbf{W}$ be a finite set of *weight symbols*, such that each $\mathtt{w} \in \mathbf{W}$ has an associated *arity* $\operatorname{ar}(\mathtt{w}) \in \mathbb{N}_{\geqslant 1}$ and a *type* $\operatorname{type}(\mathtt{w}) \in \mathbb{S}$. A $(\sigma, \mathbf{W})$*-structure* is a σ-structure $\mathcal{A}$ that is enriched, for every $\mathtt{w} \in \mathbf{W}$, by an interpretation $\mathtt{w}^{\mathcal{A}} \colon A^{\operatorname{ar}(\mathtt{w})} \to \operatorname{type}(\mathtt{w})$, which satisfies the following *locality condition*: if $\mathtt{w}^{\mathcal{A}}(a_1, \dots, a_k) \neq 0_S$ for $S := \operatorname{type}(\mathtt{w})$, $k := \operatorname{ar}(\mathtt{w})$ and $(a_1, \dots, a_k) \in A^k$, then $k = 1$ or $a_1 = \cdots = a_k$ or there exists an $R \in \sigma$ and a tuple $(b_1, \dots, b_{\operatorname{ar}(R)}) \in R^{\mathcal{A}}$ such that $\{a_1, \dots, a_k\} \subseteq \{b_1, \dots, b_{\operatorname{ar}(R)}\}$. All notions that were introduced in Section 2 for σ-structures carry over to $(\sigma, \mathbf{W})$-structures in the obvious way. Specifically, if $\mathcal{A}$ is a $(\sigma, \mathbf{W})$-structure and σ' is a signature with $\sigma' \supseteq \sigma$, then a σ'*-expansion of* $\mathcal{A}$ is a $(\sigma', \mathbf{W})$-structure $\mathcal{B}$ with $B = A$, $R^{\mathcal{B}} = R^{\mathcal{A}}$ for all $R \in \sigma$, and $\mathtt{w}^{\mathcal{B}} = \mathtt{w}^{\mathcal{A}}$ for all $\mathtt{w} \in \mathbf{W}$.

We will use the following as running examples throughout this section.

▶ **Example 3.1.**

(a) Consider an online marketplace that allows retailers to sell their products to consumers. The database of the marketplace contains a table with transactions, and each entry consists of an identifier, a customer, a product, a retailer, the price per item, and the number of items sold. We can describe the database of the marketplace as a weighted structure as follows. Let $(\mathbb{Q}, +, \cdot)$ be the field of rationals, let $\mathbf{W}$ contain two unary weight symbols `price` and `quantity` of type $(\mathbb{Q}, +, \cdot)$, let $\sigma = \{T\}$, and let $\mathcal{A}$ be a $(\sigma, \mathbf{W})$-structure such that the universe A contains the identifiers for the transactions, customers, products, and retailers. For every transaction, let $T^{\mathcal{A}}$ contain the 4-tuple (i, c, p, r) consisting of the identifier for the transaction, the customer, the product, and the retailer. For every transaction identifier i, let $\mathtt{price}^{\mathcal{A}}(i)$ be the price per item in the transaction and let $\mathtt{quantity}^{\mathcal{A}}(i)$ be the number of items sold; for every other identifier a in A, let $\mathtt{price}^{\mathcal{A}}(a) = \mathtt{quantity}^{\mathcal{A}}(a) = 0$.

(b) In a recent survey [17], Pan and Ding describe different approaches to represent social media users via embeddings into a low-dimensional vector space, where the embeddings are based on the users' social media posts[1]. We represent the available data by a weighted structure $\mathcal{A}$ as follows. Consider the group $(\mathbb{R}^k, +)$, where $\mathbb{R}^k$ is the set of k-dimensional real vectors and $+$ is the usual vector addition, and let $\mathbf{W}$ contain a unary weight symbol `embedding` of type $(\mathbb{R}^k, +)$. Let $\sigma = \{F\}$ and let $\mathcal{A}$ be a $(\sigma, \mathbf{W})$-structure such that the universe A consists of the users of a social network. Let $F^{\mathcal{A}}$ contain all pairs of users (a, b) such that a is a follower of b. For every user $a \in A$, let $\mathtt{embedding}^{\mathcal{A}}(a)$ be a k-dimensional vector representing a's social media posts.

(c) Consider vertex-coloured edge-weighted graphs, where R, B, G are unary relations of red, blue, and green vertices, E is a binary relation of edges, and where every edge (a, b) has an associated *weight* that is a k-dimensional vector of reals (for some fixed number k). Such graphs can be viewed as $(\sigma, \mathbf{W})$-structures $\mathcal{A}$, where $\sigma = \{E, R, B, G\}$, $\mathbf{W}$ contains a binary weight symbol $\mathtt{w}$ of type $(\mathbb{R}^k, +)$ and $\mathtt{w}^{\mathcal{A}}(a, b) \in \mathbb{R}^k$ for all edges $(a, b) \in E^{\mathcal{A}}$.

Fix a countably infinite set vars of *variables.* A $(\sigma, \mathbf{W})$*-interpretation* $\mathcal{I} = (\mathcal{A}, \beta)$ consists of a $(\sigma, \mathbf{W})$-structure $\mathcal{A}$ and an *assignment* $\beta \colon \mathsf{vars} \to A$. For $k \in \mathbb{N}_{\geqslant 1}$, elements $a_1, \dots, a_k \in A$, and k distinct variables $y_1, \dots, y_k$, we write $\mathcal{I}\frac{a_1, \dots, a_k}{y_1, \dots, y_k}$ for the interpretation $(\mathcal{A}, \beta\frac{a_1, \dots, a_k}{y_1, \dots, y_k})$, where $\beta\frac{a_1, \dots, a_k}{y_1, \dots, y_k}$ is the assignment β' with $\beta'(y_i) = a_i$ for every $i \in [k]$ and $\beta'(z) = \beta(z)$ for all $z \in \mathsf{vars} \setminus \{y_1, \dots, y_k\}$.

[1] Among other applications, such embeddings might be used to predict a user's personality or political leaning.

The Weight Aggregation Logic FOWA and its Restrictions FOWA$_1$ and FOW$_1$. Let σ be a signature, $\mathbb{S}$ a collection of rings and/or abelian groups, and $\mathbf{W}$ a finite set of weight symbols. An $\mathbb{S}$-*predicate collection* is a 4-tuple $(\mathbb{P}, \mathrm{ar}, \mathrm{type}, [\![\cdot]\!])$ where $\mathbb{P}$ is a countable set of *predicate names* and, to each $\mathsf{P} \in \mathbb{P}$, ar assigns an *arity* $\mathrm{ar}(\mathsf{P}) \in \mathbb{N}_{\geqslant 1}$, type assigns a *type* $\mathrm{type}(\mathsf{P}) \in \mathbb{S}^{\mathrm{ar}(\mathsf{P})}$, and $[\![\cdot]\!]$ assigns a *semantics* $[\![\mathsf{P}]\!] \subseteq \mathrm{type}(\mathsf{P})$. For the remainder of this section, fix an $\mathbb{S}$-predicate collection $(\mathbb{P}, \mathrm{ar}, \mathrm{type}, [\![\cdot]\!])$.

For every $S \in \mathbb{S}$ that is not a ring but just an abelian group, a $\mathbf{W}$-*product of type* S is either an element $s \in S$ or an expression of the form $\mathtt{w}(y_1, \dots, y_k)$ where $\mathtt{w} \in \mathbf{W}$ is of type S, $k = \mathrm{ar}(\mathtt{w})$, and $y_1, \dots, y_k$ are k pairwise distinct variables in vars. For every ring $S \in \mathbb{S}$, a $\mathbf{W}$-*product of type* S is an expression of the form $t_1 \cdot \dots \cdot t_\ell$ where $\ell \in \mathbb{N}_{\geqslant 1}$ and for each $i \in [\ell]$ either $t_i \in S$ or there exists a $\mathtt{w} \in \mathbf{W}$ with $\mathrm{type}(\mathtt{w}) = S$ and there exist $k := \mathrm{ar}(\mathtt{w})$ pairwise distinct variables $y_1, \dots, y_k$ in vars such that $t_i = \mathtt{w}(y_1, \dots, y_k)$. By $\mathrm{vars}(p)$ we denote the set of all variables that occur in a $\mathbf{W}$-product p.

▶ **Example 3.2.** Recall Example 3.1(a)–(c), and let x and y be variables. Examples of $\mathbf{W}$-products are $\mathtt{price}(x) \cdot \mathtt{quantity}(x)$, $\mathtt{embedding}(x)$, and $\mathtt{w}(x, y)$. Below, in Definition 3.3, we will provide the formal definition of a logic (including notions of *formulas* and so-called $\mathbb{S}$-*terms*) which is capable of expressing the following statements.

(a) Given a first-order formula $\varphi_{\mathrm{group}}(p)$ that defines products of a certain product group based on the structure of their transactions, we can describe the amount of money a consumer c paid on the specified product group via the $\mathbb{S}$-*term*

$$t_{\mathrm{spending}}(c) := \sum \mathtt{price}(i) \cdot \mathtt{quantity}(i) \;.\; \exists p \, \exists r \, \big(\varphi_{\mathrm{group}}(p) \wedge T(i, c, p, r)\big).$$

This term associates with every consumer c the sum of the product of $\mathtt{price}(i)$ and $\mathtt{quantity}(i)$ for all transaction identifiers i for which there exists a product p and a retailer r such that the tuple (i, c, p, r) belongs to the transaction table and $\varphi_{\mathrm{group}}(p)$ holds. The $\mathbb{S}$-term

$$t_{\mathrm{sales}} := \sum \mathtt{price}(i) \cdot \mathtt{quantity}(i) \;.\; \exists c \, \exists p \, \exists r \, \big(\varphi_{\mathrm{group}}(p) \wedge T(i, c, p, r)\big)$$

specifies the amount *all* customers have paid on products from the product group.

We might want to select the "heavy hitters", i.e. all customers c for whom $t_{\mathrm{spending}}(c) > 0.01 \cdot t_{\mathrm{sales}}$ holds. In our logic, this is expressed by the formula

$$\mathsf{P}_{>}(t_{\mathrm{spending}}(c), 0.01 \cdot t_{\mathrm{sales}}), \quad \text{where}$$

$\mathsf{P}_{>}$ is a predicate name of type $(\mathbb{Q}, +, \cdot) \times (\mathbb{Q}, +, \cdot)$ with $[\![\mathsf{P}_{>}]\!] = \{(r, s) \in \mathbb{Q}^2 : r > s\}$.

(b) For vectors $u, v \in \mathbb{R}^k$, let $d(u, v)$ denote the Euclidean distance between u and v. We might want to use a formula $\varphi_{\mathrm{similar}}(x, y)$ expressing that the two k-dimensional vectors associated with persons x and y have Euclidean distance at most 1. To express this in our logic, we can add the rational field $(\mathbb{Q}, +, \cdot)$ to the collection $\mathbb{S}$ and use a predicate name P_{ED} of arity 3 and type $(\mathbb{R}^k, +) \times (\mathbb{R}^k, +) \times (\mathbb{Q}, +, \cdot)$ with $[\![\mathsf{P}_{\mathrm{ED}}]\!] = \{(u, v, q) \in \mathbb{R}^k \times \mathbb{R}^k \times \mathbb{Q} : d(u, v) \leqslant q\}$. Then,

$$\varphi_{\mathrm{similar}}(x, y) := \mathsf{P}_{\mathrm{ED}}(\mathtt{embedding}(x), \mathtt{embedding}(y), 1)$$

is a formula with the desired meaning.

(c) For each vertex x, the sum of the weights of edges between x and its blue neighbours is specified by the $\mathbb{S}$-term $t_B(x) := \sum \mathtt{w}(x', y).(x' {=} x \wedge E(x', y) \wedge B(y))$.

We have designed the definition of the syntax of our logic in a way particularly suitable for formulating and proving the locality results that are crucial for obtaining our learning results. To obtain a more user-friendly syntax, i.e. which allows to read and construct formulas in a more intuitive way, one could of course introduce syntactic sugar that allows to explicitly write statements of the form

- $t_{\text{spending}}(c) > 0.01 \cdot t_{\text{sales}}$ instead of $\mathsf{P}_{>}(t_{\text{spending}}(c), 0.01 \cdot t_{\text{sales}})$
- $d(\text{embedding}(x), \text{embedding}(y)) \leqslant 1$ instead of $\mathsf{P}_{\text{ED}}(\text{embedding}(x), \text{embedding}(y), 1)$
- $\sum\limits_{y} \mathtt{w}(x,y).(E(x,y) \wedge B(y))$ instead of $\sum \mathtt{w}(x',y).(x'{=}x \wedge E(x',y) \wedge B(y))$.

We now define the precise syntax and semantics of our weight aggregation logic.

▶ **Definition 3.3** (FOWA($\mathbb{P}$)[$\sigma, \mathbb{S}, \mathbf{W}$]). *For* $\text{FOWA}(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*, the set of* formulas *and* $\mathbb{S}$-terms *is built according to the following rules:*

(1) $x_1{=}x_2$ *and* $R(x_1, \ldots, x_{\text{ar}(R)})$ *are* formulas,
where $R \in \sigma$ *and* $x_1, x_2, \ldots, x_{\text{ar}(R)}$ *are variables*[2].

(2) *If* $\mathtt{w} \in \mathbf{W}$, $S = \text{type}(\mathtt{w})$, $s \in S$, $k = \text{ar}(\mathtt{w})$, *and* $\bar{x} = (x_1, \ldots, x_k)$ *is a tuple of* k *pairwise distinct variables, then* $(s = \mathtt{w}(\bar{x}))$ *is a* formula.

(3) *If* φ *and* ψ *are formulas, then* $\neg\varphi$ *and* $(\varphi \vee \psi)$ *are also* formulas.

(4) *If* φ *is a formula and* $y \in \mathsf{vars}$*, then* $\exists y\, \varphi$ *is a* formula.

(5) *If* φ *is a formula,* $\mathtt{w} \in \mathbf{W}$, $S = \text{type}(\mathtt{w})$, $s \in S$, $k = \text{ar}(\mathtt{w})$, *and* $\bar{y} = (y_1, \ldots, y_k)$ *is a tuple of* k *pairwise distinct variables, then* $\big(s = \sum \mathtt{w}(\bar{y}).\varphi\big)$ *is a* formula.

(6) *If* $\mathsf{P} \in \mathbb{P}$, $m = \text{ar}(\mathsf{P})$, *and* $t_1, \ldots, t_m$ *are* $\mathbb{S}$*-terms such that* $(\text{type}(t_1), \ldots, \text{type}(t_m)) = \text{type}(\mathsf{P})$*, then* $\mathsf{P}(t_1, \ldots, t_m)$ *is a* formula.

(7) *For every* $S \in \mathbb{S}$ *and every* $s \in S$*,* s *is an* $\mathbb{S}$-term *of type* S.

(8) *For every* $S \in \mathbb{S}$*, every* $\mathtt{w} \in \mathbf{W}$ *of type* S*, and every tuple* $(x_1, \ldots, x_k)$ *of* $k := \text{ar}(\mathtt{w})$ *pairwise distinct variables in* vars*,* $\mathtt{w}(x_1, \ldots, x_k)$ *is an* $\mathbb{S}$-term *of type* S.

(9) *If* t_1 *and* t_2 *are* $\mathbb{S}$*-terms of the same type* S*, then so are* $(t_1{+}t_2)$ *and* $(t_1{-}t_2)$*; furthermore, if* S *is a ring (and not just an abelian group), then also* $(t_1{\cdot}t_2)$ *is an* $\mathbb{S}$*-term of type* S.

(10) *If* φ *is a formula,* $S \in \mathbb{S}$*, and* p *is a* $\mathbf{W}$*-product of type* S*, then* $\sum p.\varphi$ *is an* $\mathbb{S}$-term *of type* S.

Let $\mathcal{I} = (\mathcal{A}, \beta)$ *be a* $(\sigma, \mathbf{W})$*-interpretation. For a formula or* $\mathbb{S}$*-term* ξ *of* $\text{FOWA}(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*, the semantics* $[\![\xi]\!]^{\mathcal{I}}$ *is defined as follows.*

(1) $[\![x_1{=}x_2]\!]^{\mathcal{I}} = 1$ *if* $a_1{=}a_2$*, and* $[\![x_1{=}x_2]\!]^{\mathcal{I}} = 0$ *otherwise;*
$[\![R(x_1, \ldots, x_{\text{ar}(R)})]\!]^{\mathcal{I}} = 1$ *if* $(a_1, \ldots, a_{\text{ar}(R)}) \in R^{\mathcal{A}}$*, and* $[\![R(x_1, \ldots, x_{\text{ar}(R)})]\!]^{\mathcal{I}} = 0$ *otherwise;*
where $a_j := \beta(x_j)$ *for* $j \in \{1, \ldots, \max\{2, \text{ar}(R)\}\}$.

(2) $[\![(s = \mathtt{w}(\bar{x}))]\!]^{\mathcal{I}} = 1$ *if* $s = \mathtt{w}^{\mathcal{A}}(\beta(x_1), \ldots, \beta(x_k))$*, and* $[\![(s = \mathtt{w}(\bar{x}))]\!]^{\mathcal{I}} = 0$ *otherwise.*

(3) $[\![\neg\varphi]\!]^{\mathcal{I}} = 1 - [\![\varphi]\!]^{\mathcal{I}}$ *and* $[\![(\varphi \vee \psi)]\!]^{\mathcal{I}} = \max\{[\![\varphi]\!]^{\mathcal{I}}, [\![\psi]\!]^{\mathcal{I}}\}$.

(4) $[\![\exists y\, \varphi]\!]^{\mathcal{I}} = \max\{[\![\varphi]\!]^{\mathcal{I}\frac{a}{y}} : a \in A\}$.

(5) $[\![\big(s = \sum \mathtt{w}(\bar{y}).\varphi\big)]\!]^{\mathcal{I}} = 1$ *if* $s = \sum_S \{\mathtt{w}^{\mathcal{A}}(\bar{a}) : \bar{a} = (a_1, \ldots, a_k) \in A^k$ *with* $[\![\varphi]\!]^{\mathcal{I}\frac{a_1, \ldots, a_k}{y_1, \ldots, y_k}} = 1\}$
(as usual, by convention, we let $\sum_S X = 0_S$ *if* $X = \emptyset$*).*

(6) $[\![\mathsf{P}(t_1, \ldots, t_m)]\!]^{\mathcal{I}} = 1$ *if* $\big([\![t_1]\!]^{\mathcal{I}}, \ldots, [\![t_m]\!]^{\mathcal{I}}\big) \in [\![\mathsf{P}]\!]$*, and* $[\![\mathsf{P}(t_1, \ldots, t_m)]\!]^{\mathcal{I}} = 0$ *otherwise.*

(7) $[\![s]\!]^{\mathcal{I}} = s$.

(8) $[\![\mathtt{w}(x_1, \ldots, x_k)]\!]^{\mathcal{I}} = \mathtt{w}^{\mathcal{A}}(\beta(x_1), \ldots, \beta(x_k))$.

(9) $[\![(t_1 * t_2)]\!]^{\mathcal{I}} = [\![t_1]\!]^{\mathcal{I}} *_S [\![t_2]\!]^{\mathcal{I}}$*, for* $* \in \{+, -, \cdot\}$.

[2] In particular, if $\text{ar}(R) = 0$, then $R()$ is a formula.

(10) $[\![\sum p.\varphi]\!]^{\mathcal{I}} = \sum_S \{ [\![p]\!]^{\mathcal{I}\frac{a_1,\dots,a_k}{y_1,\dots,y_k}} : a_1,\dots,a_k \in A$ *with* $[\![\varphi]\!]^{\mathcal{I}\frac{a_1,\dots,a_k}{y_1,\dots,y_k}} = 1\}$, *where* $\{y_1,\dots,y_k\} = \mathrm{vars}(p)$ *and* $k = |\mathrm{vars}(p)|$ *and* $[\![p]\!]^{\mathcal{I}} = [\![t_1]\!]^{\mathcal{I}} \cdot_S \cdots \cdot_S [\![t_\ell]\!]^{\mathcal{I}}$ *if* $p = t_1 \cdot \cdots \cdot t_\ell$ *is of type* S*.*

An *expression* is a formula or an $\mathbb{S}$-term. As usual, for a formula φ and a $(\sigma,\mathbf{W})$-interpretation $\mathcal{I}$, we will often write $\mathcal{I} \models \varphi$ to indicate that $[\![\varphi]\!]^{\mathcal{I}} = 1$. Accordingly, $\mathcal{I} \not\models \varphi$ indicates that $[\![\varphi]\!]^{\mathcal{I}} = 0$.

The set $\mathrm{vars}(\xi)$ of an expression ξ is defined as the set of all variables in vars that occur in ξ. The *free variables* $\mathrm{free}(\xi)$ of ξ are defined as follows: $\mathrm{free}(\xi) = \mathrm{vars}(\xi)$ if ξ is built according to one of the rules (1), (2), (7), (8); $\mathrm{free}(\neg\varphi) = \mathrm{free}(\varphi)$, $\mathrm{free}((\varphi \vee \psi)) = \mathrm{free}(\varphi) \cup \mathrm{free}(\psi)$, $\mathrm{free}(\exists y\,\varphi) = \mathrm{free}(\varphi) \setminus \{y\}$, $\mathrm{free}((s = \sum \mathtt{w}(y_1,\dots,y_k).\varphi)) = \mathrm{free}(\varphi) \setminus \{y_1,\dots,y_k\}$; $\mathrm{free}(\mathsf{P}(t_1,\dots,t_m)) = \bigcup_{i=1}^m \mathrm{free}(t_i)$; $\mathrm{free}((t_1 * t_2)) = \mathrm{free}(t_1) \cup \mathrm{free}(t_2)$ for $* \in \{+,-,\cdot\}$; $\mathrm{free}(\sum p.\varphi) = \mathrm{free}(\varphi) \setminus \mathrm{vars}(p)$. As usual, we will write $\xi(\bar{x})$ for $\bar{x} = (x_1,\dots,x_k)$ to indicate that $\mathrm{free}(\xi) \subseteq \{x_1,\dots,x_k\}$. A *sentence* is a $\mathrm{FOWA}(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$-formula φ with $\mathrm{free}(\varphi) = \emptyset$. A *ground* $\mathbb{S}$*-term* is an $\mathbb{S}$-term t of $\mathrm{FOWA}(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$ with $\mathrm{free}(t) = \emptyset$.

For a $(\sigma,\mathbf{W})$-structure $\mathcal{A}$ and a tuple $\bar{a} = (a_1,\dots,a_k) \in A^k$, we write $\mathcal{A} \models \varphi[\bar{a}]$ or $(\mathcal{A},\bar{a}) \models \varphi$ to indicate that for every assignment $\beta\colon \mathsf{vars} \to A$ with $\beta(x_i) = a_i$ for all $i \in [k]$, we have $\mathcal{I} \models \varphi$, for $\mathcal{I} = (\mathcal{A},\beta)$. Similarly, for an $\mathbb{S}$-term $t(\bar{x})$ we write $t^{\mathcal{A}}[\bar{a}]$ to denote $[\![t]\!]^{\mathcal{I}}$.

▶ **Definition 3.4** (FOWA$_1$ and FOW$_1$)**.** *The set of* formulas *and* $\mathbb{S}$-terms *of the logic* $\mathrm{FOWA}_1(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$ *is built according to the same rules as for the logic* $\mathrm{FOWA}(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$*, with the following restrictions:*
$(5)_1$*: rule* (5) *can only be applied if* S *is* finite*,*
$(6)_1$*: rule* (6) *can only be applied if* $|\mathrm{free}(t_1) \cup \cdots \cup \mathrm{free}(t_m)| \leqslant 1$*.*
$\mathrm{FOW}_1(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$ *is the restriction of* $\mathrm{FOWA}_1(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$ *where rule* (10) *cannot be applied.*

Note that first-order logic $\mathrm{FO}[\sigma]$ is the restriction of $\mathrm{FOW}_1(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$ where only rules (1), (3), and (4) can be applied. As usual, we write $(\varphi \wedge \psi)$ and $\forall y\,\varphi$ as shorthands for $\neg(\neg\varphi \vee \neg\psi)$ and $\neg\exists y\,\neg\varphi$. The *quantifier rank* $\mathrm{qr}(\xi)$ of a $\mathrm{FOWA}(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$-expression ξ is defined as the maximum nesting depth of constructs using rules (4) and (5) in order to construct ξ. The *aggregation depth* $\mathrm{d_{ag}}(\xi)$ of ξ is defined as the maximum nesting depth of term constructions using rule (10) in order to construct ξ.

▶ Remark 3.5. FOW$_1$ can be viewed as an extension of first-order logic with modulo-counting quantifiers: if $\mathbb{S}$ contains the abelian group $(\mathbb{Z}/m\mathbb{Z},+)$ for some $m \geqslant 2$, and $\mathbf{W}$ contains a unary weight symbol $\mathtt{one}_m$ of type $\mathbb{Z}/m\mathbb{Z}$ such that $\mathtt{one}_m^{\mathcal{A}}(a) = 1$ for all $a \in A$, then the modulo m counting quantifier $\exists^{i \bmod m} y\,\varphi$ (stating that the number of interpretations for y that satisfy φ is congruent to i modulo m) can be expressed in $\mathrm{FOW}_1(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$ via $\big(i = \sum \mathtt{one}_m(y).\varphi\big)$.

FOWA$_1$ can be viewed as an extension of the logic FOC$_1$ of [8]: if $\mathbb{S}$ contains the integer ring $(\mathbb{Z},+,\cdot)$ and $\mathbf{W}$ contains a unary weight symbol $\mathtt{one}$ of type $\mathbb{Z}$ such that $\mathtt{one}^{\mathcal{A}}(a) = 1$ for all $a \in A$ on all considered $(\sigma,\mathbf{W})$-structures $\mathcal{A}$, then the counting term $\#(y_1,\dots,y_k).\varphi$ of FOC$_1$ (which counts the number of tuples $(y_1,\dots,y_k)$ that satisfy φ) can be expressed in $\mathrm{FOWA}_1(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$ via the $\mathbb{S}$-term $\sum p.\varphi$ for $p := \mathtt{one}(y_1) \cdot \cdots \cdot \mathtt{one}(y_k)$.

Let us mention, again, that we have designed the precise definition of the syntax of our logic in a way particularly suitable for formulating and proving the locality results that are crucial for obtaining our learning results. To obtain a more user-friendly syntax, i.e. which allows to read and construct formulas in a more intuitive way, it would of course make sense to introduce syntactic sugar that allows to explicitly write statements of the form

- $\#(y_1,\dots,y_k).\varphi$ instead of $\sum p.\varphi$ for $p := \mathtt{one}(y_1) \cdot \cdots \cdot \mathtt{one}(y_k)$
- $\big(\#(y).\varphi \equiv i \bmod m\big)$ or $\exists^{i \bmod m} y\,\varphi$ instead of $\big(i = \sum \mathtt{one}_m(y).\varphi\big)$.

For this, one would tacitly assume that $\mathbb{S}$ contains $(\mathbb{Z},+,\cdot)$ (or $(\mathbb{Z}/m\mathbb{Z},+)$) and $\mathbf{W}$ contains a unary weight symbol $\texttt{one}$ of type $\mathbb{Z}$ (or $\texttt{one}_m$ of type $\mathbb{Z}/m\mathbb{Z}$) where $\texttt{one}^{\mathcal{A}}(a)=1$ $(=\texttt{one}_m^{\mathcal{A}}(a))$ for every $a\in A$ and every considered $(\sigma,\mathbf{W})$-structure $\mathcal{A}$.

To close this section, we return to the running examples from Examples 3.1 and 3.2.

▶ **Example 3.6.** We use the syntactic sugar introduced at the end of Remark 3.5.

(a) The number of consumers who bought products p from the product group defined by $\varphi_{\text{group}}(p)$ is specified by the $\mathbb{S}$-term

$$t_{\#\text{cons}} := \sum \texttt{one}(c)\ .\ \exists i\,\exists p\,\exists r\,(\varphi_{\text{group}}(p)\wedge T(i,c,p,r));$$

and using the syntactic sugar described above, this $\mathbb{S}$-term can be expressed via $\#(c).\ \exists i\,\exists p\,\exists r\,(\varphi_{\text{group}}(p)\wedge T(i,c,p,r))$.
The consumers c who spent at least as much as the *average consumer* on the products p satisfying $\varphi_{\text{group}}(p)$ can be described by the formula

$$\varphi_{\text{spending}}(c) := \ \mathsf{P}_{\geqslant}\big((t_{\text{spending}}(c)\cdot t_{\#\text{cons}})\ ,\ t_{\text{sales}}\big),$$

where $\mathsf{P}_{\geqslant}$ is a binary predicate in $\mathbb{P}$ of type $\mathbb{Q}\times\mathbb{Q}$ that is interpreted by the $\geqslant$-relation. To improve readability, one could introduce syntactic sugar that allows to express this as $t_{\text{spending}}(c)\geqslant t_{\text{sales}}/t_{\#\text{cons}}$. The formula $\varphi_{\text{spending}}(c)$ belongs to $\text{FOWA}_1(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$.

(b) The term $t_{\#\text{follows}}(x) := \#(y).F(x,y)$ specifies the number of users y followed by person x. The term $t_{\text{sum}}(x) := \sum \texttt{embedding(y)}.F(x,y)$ specifies the sum of the vectors associated with all users y followed by x. To describe the users x whose embedding is δ-close (for some fixed $\delta>0$) to the average of the embeddings of users they follow[3], we might want to use a formula $\varphi_{\text{close}}(x)$ of the form

$$d\,\big(\,\texttt{embedding}(x)\ ,\ \tfrac{1}{t_{\#\text{follows}}(x)}\cdot t_{\text{sum}}(x)\,\big)\ <\ \delta\,.$$

We can describe this in $\text{FOWA}_1(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$ by the formula

$$\varphi_{\text{close}}(x)\ :=\ \mathsf{P}_{\text{dist}<\delta}(\texttt{embedding}(x), t_{\#\text{follows}}(x), t_{\text{sum}}(x)),$$

where $\mathsf{P}_{\text{dist}<\delta}$ is a ternary predicate in $\mathbb{P}$ of type $\mathbb{R}^k\times\mathbb{Z}\times\mathbb{R}^k$ consisting of all triples $(\bar v,\ell,\bar w)$ with $\ell>0$ and $d(\bar v,\frac{1}{\ell}\cdot\bar w)<\delta$.

(c) Recall the term $t_B(x)$ introduced in Example 3.2 (c) that specifies the sum of the weights of edges between x and its blue neighbours, and let $t_R(x)$ be a similar term summing up the weights of edges between x and its red neighbours (using the syntactic sugar introduced at the end of Example 3.2, this can be described as $\sum\limits_{y}\texttt{w}(x,y).(E(x,y)\wedge R(y))$).
To specify the vertices x that have exactly 5 red neighbours, we can use the formula $\varphi_{5\,\text{red}}(x) := (\,5=\#(y).(E(x,y)\wedge R(y))\,)$. Let us now assume we are given a particular set $H\subseteq\mathbb{R}^{2k}$ and we want to specify the vertices x that have exactly 5 red neighbours and for which, in addition, the $2k$-ary vector obtained by concatenating the k-ary vectors computed by summing up the weights of edges between x and its blue neighbours and by summing up the weights of edges between x and its red neighbours belongs to H. To express this, we can use a binary predicate P of type $\mathbb{R}^k\times\mathbb{R}^k$ with $[\![\mathsf{P}]\!]=\big\{(\bar u,\bar v)\in\mathbb{R}^k\times\mathbb{R}^k\ :\ (u_1,\dots,u_k,v_1,\dots,v_k)\in H\big\}$. Then, the $\text{FOWA}_1(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$-formula $\psi(x) := \varphi_{5\,\text{red}}(x)\wedge\mathsf{P}(t_B(x),t_G(x))$ specifies the vertices x we are interested in.

[3] Depending on the target of the embeddings, this could mean that the user mostly follows users with a very similar personality or political leaning.

4 Locality Properties of FOW_1 and FOWA_1

We now summarise locality properties of FOW_1 and FOWA_1 that are similar to well-known locality properties of first-order logic FO and to locality properties of FOC_1 achieved in [8]. This includes *Feferman-Vaught decompositions* and a *Gaifman normal form* for FOW_1, and a *localisation theorem* for the more expressive logic FOWA_1.

For the remainder of this section, let us fix a signature σ, a collection $\mathbb{S}$ of rings and/or abelian groups, a finite set $\mathbf{W}$ of weight symbols, and an $\mathbb{S}$-predicate collection $(\mathbb{P}, \mathrm{ar}, \mathrm{type}, [\![\cdot]\!])$.

The notion of *local formulas* is defined as usual [15]: let $r \in \mathbb{N}$. A $\mathrm{FOWA}(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$-formula $\varphi(\bar{x})$ with free variables $\bar{x} = (x_1, \dots, x_k)$ is *r-local (around $\bar{x}$)* if for every $(\sigma, \mathbf{W})$-structure $\mathcal{A}$ and all $\bar{a} \in A^k$, we have $\mathcal{A} \models \varphi[\bar{a}] \iff \mathcal{N}_r^{\mathcal{A}}(\bar{a}) \models \varphi[\bar{a}]$. A formula is *local* if it is r-local for some $r \in \mathbb{N}$.

For an $r \in \mathbb{N}$, it is straightforward to construct an $\mathrm{FO}[\sigma]$-formula $\mathrm{dist}^{\sigma}_{\leqslant r}(x, y)$ such that for every $(\sigma, \mathbf{W})$-structure $\mathcal{A}$ and all $a, b \in A$, we have $\mathcal{A} \models \mathrm{dist}^{\sigma}_{\leqslant r}[a, b] \iff \mathrm{dist}^{\mathcal{A}}(a, b) \leqslant r$. To improve readability, we write $\mathrm{dist}^{\sigma}(x, y) \leqslant r$ for $\mathrm{dist}^{\sigma}_{\leqslant r}(x, y)$, and $\mathrm{dist}^{\sigma}(x, y) > r$ for $\neg\, \mathrm{dist}^{\sigma}_{\leqslant r}(x, y)$; and we omit the superscript σ when it is clear from the context. For a tuple $\bar{x} = (x_1, \dots, x_k)$ of variables, $\mathrm{dist}(\bar{x}, y) > r$ is a shorthand for $\bigwedge_{i=1}^{k} \mathrm{dist}(x_i, y) > r$, and $\mathrm{dist}(\bar{x}, y) \leqslant r$ is a shorthand for $\bigvee_{i=1}^{k} \mathrm{dist}(x_i, y) \leqslant r$. For $\bar{y} = (y_1, \dots, y_\ell)$, we use $\mathrm{dist}(\bar{x}; \bar{y}) > r$ and $\mathrm{dist}(\bar{x}; \bar{y}) \leqslant r$ as shorthands for $\bigwedge_{j=1}^{\ell} \mathrm{dist}(\bar{x}, y_j) > r$ and $\bigvee_{j=1}^{\ell} \mathrm{dist}(\bar{x}, y_j) \leqslant r$, respectively.

The *r-localisation* $\varphi^{(r)}$ of a $\mathrm{FOWA}(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$-formula $\varphi(\bar{x})$ is the formula obtained from φ by replacing every subformula of the form $\exists y\, \varphi'$ with the formula $\exists y\, \big(\varphi' \wedge \mathrm{dist}(\bar{x}, y) \leqslant r\big)$, replacing every subformula of the form $\big(s = \sum \mathtt{w}(\bar{y}).\varphi'\big)$, for $\bar{y} = (y_1, \dots, y_k)$, with the formula $\big(s = \sum \mathtt{w}(\bar{y}).(\varphi' \wedge \bigwedge_{j=1}^{k} \mathrm{dist}(\bar{x}, y_j) \leqslant r)\big)$, and replacing every $\mathbb{S}$-term of the form $\sum p.\varphi'$ with the $\mathbb{S}$-term $\sum p.\big(\varphi' \wedge \bigwedge_{j=1}^{k} \mathrm{dist}(\bar{x}, y_j) \leqslant r\big)$, where $\{y_1, \dots, y_k\} = \mathrm{free}(\varphi')$. The resulting formula $\varphi^{(r)}(\bar{x})$ is r-local.

Feferman-Vaught Decompositions for FOW_1. We pick two new unary relation symbols X, Y that do not belong to σ, and we let $\sigma' := \sigma \cup \{X, Y\}$.

▶ **Definition 4.1.** *Let $\mathcal{A}, \mathcal{B}$ be $(\sigma, \mathbf{W})$-structures with $A \cap B = \emptyset$. The* disjoint sum $\mathcal{A} \oplus \mathcal{B}$ *is the $(\sigma', \mathbf{W})$-structure $\mathcal{C}$ with universe $C = A \cup B$, $X^{\mathcal{C}} = A$, $Y^{\mathcal{C}} = B$, $R^{\mathcal{C}} = R^{\mathcal{A}} \cup R^{\mathcal{B}}$ for all $R \in \sigma$, and such that for all $\mathtt{w} \in \mathbf{W}$ and $k := \mathrm{ar}(\mathtt{w})$ and all $\bar{c} = (c_1, \dots, c_k) \in C^k$, we have $\mathtt{w}^{\mathcal{C}}(\bar{c}) = \mathtt{w}^{\mathcal{A}}(\bar{c})$ if $\bar{c} \in A^k$, $\mathtt{w}^{\mathcal{C}}(\bar{c}) = \mathtt{w}^{\mathcal{B}}(\bar{c})$ if $\bar{c} \in B^k$, and $\mathtt{w}^{\mathcal{C}}(\bar{c}) = 0_S$ otherwise (for $S := \mathrm{type}(\mathtt{w})$). The* disjoint union $\mathcal{A} \sqcup \mathcal{B}$ *is the $(\sigma, \mathbf{W})$-structure obtained from $\mathcal{C} := \mathcal{A} \oplus \mathcal{B}$ by omitting the relations $X^{\mathcal{C}}, Y^{\mathcal{C}}$.*

▶ **Definition 4.2.** *Let L be a subset of $\mathrm{FOWA}(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$.
Let $k, \ell \in \mathbb{N}$ and let $\bar{x} = (x_1, \dots, x_k)$, $\bar{y} = (y_1, \dots, y_\ell)$ be tuples of $k{+}\ell$ pairwise distinct variables. Let φ be a $\mathrm{FOWA}(\mathbb{P})[\sigma', \mathbb{S}, \mathbf{W}]$-formula with $\mathrm{free}(\varphi) \subseteq \{x_1, \dots, x_k, y_1, \dots, y_\ell\}$. A* Feferman-Vaught decomposition *of φ in L w.r.t. $(\bar{x}; \bar{y})$ is a finite, non-empty set Δ of tuples of the form (α, β) where $\alpha, \beta \in \mathrm{L}$ and $\mathrm{free}(\alpha) \subseteq \{x_1, \dots, x_k\}$ and $\mathrm{free}(\beta) \subseteq \{y_1, \dots, y_\ell\}$, such that the following is true for all $(\sigma, \mathbf{W})$-structures $\mathcal{A}, \mathcal{B}$ with $A \cap B = \emptyset$ and all $\bar{a} \in A^k$, $\bar{b} \in B^\ell$: $\mathcal{A} \oplus \mathcal{B} \models \varphi[\bar{a}, \bar{b}] \iff$ there exists $(\alpha, \beta) \in \Delta$ such that $\mathcal{A} \models \alpha[\bar{a}]$ and $\mathcal{B} \models \beta[\bar{b}]$.*

Our first main result provides Feferman-Vaught decompositions for FOW_1.

▶ **Theorem 4.3** (Feferman-Vaught decompositions for $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$).
Let $k, \ell \in \mathbb{N}$ and let $\bar{x} = (x_1, \dots, x_k)$, $\bar{y} = (y_1, \dots, y_\ell)$ be tuples of $k{+}\ell$ pairwise distinct variables. For every $\mathrm{FOW}_1(\mathbb{P})[\sigma', \mathbb{S}, \mathbf{W}]$-formula φ with $\mathrm{free}(\varphi) \subseteq \{x_1, \dots, x_k, y_1, \dots, y_\ell\}$, there exists a Feferman-Vaught decomposition Δ in L of φ w.r.t. $(\bar{x}; \bar{y})$, where $\mathrm{L} := \mathrm{L}_\varphi$ is

the class of all $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*-formulas of quantifier rank at most* $\mathrm{qr}(\varphi)$ *which use only those* $\mathsf{P} \in \mathbb{P}$ *and* $S \in \mathbb{S}$ *that occur in* φ *and only those* $\mathbb{S}$*-terms that occur in* φ *or that are of the form* s *for an* $s \in S \in \mathbb{S}$ *where* S *is finite and occurs in* φ*.*

Furthermore, there is an algorithm that computes Δ *upon input of* $\varphi, \bar{x}, \bar{y}$*.*

The proof proceeds in a similar way as the proof of the Feferman-Vaught decomposition for first-order logic with modulo-counting quantifiers in [14]. For details as well as for the proof of the following corollary of the theorem, we refer to [23].

▶ **Corollary 4.4.** *Let* $k, \ell \in \mathbb{N}$ *and let* $\bar{x} = (x_1, \ldots, x_k)$*,* $\bar{y} = (y_1, \ldots, y_\ell)$ *be tuples of* $k+\ell$ *pairwise distinct variables. Upon input of an* $r \in \mathbb{N}$ *and an* r*-local* $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*-formula* $\varphi(\bar{x}, \bar{y})$*, one can compute a finite, non-empty set* Δ *of pairs* $\big(\alpha(\bar{x}), \beta(\bar{y})\big)$ *of* L*-formulas, where* L *is the class of all* r*-localisations of formulas in the class* L_φ *of Theorem 4.3, such that the following two formulas are equivalent:*

- $\Big(\bigwedge_{i=1}^{k} \bigwedge_{j=1}^{\ell} \mathrm{dist}(x_i, y_j) > 2r{+}1\Big) \ \wedge \ \varphi(\bar{x}, \bar{y})$
- $\Big(\bigwedge_{i=1}^{k} \bigwedge_{j=1}^{\ell} \mathrm{dist}(x_i, y_j) > 2r{+}1\Big) \ \wedge \ \bigvee_{(\alpha,\beta)\in\Delta} \big(\alpha(\bar{x}) \wedge \beta(\bar{y})\big).$

Gaifman Normal Form for $\mathbf{FOW_1}$. We now turn to a Gaifman normal form for FOW_1.

▶ **Definition 4.5.** *A* basic-local sentence *in* $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$ *is a sentence of the form* $\exists x_1 \cdots \exists x_\ell \big(\bigwedge_{1 \leqslant i < j \leqslant \ell} \mathrm{dist}(x_i, x_j) > 2r \ \wedge \ \bigwedge_{i=1}^{\ell} \lambda(x_i)\big)$*, where* $\ell \in \mathbb{N}_{\geqslant 1}$*,* $r \in \mathbb{N}$*,* $\lambda(x)$ *is an* r*-local* $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*-formula, and* $x_1, \ldots, x_\ell$ *are* ℓ *pairwise distinct variables.*

A local aggregation sentence *in* $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$ *is a sentence of the form* $\big(s = \sum \mathtt{w}(\bar{y}).\lambda(\bar{y})\big)$*, where* $\mathtt{w} \in \mathbf{W}$*,* $s \in S := \mathrm{type}(\mathtt{w})$*,* $\ell = \mathrm{ar}(\mathtt{w})$*,* $\bar{y} = (y_1, \ldots, y_\ell)$ *is a tuple of* ℓ *pairwise distinct variables, and* $\lambda(\bar{y})$ *is an* r*-local* $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*-formula.*

A $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*-formula* in Gaifman normal form *is a Boolean combination of local* $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*-formulas, basic-local sentences in* $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*, and local aggregation sentences in* $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*.*

Our next main theorem provides a Gaifman normal form for FOW_1.

▶ **Theorem 4.6** (Gaifman normal form for $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$)**.** *Every* $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*-formula* φ *is equivalent to an* $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*-formula* γ *in Gaifman normal form with* $\mathrm{free}(\gamma) = \mathrm{free}(\varphi)$*. Furthermore, there is an algorithm that computes* γ *upon input of* φ*.*

The proof proceeds similarly as Gaifman's original proof for first-order logic FO ([2], see also [4, Sect. 4.1]), but since subformulas are from $\mathrm{FOW}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$, we use Corollary 4.4 instead of Feferman-Vaught decompositions for FO (cf. [4, Lemma 2.3]). Furthermore, for formulas built according to rule $(5)_1$, we proceed in a similar way as for the modulo-counting quantifiers in the Gaifman normal construction of [14]. We defer the reader to the full version for the details [23].

A Localisation Theorem for $\mathbf{FOWA_1}$. Our next main theorem provides a locality result for the logic FOWA_1, which is a logic substantially more expressive than FOW_1.

▶ **Theorem 4.7** (Localisation Theorem for FOWA_1)**.** *For every* $\mathrm{FOWA}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$*-formula* $\varphi(x_1, \ldots, x_k)$ *(with* $k \geqslant 0$*), there is an extension* σ_φ *of* σ *with relation symbols of arity* $\leqslant 1$*, and a* $\mathrm{FOW}_1(\mathbb{P})[\sigma_\varphi, \mathbb{S}, \mathbf{W}]$*-formula* $\varphi'(x_1, \ldots, x_k)$ *that is a Boolean combination of local formulas and statements of the form* $R()$ *where* $R \in \sigma_\varphi$ *has arity* 0*, for which the following* *is true: there is an algorithm*[4] *that, upon input of a* $(\sigma, \mathbf{W})$*-structure* $\mathcal{A}$*, computes in time* $|A| \cdot d^{\mathcal{O}(1)}$*, where* d *is the degree of* $\mathcal{A}$*, a* σ_φ*-expansion* $\mathcal{A}^\varphi$ *of* $\mathcal{A}$ *such that for all* $\bar{a} \in A^k$ *it holds that* $\mathcal{A}^\varphi \models \varphi'[\bar{a}] \iff \mathcal{A} \models \varphi[\bar{a}]$*.*

We prove this by decomposing FOWA_1-expressions into simpler expressions that can be evaluated in a structure $\mathcal{A}$ by exploring for each element a in the universe of $\mathcal{A}$ only a local neighbourhood around a. This is achieved by proving a decomposition theorem that is a generalisation of the decomposition for $\mathrm{FOC}_1(\mathbb{P})$ provided in [8, Theorem 6.6], and it builds upon the Gaifman normal form result of Theorem 4.6. Again, we defer the reader to the full version for the details [23].

5 Learning Concepts on Weighted Structures

Throughout this section, fix a collection $\mathbb{S}$ of rings and/or abelian groups, an $\mathbb{S}$-predicate collection $(\mathbb{P}, \mathrm{ar}, \mathrm{type}, [\![\cdot]\!])$, and a finite set $\mathbf{W}$ of weight symbols.

Furthermore, fix numbers $k, \ell \in \mathbb{N}$. Let L be a logic (e.g. FO, $\mathrm{FOW}_1(\mathbb{P})$, $\mathrm{FOWA}_1(\mathbb{P})$, $\mathrm{FOWA}(\mathbb{P})$), let σ be a signature, and let $\Phi \subseteq \mathrm{L}[\sigma, \mathbb{S}, \mathbf{W}]$ be a set of formulas $\varphi(\bar{x}, \bar{y})$ with $|\bar{x}| = k$ and $|\bar{y}| = \ell$. For a $(\sigma, \mathbf{W})$-structure $\mathcal{A}$, we follow the same approach as [3, 6, 7, 9, 22] and consider the instance space $\mathcal{X} = A^k$ and concepts from the concept class

$$\mathcal{C}(\Phi, \mathcal{A}, k, \ell) \;\coloneqq\; \{\; [\![\varphi(\bar{x}, \bar{y})]\!]^{\mathcal{A}}(\bar{x}, \bar{v}) \;:\; \varphi \in \Phi,\; \bar{v} \in A^\ell \;\},$$

where $[\![\varphi(\bar{x}, \bar{y})]\!]^{\mathcal{A}}(\bar{x}, \bar{v})$ is defined as the mapping from A^k to $\{0,1\}$ that maps $\bar{a} \in A^k$ to $[\![\varphi(\bar{a}, \bar{v})]\!]^{\mathcal{A}}$, which is 1 iff $\mathcal{A} \models \varphi[\bar{a}, \bar{v}]$. Given a training sequence $T = \big((\bar{a}_1, b_1), \ldots, (\bar{a}_t, b_t)\big)$ from $(A^k \times \{0,1\})^t$, we want to compute a hypothesis that consists of a formula φ and a tuple of parameters $\bar{v}$ and is, depending on the approach, consistent with the training sequence or probably approximately correct.

Instead of allowing random access to the background structure, we limit our algorithms to have only *local access*. That is, an algorithm may only interact with the structure via queries of the form "Is $\bar{a} \in R^{\mathcal{A}}$?", "Return $\mathtt{w}^{\mathcal{A}}(\bar{a})$" and "Return a list of all neighbours of a in the Gaifman graph of $\mathcal{A}$". Hence, in this model, algorithms are required to access new vertices only via neighbourhood queries of vertices they have already seen. This enables us to learn a concept from examples even if the background structure is too large to fit into the main memory. To obtain a reasonable running time, we intend to find algorithms that compute a hypothesis in sublinear time, measured in the size of the background structure. This local access model has already been studied for relational structures in [7, 22] for concepts definable in FO or in $\mathrm{FOCN}(\mathbb{P})$. Modifications of the local access model for strings and trees have been studied in [3, 6].

In many applications, the same background structure is used multiple times to learn different concepts. Hence, similar to the approaches in [3, 6], we allow a precomputation step to enrich the background structure with additional information. That is, instead of learning on a $(\sigma, \mathbf{W})$-structure $\mathcal{A}$, we use an enriched $(\sigma^*, \mathbf{W})$-structure $\mathcal{A}^*$, which has the same universe as $\mathcal{A}$, but $\sigma^* \supseteq \sigma$ contains additional relation symbols. The hypotheses we compute may make use of this additional information and thus, instead of representing them via formulas from the fixed set Φ, we consider a set Φ^* of formulas of signature σ^*. These formulas may even belong to a logic L^* different from L. We study the following learning problem.

[4] with $\mathbb{P}$- and $\mathbb{S}$-oracles, i.e., the operations $+_S, \cdot_S$ for $S \in \mathbb{S}$ and checking if a tuple belongs to $[\![\mathsf{P}]\!]$ for $\mathsf{P} \in \mathbb{P}$ can be done in constant time by referring to an oracle that provides us with the answers

▶ **Problem 5.1** (Exact Learning with Precomputation). Let $\Phi \subseteq \mathrm{L}[\sigma, \mathbb{S}, \mathbf{W}]$ and $\Phi^* \subseteq \mathrm{L}^*[\sigma^*, \mathbb{S}, \mathbf{W}]$ such that, for every $(\sigma, \mathbf{W})$-structure $\mathcal{A}$, there is a $(\sigma^*, \mathbf{W})$-structure $\mathcal{A}^*$ with $U(\mathcal{A}^*) = U(\mathcal{A})$ that satisfies $\mathcal{C}(\Phi, \mathcal{A}, k, \ell) \subseteq \mathcal{C}(\Phi^*, \mathcal{A}^*, k, \ell)$, i.e. every concept that can be defined on $\mathcal{A}$ using Φ can also be defined on $\mathcal{A}^*$ using Φ^*. The task is as follows.

Given a training sequence $T = \big((\bar{a}_1, b_1), \dots, (\bar{a}_t, b_t)\big) \in (A^k \times \{0,1\})^t$ and, for a $(\sigma, \mathbf{W})$-structure $\mathcal{A}$, local access to the associated $(\sigma^*, \mathbf{W})$-structure $\mathcal{A}^*$,

return a formula $\varphi^* \in \Phi^*$ and a tuple $\bar{v} \in A^\ell$ of parameters such that the hypothesis $[\![\varphi^*(\bar{x}, \bar{y})]\!]^{\mathcal{A}^*}(\bar{x}, \bar{v})$ is consistent with T, i.e. it maps $\bar{a}_i$ to b_i for every $i \in [t]$.

The algorithm may reject if there is no consistent classifier using a formula from Φ on $\mathcal{A}$.

Next, we examine requirements for Φ and Φ^* that help us solve Problem 5.1 efficiently. Following the approach presented in [7], to obtain algorithms that run in sublinear time, we study concepts that can be represented via a set of *local* formulas Φ with a finite set Φ^* of normal forms. Using Feferman-Vaught decompositions and the locality of the formulas, we can then limit the search space for the parameters to those that are in a certain neighbourhood of the training sequence. Recall that Φ is a set of formulas $\varphi(\bar{x}, \bar{y})$ in $\mathrm{L}[\sigma, \mathbb{S}, \mathbf{W}]$ with $|\bar{x}| = k$ and $|\bar{y}| = \ell$. In the following, we require Φ to have the following property.

▶ **Property 5.2.** There are a signature σ^*, a logic L^*, an $r \in \mathbb{N}$, and a finite set of r-local formulas $\Phi^* \subseteq \mathrm{L}^*[\sigma^*, \mathbb{S}, \mathbf{W}]$ such that the following hold.

(1) For every $(\sigma, \mathbf{W})$-structure $\mathcal{A}$, there is a $(\sigma^*, \mathbf{W})$-structure $\mathcal{A}^*$ with $U(\mathcal{A}^*) = U(\mathcal{A})$ such that, for every $\varphi(\bar{x}, \bar{y}) \in \Phi$, there is a $\varphi^*(\bar{x}, \bar{y}) \in \Phi^*$ with $\mathcal{A} \models \varphi[\bar{a}, \bar{b}] \iff \mathcal{A}^* \models \varphi^*[\bar{a}, \bar{b}]$ for all $\bar{a} \in A^k$, $\bar{b} \in A^\ell$.

(2) Every $\varphi^* \in \Phi^*$ has, for every partition $(\bar{z}; \bar{z}')$ of the free variables of φ^*, a Feferman-Vaught decomposition in Φ^* w.r.t. $(\bar{z}; \bar{z}')$.

(3) For all $\varphi_1^*, \varphi_2^* \in \Phi^*$, the set Φ^* contains formulas equivalent to $\neg\varphi_1^*$ and to $(\varphi_1^* \vee \varphi_2^*)$.

This property suffices to solve Problem 5.1:

▶ **Theorem 5.3** (Exact Learning with Precomputation). *There is an algorithm that solves Problem 5.1 with local access to a structure $\mathcal{A}^*$ associated with a structure $\mathcal{A}$ in time $f_{\Phi^*}(\mathcal{A}^*) \cdot \big(\log n + d + t\big)^{\mathcal{O}(1)}$, where $\mathcal{A}$, $\mathcal{A}^*$, Φ, and Φ^* are as described in Property 5.2, t is the number of training examples, n and d are the size and the degree of $\mathcal{A}^*$, and $f_{\Phi^*}(\mathcal{A}^*)$ is an upper bound on the time complexity of model checking for formulas in Φ^* on $\mathcal{A}^*$.*

We prove the theorem in Section 5.1.

Apart from exact learning with precomputation, we also study hypotheses that generalise well in the following sense. The *generalisation error* of a hypothesis $h\colon A^k \to \{0,1\}$ for a probability distribution $\mathcal{D}$ on $A^k \times \{0,1\}$ is

$$\mathrm{err}_{\mathcal{D}}(h) \;:=\; \Pr_{(\bar{a},b)\sim\mathcal{D}}(h(\bar{a}) \neq b).$$

We write $\mathrm{rat}(0,1)$ for the set of all rationals q with $0 < q < 1$. A hypothesis class $\mathcal{H} \subseteq \{0,1\}^{A^k}$ is *agnostically PAC-learnable* if there is a function $t_{\mathcal{H}}\colon \mathrm{rat}(0,1)^2 \to \mathbb{N}$ and a learning algorithm $\mathfrak{L}$ such that for all $\varepsilon, \delta \in \mathrm{rat}(0,1)$ and for every distribution $\mathcal{D}$ over $A^k \times \{0,1\}$, when running $\mathfrak{L}$ on a sequence T of $t_{\mathcal{H}}(\varepsilon, \delta)$ examples drawn i.i.d. from $\mathcal{D}$, it holds that

$$\Pr\left(\mathrm{err}_{\mathcal{D}}(\mathfrak{L}(T)) \leqslant \inf_{h\in\mathcal{H}} \mathrm{err}_{\mathcal{D}}(h) + \varepsilon\right) \geqslant 1 - \delta.$$

The following theorem, which we prove in Section 5.2, provides an agnostic PAC learning algorithm.

▶ **Theorem 5.4** (Agnostic PAC Learning with Precomputation). *Let $\mathcal{A}$, $\mathcal{A}^*$, and Φ^* be as in Property 5.2. There is an $s \in \mathbb{N}$ such that, given local access to $\mathcal{A}^*$, the hypothesis class $\mathcal{H} := \mathcal{C}(\Phi^*, \mathcal{A}^*, k, \ell)$ is agnostically PAC-learnable with $t_{\mathcal{H}}(\varepsilon, \delta) = s \cdot \left\lceil \frac{\log(n/\delta)}{\varepsilon^2} \right\rceil$ via an algorithm that, given $t_{\mathcal{H}}(\varepsilon, \delta)$ examples, returns a hypothesis of the form $(\varphi^*, \bar{v}^*)$ with $\varphi^* \in \Phi^*$ and $\bar{v}^* \in A^\ell$ in time $f_{\Phi^*}(\mathcal{A}^*) \cdot \big(\log n + d + \frac{1}{\varepsilon} + \log \frac{1}{\delta}\big)^{\mathcal{O}(1)}$ with only local access to $\mathcal{A}^*$, where n and d are the size and the degree of $\mathcal{A}^*$, and $f_{\Phi^*}(\mathcal{A}^*)$ is an upper bound on the time complexity of model checking for formulas in Φ^* on $\mathcal{A}^*$.*

The next remark establishes the crucial link between the learning results of this section and the locality results of Section 4: it shows that suitably chosen sets $\Phi \subseteq \mathrm{FOWA}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$ indeed have Property 5.2.

▶ Remark 5.5. Fix a $q \in \mathbb{N}$ and let $\Phi := \Phi_{q,k+\ell}$ be the set of all $\mathrm{FO}[\sigma]$-formulas φ of quantifier rank at most q and with free variables among $\{x_1, \dots, x_k, y_1, \dots, y_\ell\}$. By the well-known properties of first-order logic, Φ has Property 5.2 (e.g. via $L' := L = \mathrm{FO}$, $\sigma^* := \sigma$, and $\mathcal{A}^* := \mathcal{A}$; this is exactly the setting considered in [7]). By using the locality properties of FOW_1 and FOWA_1 from Section 4, we can apply a similar reasoning to $\mathrm{FOWA}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$ as to $\mathrm{FO}[\sigma]$: let the collections $\mathbb{P}$ and $\mathbb{S}$ be finite (but $\mathbb{S}$ may contain some infinite rings or abelian groups), fix a finite set $\mathcal{S}$ of elements $s \in S \in \mathbb{S}$, and fix a $q \in \mathbb{N}$. Let $\Phi := \Phi_{q,k+\ell,\mathcal{S}}$ be the set of all $\mathrm{FOWA}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$-formulas φ of quantifier rank and aggregation depth at most q and with free variables among $\{x_1, \dots, x_k, y_1, \dots, y_\ell\}$ that have the following additional property: all symbols $s \in S \in \mathbb{S}$ that are present in φ belong to $\mathcal{S}$, all $\mathbf{W}$-products present in φ have length at most q, and the maximum nesting depth of term constructions using rule (9) in order to construct terms present in φ is at most q. This set Φ has Property 5.2. To see why, note that up to logical equivalence, Φ only contains a finite number of formulas. For each of these finitely many formulas φ, we apply Theorem 4.7 to obtain an extension σ_φ of σ, a σ_φ-expansion $\mathcal{A}^\varphi$ of $\mathcal{A}$, and a local $\mathrm{FOW}_1(\mathbb{P})[\sigma_\varphi, \mathbb{S}, \mathbf{W}]$-formula φ'. Then we let σ^* be the union of all the σ_φ, we let $\mathcal{A}^*$ be the σ^*-expansion of $\mathcal{A}$ whose σ_φ-reduct coincides with $\mathcal{A}^\varphi$ for each φ, and we let Φ' be the set of all the formulas φ'. Choose a number $r \in \mathbb{N}$ such that each of the $\varphi' \in \Phi'$ is r-local. Now we can repeatedly apply Theorem 4.3, take the r-localisations $\alpha^{(r)}, \beta^{(r)}$ of the resulting formulas α, β, and take Boolean combinations to obtain a finite extension Φ^* of Φ' such that Φ^* satisfies statements (2) and (3) of Property 5.2 and contains only r-local formulas (see [23] for details on this construction). This Φ^* witnesses that $\Phi := \Phi_{q,k+\ell,\mathcal{S}}$ has Property 5.2.

5.1 Exact Learning with Precomputation

Section 5.1 is devoted to the proof of Theorem 5.3.

Let $\mathcal{A}$ be a $(\sigma, \mathbf{W})$-structure and let $\mathcal{A}^*$ and Φ^* be as in Property 5.2. To prove Theorem 5.3, we present an algorithm that follows similar ideas as the algorithm presented in [7]. Note, however, that [7] focuses on first-order logic, whereas our setting allows to achieve results for considerably stronger logics.

While the set of possible formulas Φ^* already has constant size, we have to reduce the parameter space to obtain an algorithm that runs in sublinear time. Since the formulas in Φ^* are r-local, we show that it suffices to consider parameters in a neighbourhood of the training sequence with a fixed radius.

For $S \subseteq A$ and an element $b \in A$, let $\mathrm{dist}^{\mathcal{A}^*}(b, S) := \min_{a \in S} \mathrm{dist}^{\mathcal{A}^*}(b, a)$. For $R \geqslant 0$, set $N_R^{\mathcal{A}^*}(S) := \bigcup_{a \in S} N_R^{\mathcal{A}^*}(a)$. Also, for a training sequence $T = \big((\bar{a}_1, b_1), \dots, (\bar{a}_t, b_t)\big) \in (A^k \times \{0,1\})^t$, let $N_R^{\mathcal{A}^*}(T) := N_R^{\mathcal{A}^*}(S)$, where S is the set of all $a \in A$ that occur in one of the $\bar{a}_i$.

```
1:  $N \leftarrow N^{\mathcal{A}^*}_{(2r+1)\ell}(T)$
2:  for all $\bar{v}^* \in N^\ell$ do
3:      for all $\varphi^*(\bar{x}, \bar{y}) \in \Phi^*$ do
4:          $consistent \leftarrow$ true
5:          for all $i \in [t]$ do
6:              $\mathcal{N} = \mathcal{N}^{\mathcal{A}^*}_r(\bar{a}_i\bar{v}^*)$
7:              if $[\![\varphi^*(\bar{a}_i, \bar{v}^*)]\!]^{\mathcal{N}} \neq b_i$ then
8:                  $consistent \leftarrow$ false
9:          if $consistent$ then
10:             return $(\varphi^*, \bar{v}^*)$
11: reject
```

```
1:  $N \leftarrow N^{\mathcal{A}^*}_{(2r+1)\ell}(T)$
2:  $err_{\text{min}} \leftarrow |T| + 1$
3:  for all $\bar{v}^* \in N^\ell$ do
4:      for all $\varphi^*(\bar{x}, \bar{y}) \in \Phi^*$ do
5:          $err_{\text{cur}} \leftarrow 0$
6:          for all $i \in [t]$ do
7:              $\mathcal{N} = \mathcal{N}^{\mathcal{A}^*}_r(\bar{a}_i\bar{v}^*)$
8:              if $[\![\varphi^*(\bar{a}_i, \bar{v}^*)]\!]^{\mathcal{N}} \neq b_i$ then
9:                  $err_{\text{cur}} \leftarrow err_{\text{cur}} + 1$
10:         if $err_{\text{cur}} < err_{\text{min}}$ then
11:             $err_{\text{min}} \leftarrow err_{\text{cur}}$
12:             $\varphi^*_{\text{min}} \leftarrow \varphi^*$
13:             $\bar{v}^*_{\text{min}} \leftarrow \bar{v}^*$
14: return $(\varphi^*_{\text{min}}, \bar{v}^*_{\text{min}})$
```

Figure 1 Learning algorithms for Theorems 5.3 (left) and 5.4 (right). Both algorithms use as input a training sequence $T = \big((\bar{a}_1, b_1), \ldots, (\bar{a}_t, b_t)\big) \in (A^k \times \{0,1\})^t$ and have only local access to the structure $\mathcal{A}^*$.

▶ **Lemma 5.6.** *Let $T = \big((\bar{a}_1, b_1), \ldots, (\bar{a}_t, b_t)\big) \in (A^k \times \{0,1\})^t$ be consistent with some classifier in $\mathcal{C}(\Phi^*, \mathcal{A}^*, k, \ell)$. Then there are a formula $\varphi^*(\bar{x}, \bar{y}) \in \Phi^*$ and a tuple $\bar{v}^* \in N^{\mathcal{A}^*}_{(2r+1)\ell}(T)^\ell$ such that $[\![\varphi^*(\bar{x}, \bar{y})]\!]^{\mathcal{A}^*}(\bar{x}, \bar{v}^*)$ is consistent with T.*

The proof can be found in [23]. It is similar to the proof of the analogous statement in [7] for the special case of FO, but relies on Property 5.2. We can now prove Theorem 5.3.

Proof of Theorem 5.3. We show that the algorithm depicted on the left-hand side of Figure 1 fulfils the requirements given in Theorem 5.3. The algorithm goes through all tuples $\bar{v}^* \in (N^{\mathcal{A}^*}_{(2r+1)\ell}(T))^\ell$ and all formulas $\varphi^*(\bar{x}, \bar{y}) \in \Phi^*$. A hypothesis $[\![\varphi^*(\bar{x}, \bar{y})]\!]^{\mathcal{A}^*}(\bar{x}, \bar{v}^*)$ is consistent with the training sequence T if and only if $[\![\varphi^*(\bar{a}_i, \bar{v}^*)]\!]^{\mathcal{A}^*} = b_i$ for all $i \in [t]$. Since Φ^* only contains r-local formulas, this holds if and only if $[\![\varphi^*(\bar{a}_i, \bar{v}^*)]\!]^{\mathcal{N}^{\mathcal{A}^*}_r(\bar{a}_i\bar{v}^*)} = b_i$ for every $i \in [t]$. Hence, the algorithm only returns a hypothesis if it is consistent. Furthermore, if there is a consistent hypothesis in $\mathcal{C}(\Phi, \mathcal{A}, k, \ell)$, then by Property 5.2 (1), there is also a consistent hypothesis in $\mathcal{C}(\Phi^*, \mathcal{A}^*, k, \ell)$, and Lemma 5.6 ensures that the algorithm then returns a hypothesis.

It remains to show that the algorithm satisfies the running time requirements while only using local access to the structure $\mathcal{A}^*$. For all $\bar{a} \in A^k$ and $\bar{v}^* \in A^\ell$, we can bound the size of their neighbourhood by $\big|N^{\mathcal{A}^*}_r(\bar{a}\bar{v}^*)\big| \leqslant (k+\ell) \cdot \sum_{i=0}^r d^i \leqslant (k+\ell) \cdot (1 + d^{r+1})$. Therefore, the representation size of the substructure $\mathcal{N}^{\mathcal{A}^*}_r(\bar{a}\bar{v}^*)$ is in $\mathcal{O}\big((k+\ell) \cdot d^{r+1} \cdot \log n\big)$. Thus, the consistency check in lines 4–8 runs in time $f_{\Phi^*}(\mathcal{A}^*) \cdot t \cdot \mathcal{O}\big((k+\ell) \cdot d^{r+1} \cdot \log n\big)$. The algorithm checks up to $|N|^\ell \cdot |\Phi^*| \in \mathcal{O}\big((tkd^{(2r+1)\ell+1})^\ell \cdot |\Phi^*|\big)$ hypotheses with $N = N^{\mathcal{A}^*}_{(2r+1)\ell}(T)$. All in all, since k, ℓ, r are considered constant, the running time of the algorithm is in $f_{\Phi^*}(\mathcal{A}^*) \cdot (\log n + d + t)^{\mathcal{O}(1)}$ and it only uses local access to the structure $\mathcal{A}^*$. ◀

5.2 Agnostic PAC Learning with Precomputation

Section 5.2 is devoted to the proof of Theorem 5.4.

To obtain a hypothesis that generalises well, we follow the *Empirical Risk Minimization* rule (ERM) [19, 24], i.e. our algorithm should return a hypothesis h that minimises the *training error*

$$\mathrm{err}_T(h) \;:=\; \tfrac{1}{|T|}\cdot\left|\{(\bar a,b)\in T : h(\bar a)\neq b\}\right|$$

on the training sequence T. To prove Theorem 5.4, we use the following result from [19].

▶ **Lemma 5.7** (Uniform Convergence [19]). *Let $\mathcal{H}$ be a finite class of hypotheses $h\colon A^k\to\{0,1\}$. Then $\mathcal{H}$ is agnostically PAC-learnable using an ERM algorithm and*

$$t_{\mathcal{H}}(\varepsilon,\delta) \;:=\; \left\lceil \frac{2\log(2\,|\mathcal{H}|\,/\delta)}{\varepsilon^2} \right\rceil.$$

Proof of Theorem 5.4. We show that the algorithm depicted on the right-hand side of Figure 1 fulfils the requirements from Theorem 5.4. The algorithm goes through all tuples $\bar v^*\in(N^{\mathcal{A}^*}_{(2r+1)\ell}(T))^\ell$ and all formulas $\varphi^*(\bar x,\bar y)\in\Phi^*$ and counts the number of errors that $[\![\varphi^*(\bar x,\bar y)]\!]^{\mathcal{A}^*}(\bar x,\bar v^*)$ makes on T. Then it returns the hypothesis with the minimal training error.

Since Φ^* and A^ℓ are finite, $\mathcal{H}=\mathcal{C}(\Phi^*,\mathcal{A}^*,k,\ell)$ is finite. Thus, using Lemma 5.7, $\mathcal{H}$ is agnostically PAC-learnable with $t_{\mathcal{H}}(\varepsilon,\delta) = \left\lceil\frac{2\log(2|\mathcal{H}|/\delta)}{\varepsilon^2}\right\rceil \leqslant \left\lceil\frac{4\ell\log(|\Phi^*|)\log(n/\delta)}{\varepsilon^2}\right\rceil$. The running time analysis works as in the proof of Theorem 5.3. The algorithm returns a hypothesis in time $f_{\Phi^*}(\mathcal{A}^*)\cdot(\log n+d+t)^{\mathcal{O}(1)}$. For a training sequence of length $t=t_{\mathcal{H}}(\varepsilon,\delta)$, we obtain a running time in $f_{\Phi^*}(\mathcal{A}^*)\cdot\big(\log n+d+\log(1/\delta)+1/\varepsilon\big)^{\mathcal{O}(1)}$. ◀

6 Putting Things Together

Let the collections $\mathbb{P}$ and $\mathbb{S}$ be finite (but $\mathbb{S}$ may contain infinite rings or abelian groups), fix a *finite* set $\mathcal{S}$ of elements $s\in S\in\mathbb{S}$, fix a $q\in\mathbb{N}$, and let $\Phi:=\Phi_{q,k+\ell,\mathcal{S}}$ be the set of $\mathsf{FOWA}_1(\mathbb{P})[\sigma,\mathbb{S},\mathbf{W}]$-formulas defined in Remark 5.5. Let Φ^*, σ^*, and $\mathcal{A}^*$ (for all $(\sigma,\mathbf{W})$-structures $\mathcal{A}$) be as described in Remark 5.5. By Theorem 4.7, $\mathcal{A}^*$ can be computed from $\mathcal{A}$ in time $|A|{\cdot}d^{\mathcal{O}(1)}$, where d is the degree of $\mathcal{A}$. By Remark 5.5, the formulas in Φ^* are r-local for a fixed number r, and this implies that model checking for a formula in Φ^* on $\mathcal{A}^*$ can be done in time polynomial in d. Combining this with Theorems 5.3 and 5.4 yields the following[5].

▶ **Theorem 6.1.** *Let n and d denote the size and the degree of $\mathcal{A}$.*

(1) *There is an algorithm that solves Exact Learning with Precomputation for Φ and Φ^* with local access to a structure $\mathcal{A}^*$ associated with a structure $\mathcal{A}$ in time $(\log n+d+t)^{\mathcal{O}(1)}$, where t is the number of training examples.*

(2) *There is an $s\in\mathbb{N}$ such that, given local access to a structure $\mathcal{A}^*$ associated with a structure $\mathcal{A}$, the hypothesis class $\mathcal{H}:=\mathcal{C}(\Phi^*,\mathcal{A}^*,k,\ell)$ is agnostically PAC-learnable with $t_{\mathcal{H}}(\varepsilon,\delta)=s\cdot\left\lceil\frac{\log(n/\delta)}{\varepsilon^2}\right\rceil$ via an algorithm that, given $t_{\mathcal{H}}(\varepsilon,\delta)$ examples, returns a hypothesis of the form $(\varphi^*,\bar v^*)$ with $\varphi^*\in\Phi^*$ and $\bar v^*\in A^\ell$ in time $\big(\log n+d+\frac{1}{\varepsilon}+\log\frac{1}{\delta}\big)^{\mathcal{O}(1)}$ with only local access to $\mathcal{A}^*$.*

Additionally, the algorithms can be chosen such that the returned hypotheses can be evaluated in time $(\log n+d)^{\mathcal{O}(1)}$.

[5] All mentioned algorithms are assumed to have $\mathbb{P}$- and $\mathbb{S}$-oracles, so that operations $+_S,\cdot_S$ for $S\in\mathbb{S}$ and checking if a tuple is in $[\![\mathsf{P}]\!]$ for $\mathsf{P}\in\mathbb{P}$ takes time $\mathcal{O}(1)$.

We conclude with an example that illustrates an application scenario for Theorem 6.1.

▶ **Example 6.2.** Recall the $(\sigma, \mathbf{W})$-structure $\mathcal{A}$ for the online marketplace from part (a) of Examples 3.1, 3.2, and 3.6. Retailers can pay the marketplace to advertise their products to consumers. Since the marketplace demands a fee for every single view of the advertisement, retailers want the marketplace to only show the advertisement to those consumers that are likely to buy the product. One possible way to choose suitable consumers is to consider only those who buy a variety of products from the same or a similar product group as the advertised product and who are thus more likely to try new products that are similar to the advertised one. At the same time, the money spent by the chosen consumers on the product group should be above average.

In the previous examples, we have already seen a formula $\varphi_{\text{spending}}(c)$ that defines consumers who have spent at least as much as the average consumer on the product group. The formula depends on a formula $\varphi_{\text{group}}(p)$ that defines a certain group of products based on the structure of their transactions. Due to the connection between graph neural networks and the Weisfeiler-Leman algorithm described in [16], we may assume that there is a formula in $\text{FO}[\sigma]$ that at least roughly approximates such a product group. Likewise, we might assume that there is a formula $\varphi_{\text{variety}}(c)$ in $\text{FO}[\sigma]$ that defines consumers with a wide variety of products bought from a specific product group. However, it is a non-trivial task to design such formulas by hand. It is even not clear whether there exist better rules for finding suitable consumers. Meanwhile, we can easily show the advertisement to consumers and then check whether they buy the product. Thus, we can generate a list with positive and negative examples of consumers. Since the proposed rule can be defined in $\text{FOWA}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$ as $\varphi_{\text{advertise}}(c) := (\varphi_{\text{variety}}(c) \wedge \varphi_{\text{spending}}(c))$, we can use one of the learning algorithms from Theorem 6.1 to find good definitions for $\varphi_{\text{variety}}(c)$ and $\varphi_{\text{group}}(p)$ or to learn an even better definition for $\varphi_{\text{advertise}}(c)$ in $\text{FOWA}_1(\mathbb{P})[\sigma, \mathbb{S}, \mathbf{W}]$ from examples.

We believe that our results can be generalised to an extension of FOWA_1 where constructions of the form $\mathsf{P}(t_1, \ldots, t_m)$ are not restricted to the case that $|V| = 1$ for $V := \text{free}(t_1) \cup \cdots \cup \text{free}(t_m)$, but may also be used in a guarded setting of the form $\big(\mathsf{P}(t_1, \ldots, t_m) \wedge \bigwedge_{v,w \in V} \text{dist}(v, w) \leqslant r\big)$. It would also be interesting to study non-Boolean classification problems, where classifiers are described by $\mathbb{S}$-terms defined in a suitable fragment of FOWA. We plan to do this in future work.

References

1 Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987. doi:10.1007/BF00116828.

2 Haim Gaifman. On local and non-local properties. In Jacques Stern, editor, *Proceedings of the Herbrand Symposium*, volume 107 of *Studies in Logic and the Foundations of Mathematics*, pages 105–135. North-Holland, 1982. doi:10.1016/S0049-237X(08)71879-2.

3 Emilie Grienenberger and Martin Ritzert. Learning definable hypotheses on trees. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, pages 24:1–24:18, 2019. doi:10.4230/LIPIcs.ICDT.2019.24.

4 Martin Grohe. Logic, graphs, and algorithms. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 357–422. Amsterdam University Press, 2008.

5 Martin Grohe. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS*

2020, Portland, OR, USA, June 14-19, 2020, pages 1–16. ACM, 2020. doi:10.1145/3375395.3387641.

6 Martin Grohe, Christof Löding, and Martin Ritzert. Learning MSO-definable hypotheses on strings. In *International Conference on Algorithmic Learning Theory, ALT 2017, 15-17 October 2017, Kyoto University, Kyoto, Japan*, pages 434–451, 2017. URL: http://proceedings.mlr.press/v76/grohe17a.html.

7 Martin Grohe and Martin Ritzert. Learning first-order definable concepts over structures of small degree. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005080.

8 Martin Grohe and Nicole Schweikardt. First-order query evaluation with cardinality conditions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 253–266, 2018. doi:10.1145/3196959.3196970.

9 Martin Grohe and György Turán. Learnability and definability in trees and similar structures. *Theory Comput. Syst.*, 37(1):193–220, 2004. doi:10.1007/s00224-003-1112-8.

10 Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 855–864, 2016. doi:10.1145/2939672.2939754.

11 David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Inf. Comput.*, 100(1):78–150, 1992. doi:10.1016/0890-5401(92)90010-D.

12 Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994. URL: https://mitpress.mit.edu/books/introduction-computational-learning-theory.

13 Dietrich Kuske and Nicole Schweikardt. First-order logic with counting. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005133.

14 Dietrich Kuske and Nicole Schweikardt. Gaifman normal forms for counting extensions of first-order logic. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 133:1–133:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.133.

15 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

16 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *The 33rd AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4602–4609, 2019. doi:10.1609/aaai.v33i01.33014602.

17 Shimei Pan and Tao Ding. Social media-based user embedding: A literature review. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 6318–6324, 2019. doi:10.24963/ijcai.2019/881.

18 Maximilian Schleich, Dan Olteanu, Mahmoud Abo Khamis, Hung Q. Ngo, and XuanLong Nguyen. Learning models over relational data: A brief tutorial. In Nahla Ben Amor, Benjamin Quost, and Martin Theobald, editors, *Scalable Uncertainty Management - 13th International Conference, SUM 2019, Compiègne, France, December 16-18, 2019, Proceedings*, volume 11940 of *Lecture Notes in Computer Science*, pages 423–432. Springer, 2019. doi:10.1007/978-3-030-35514-2_32.

19 Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014.

20 Szymon Toruńczyk. Aggregate queries on sparse databases. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 427–443. ACM, 2020. `doi:10.1145/3375395.3387660`.

21 Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. `doi:10.1145/1968.1972`.

22 Steffen van Bergerem. Learning concepts definable in first-order logic with counting. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13, 2019. `doi:10.1109/LICS.2019.8785811`.

23 Steffen van Bergerem and Nicole Schweikardt. Learning concepts described by weight aggregation logic. *CoRR*, abs/2009.10574, 2020. `arXiv:2009.10574`.

24 Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 831–838, 1991. URL: `http://papers.nips.cc/paper/506-principles-of-risk-minimization-for-learning-theory`.

Open Bar – a Brouwerian Intuitionistic Logic with a Pinch of Excluded Middle

Mark Bickford
Cornell University, Ithaca, NY, USA
markb@cs.cornell.edu

Liron Cohen
Ben Gurion University of the Negev, Beer Sheva, Israel
cliron@cs.bgu.ac.il

Robert L. Constable
Cornell University, Ithaca, NY, USA
rc@cs.cornell.edu

Vincent Rahli
University of Birmingham, UK
V.Rahli@bham.ac.uk

Abstract

One of the differences between Brouwerian intuitionistic logic and classical logic is their treatment of time. In classical logic truth is atemporal, whereas in intuitionistic logic it is time-relative. Thus, in intuitionistic logic it is possible to acquire new knowledge as time progresses, whereas the classical Law of Excluded Middle (LEM) is essentially flattening the notion of time stating that it is possible to decide whether or not some knowledge will *ever* be acquired. This paper demonstrates that, nonetheless, the two approaches are not necessarily incompatible by introducing an intuitionistic type theory along with a Beth-like model for it that provide some middle ground. On one hand they incorporate a notion of progressing time and include evolving mathematical entities in the form of choice sequences, and on the other hand they are consistent with a variant of the classical LEM. Accordingly, this new type theory provides the basis for a more classically inclined Brouwerian intuitionistic type theory.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Constructive mathematics

Keywords and phrases Intuitionism, Extensional type theory, Constructive Type Theory, Realizability, Choice sequences, Classical Logic, Law of Excluded Middle, Theorem proving, Coq

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.11

Supplementary Material All the results in the paper are formalized in Coq, see `https://github.com/vrahli/NuprlInCoq/tree/ls3/`.

1 Introduction

Classical logic and intuitionistic logic are commonly viewed as distinct philosophies. Much of the difference between the two philosophies can be attributed to the way they handle the notion of *time*. In intuitionistic logic time plays a major role as the intuitionistic notions of knowledge and truth evolve over time. In particular, the seminal concept of intuitionistic mathematics as developed by Brouwer is that of *infinitely proceeding* sequences of choices (called choice sequences) from which the continuum is defined [47, Ch.3]. Choice sequences are a primitive concept of finite sequences of entities (e.g., natural numbers) that are never complete, and can always be further extended with new choices [26, 48, 44, 43, 31, 50, 36]. These sequences can be "free" in the sense that they are not necessarily procedurally generated. This manifestation of the evolving concept of time in intuitionistic logic entails a notion of computability that goes far beyond that of Church-Turing. In fact, the concept of evolving

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 11; pp. 11:1–11:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

knowledge in intuitionistic logic is grounded in Krikpe's Schema, which in turn relies on the notion of choice sequences, and is inconsistent with Church's Thesis [49, Sec.5]. Classical logic, on the other hand, is time-invariant. That is, its notions of knowledge and truth are constant and so the aspect of time is, intuitively speaking, flattened. As mentioned by van Atten, "Many people believe, unlike Brouwer, that mathematical truths are not tensed but eternal – either because such truths are outside time altogether (atemporal) or because they hold in all time (omnitemporal)" [47, p.19].

This critical difference between the two philosophies has been used extensively to refute classical results in intuitionistic logic. Brouwer himself used his concept of choice sequences to provide weak counterexamples to classical results such as "any real number different from 0 is also apart from 0" [24, Ch.8]. Those counterexamples are called *weak* in the sense that they depend on the existence of formulas that have not been either proven or disproven yet (e.g., the Goldbach conjecture). By defining a choice sequence in which the value 1 can only be picked once such an undecided conjecture has been resolved (proved or disproved), one could resolve this undecided conjecture using the Law of Excluded Middle (LEM), leading to a weak counterexample of LEM [12, Ch.1,Sec.1]. Kripke [33, Sec.1.1] also used the unconstrained nature of choice sequences to refute other classical results, namely Kuroda's conjecture and Markov's principle in Kreisel's FC system [28].[1] A constructive version of LEM in which the operators are interpreted constructively is also false in realizability theories such as the CTT constructive type theory [14, 5] because it allows deciding the undecidable halting problem [40, Sec.6.3] (therefore not relying on undecided conjectures). However, a weaker version of LEM that does not require providing a realizer of either its left or right disjuncts, was proved to be consistent with CTT [17, 27, 40].But using a similar technique to Brouwer's, even this weak version of LEM was shown to be inconsistent with BITT, an intuitionistic extension of CTT with a computable notion of choice sequences [8, Appx.A].

The use of the growing-over-time nature of choice sequences to refute classical axioms, and in particular LEM which is a key component of classical reasoning, seems to indicate an incompatibility between classical logic and intuitionistic logic. However, in this paper we show that this does not have to be the case. To this end, we present a relaxed model of time that mitigates the two approaches. Namely, on one hand it supports the evolving nature of choice sequences, and on the other hand it enables variants of the classical LEM.

Concretely, we present OpenTT, a novel intuitionistic extensional type theory that incorporates the Brouwerian notion of choice sequences, and is inspired by BITT [8]. OpenTT goes beyond and departs from BITT in several ways. First, it is validated w.r.t. a novel Beth-like model, which we call the *open bar* model, that is significantly simpler than the one presented in [8]. Beth models were originally developed to provide meaning to intuitionistic formulas [51, 7, 21, 19],and they have proven especially well-suited to interpret choice sequences [49]. In such models, formulas are interpreted w.r.t. infinite trees of elements (such as numbers). The models are typically formulated using a forcing interpretation where the forcing conditions are finite elements of those trees that provide meaning to choice sequences at a given point in time. Allowing access within the logic to the infinitely proceeding elements of the forcing layer, i.e., the branches of the Beth trees formulas are interpreted against, enables the use of the undecided nature of those elements to derive the negation of otherwise classically valid formulas such as LEM. The open bar model sufficiently weakens the "undecided" nature of those elements to enable validating a variant of LEM.

[1] This method to refute classical axioms was reused via forcing methods (see, e.g., [18, Sec.7.2.4] for the relation between forcing and choice sequences). E.g., the independence of Markov's Principle with Martin-Löf's type theory was proven using a forcing method where the "free" nature of forcing conditions replaces the "free" nature of free choice sequences in Kripke's proof [15].

Another benefit of OpenTT over BITT is that the notion of time induced by the new model is flexible enough to capture an intuitionistic theory of computable choice sequences, and in particular the Axiom of Open Data (a continuity axiom) that was missing from BITT [8] and is a key axiom of choice sequence theories. Therefore, OpenTT provides a computational setting for exploring the implications of such entities, for example, it can enable the development of constructive Brouwerian real number theories. At the same time, it also enables validating variants of the classical LEM. In other words, OpenTT together with the open bar model presented in the paper enable a more relaxed notion of time, providing a basis for a more classically-inclined Brouwerian intuitionistic theory.

Contributions and roadmap. Sec. 2 describes the core *syntactic* components of the type theory OpenTT. Sec. 3 presents the novel open bar *semantic* model, which is used to validate OpenTT. Then, OpenTT is shown to capture both a theory of choice sequences (Sec. 4), as well as a variant of LEM (Sec. 5). Sec. 6 concludes by discussing related and future work. All the results in the paper are formalized in Coq, see `https://github.com/vrahli/NuprlInCoq/tree/ls3/`, and we provide clickable hyperlinks to the formalization throughout the paper.

2 OpenTT and Choice Sequences

OpenTT is an intuitionistic extensional dependent type theory. It is composed of an untyped programming language, and a dependent type system that associates types with programs. A type T, viewed as a proposition, is said to be true if it is inhabited, i.e., if some program t has type T – in which case t is said to realize T. This connection is made formal through a realizability model described in Sec. 3, where types are interpreted as partial equivalence relations on programs. In addition to standard program constructs, OpenTT contains computable choice sequences.

Choice sequences are the seminal component in Brouwer's intuitionistic theory, and the one manifesting notions of time and growth over time. Choice sequences are infinitely proceeding sequences of elements, which are chosen over time from a previously well-defined collection. There are two main classes of choice sequences, which are often referred to as *lawlike* and *lawless* [45]. The lawlike ones are "completed constructions" [45, Sec.1.2], where the choices must be chosen w.r.t. a pre-determined "law" (e.g., a general recursive program). The lawless ones, by contrast, are never fully completed and can always be extended over time with further choices that are not constrained by any law, that is, they can be chosen "freely" (hence the name *free choice sequences*). In this paper we focus on a theory with free choice sequences, which is a key distinguishing feature in Brouwer's intuitionistic logic, and a manifestation of the fact that time is an essential component of Brouwer's logic because unlike lawlike sequences that are time-invariant, lawless ones keep on evolving over time.

The notion of time in OpenTT is captured through the use of worlds. The worlds discussed in Sec. 2.2 constitute, as is standard practice, a poset, and are concretely defined as states that store definitions as well as choice sequences' choices. Thus, a world captures a state at a given point in time. The evolving nature of time is then captured via a notion of world extension, allowing to add new definitions, choice sequences, and choices.

OpenTT is inspired by BITT [8]. To make the paper self-contained we shall also review the components that are identical to those in BITT, noting the differences, which we summarize here. In addition to the standard inference rules for the standard types that are listed in Fig. 1 (which are discussed in Appx. B), OpenTT also contains inference rules that capture

Figure 1 Syntax of OpenTT.

$\eta \in \texttt{CSName}$	(C.S. name)	$\delta \in \texttt{Abstraction}$	(abstraction)		
$v \in \texttt{Value} ::= vt$	(type)	$\mid \lambda x.t$	(lambda)	$\mid \langle t_1, t_2 \rangle$	(pair)
$\mid \star$	(axiom)	$\mid \texttt{inl}(t)$	(left injection)	$\mid \texttt{inr}(t)$	(right injection)
$\mid \underline{i}$	(integer)	$\mid \eta$	(choice sequence)		
$vt \in \texttt{Type} ::= \mathbf{\Pi} x{:}t_1.t_2$	(product)	$\mid \mathbf{\Sigma} x{:}t_1.t_2$	(sum)		
$\mid \mathbb{U}_i$	(universe)	$\mid t_1 = t_2 \in t$	(equality)		
$\mid t_1 + t_2$	(disjoint union)	$\mid \{x : t_1 \mid t_2\}$	(set)		
$\mid \mathbb{N}$	(numbers)	$\mid t_1 < t_2$	(less than)		
$\mid \mathbb{N}_{\lightning}$	(T.S. numbers)	$\mid t_1 <_{\lightning} t_2$	(T.S. less than)		
$\mid t_1 \# t_2$	(free from definitions)	$\mid \texttt{Free}$	(choice sequences)		
$\mid \lightning t$	(time squashing)				
$t \in \texttt{Term} ::= x$	(variable)	$\mid t_1\ t_2$	(application)		
$\mid v$	(value)	$\mid \texttt{let}\ x, y = t_1\ \texttt{in}\ t_2$	(spread)		
$\mid \texttt{fix}(t)$	(fixpoint)	$\mid \texttt{case}\ t_1\ \texttt{of}\ \texttt{inl}(x) \Rightarrow t_2 \mid \texttt{inr}(y) \Rightarrow t_3$	(decide)		
$\mid \texttt{wDepth}$	(world depth)	$\mid \textbf{if}\ t_1{=}t_2\ \textbf{then}\ t_3\ \textbf{else}\ t_4$	(equality test)		
$\mid \delta$	(abstraction)				

a theory of choice sequences, as described in Sec. 4. Among those, the Axiom of Open Data is new compared to BITT. Another key difference between OpenTT and BITT is that the former also contains a variant of the Law of Excluded Middle (the salient principle of classical logic), described in Sec. 5, which is not valid in the latter.[2]

2.1 Syntax

OpenTT's programming language is an untyped, call-by-name λ-calculus, whose syntax is given in Fig. 1, and operational semantics in Sec. 2.3. For simplicity, numbers are considered to be primitive, and we write $\underline{n}$ for an OpenTT number, where n is a metatheoretical number. A term is either (1) a variable; (2) a canonical term, i.e., a value; or (3) a non-canonical term. Non-canonical terms are evaluated according to the operational semantics presented in Sec. 2.3. As discussed below, abstractions of the form δ can be unfolded through definitions, and are otherwise left abstract for the purpose of this paper. In what follows, we use all letters as metavariables and their types can be inferred from the context.

Choice sequences. A choice sequence is identified with its name, of the form η, which for the purpose of this paper is an abstract type equipped with a decidable equality. For simplicity we only discuss choice sequences of numbers, while our Coq formalization supports more kinds of choice sequences. OpenTT includes a comparison operator on choice sequences, $\textbf{if}\ t_1{=}t_2\ \textbf{then}\ t_3\ \textbf{else}\ t_4$, which as defined in Sec. 2.3 reduces to the *then* branch if t_1 and t_2 are two choice sequences with the same name, and otherwise reduces to the *else* branch.

Types. Types are syntactic forms that are given semantics in Sec. 3 via a realizability interpretation. The type system contains standard types such as dependent products of the form $\mathbf{\Pi} x{:}t_1.t_2$ and dependent sums of the form $\mathbf{\Sigma} x{:}t_1.t_2$. For convenience we often write

[2] Precisely establishing the relationship between the two systems is left for future work.

$a =_T b$ for the type $a = b \in T$; $t \in T$ for $t =_T t$; $\mathbf{\Pi} x_1, \ldots, x_n{:}t_1.t_2$ for $\mathbf{\Pi} x_1{:}t_1. \ldots \mathbf{\Pi} x_n{:}t_1.t$ (and similarly for the other operators with binders); $t_1 \rightarrow t_2$ for the non-dependent $\mathbf{\Pi}$ type; `True` for $(\underline{0} = \underline{0} \in \mathbb{N})$; `False` for $(\underline{0} = \underline{1} \in \mathbb{N})$; and $\neg T$ for $(T \rightarrow \texttt{False})$.

OpenTT also includes types that allow capturing specific aspects of choice sequences. In particular, OpenTT includes a type `Free` of free choice sequences. It also includes the type $t\#T$ that indicates that t is a *sealed* member of T in the sense that it is equivalent to a term u in T, which is syntactically free from abstractions and choice sequences, which we denote by $\texttt{synSealed}(u)$ here (see Sec. 3 for more details). Those types are used to state axioms of the theory of choice sequences in Sec. 4.1.

2.2 Worlds

OpenTT's computation system is equipped with a library of definitions in which we also store choice sequences. We here call the library a *world*. A definition entry is a pair of an abstraction δ and a term t, written δ `==` t, which stipulates that δ unfolds to t.[3] A choice sequence entry is a pair of a choice sequence name, and a list of choices (i.e. terms).[4] For example, the pair $\langle \eta, [4, 8, 15] \rangle$ is an entry for the choice sequence named η, where $[4, 8, 15]$ is its list of choices so far. A world is therefore a state that records, at a given point in time, all the current definitions together with all the choice sequences that have been started so far, along with the choices that have been made so far for those choice sequences.

▶ **Definition 1** (Worlds). *A world w is a list of entries, where an entry is either a definition entry or a choice sequence entry. We denote by `World` the type of worlds.*

Next we introduce some necessary operations and properties on worlds.

▶ **Definition 2** (World operations and properties). *Let $w \in$ `World`. (1) $|w|$ denotes w's* depth*, that is the number of choices of its longest choice sequence. (2) w is called* singular*, denoted $\texttt{sing}(w)$, if it does not have two entries with the same name.*

The depth of worlds is used in Sec. 4.1 to approximate the modulus of continuity of a predicate at a choice sequence; while `sing` is used in Lem. 14.

A world (or a particular snapshot of the library) can be seen as a the state of knowledge at a given point in time. It may grow over time by adding new definitions, new choice sequence entries, or more terms to an already existing choice sequence entry. Accordingly, a world w_2 is said to extend a world w_1 if it contains more entries and choices, without overriding the ones in w_1. Note that the extension relation on worlds defines a partial order on `World`.

▶ **Definition 3** (World extension). *A world w_2 is said to extend w_1, denoted $w_2 \succeq w_1$, if w_1 is a list of the form $[e_1, \ldots, e_n]$ and w_2 is a concatanation of some world w and $[e_1^{'}, \ldots, e_n^{'}]$, where for all $1 \leq i \leq n$, either $e_i = e_i^{'}$ or e_i and $e_i^{'}$ are choice sequence entries with the same name such that the list in e_i is an initial segment of that in $e_i^{'}$.*

2.3 Operational Semantics

Fig. 2 presents OpenTT's small-step operational semantics. It defines the $t_1 \mapsto_w t_2$ ternary relation between two terms and a world, which expresses that t_1 reduces to t_2 in one step of computation *w.r.t. the world w*. We omit the congruence rules that allow computing within terms such as: if $t_1 \mapsto_w t_2$ then $t_1(u) \mapsto_w t_2(u)$.

[3] As the precise form of definitions is irrelevant here, we refer the interested reader to [41].

[4] Our formalization also includes mechanisms to impose further restrictions on choice sequences which are not discussed here. See `computation/library.v` for further details.

Figure 2 Operational semantics of OpenTT.

$$
\begin{array}{llllll}
(\lambda x.F)\ a & \mapsto_w & F[x\backslash a] & \eta(\underline{i}) & \mapsto_w & w[\eta][i], \quad \text{if } \eta \text{ has a } i\text{'s choice in } w \\
\texttt{fix}(v) & \mapsto_w & v\ \texttt{fix}(v) & \texttt{wDepth} & \mapsto_w & |w| \\
\multicolumn{6}{l}{\texttt{let } x, y = \langle t_1, t_2\rangle \texttt{ in } F \mapsto_w F[x\backslash t_1; y\backslash t_2]} \\
\multicolumn{6}{l}{\texttt{case inl}(t) \texttt{ of inl}(x) \Rightarrow F \mid \texttt{inr}(y) \Rightarrow G \ \ \mapsto_w \ \ F[x\backslash t]} \\
\multicolumn{6}{l}{\texttt{case inr}(t) \texttt{ of inl}(x) \Rightarrow F \mid \texttt{inr}(y) \Rightarrow G \ \ \mapsto_w \ \ G[y\backslash t]} \\
\multicolumn{6}{l}{\texttt{if } \eta_1 = \eta_2 \texttt{ then } t_1 \texttt{ else } t_2 \mapsto_w t_i, \quad \text{where } i = 1 \text{ if } \eta_1 = \eta_2, \text{ and } i = 2 \text{ otherwise}}
\end{array}
$$

The application $\eta(\underline{i})$ of a choice sequence η to a number $\underline{i}$ reduces to $w[\eta][i]$, i.e., η's i's choice recorded in w, if such a choice exists, and otherwise the computation gets stuck. Note that even though this is a call-by-name calculus, it includes the following congruence rule to access choices of choice sequences: if $t_1 \mapsto_w t_2$ then $\eta(t_1) \mapsto_w \eta(t_2)$.

In OpenTT we also allow computing the depth of a world w, that is, the number of choices recorded in its longest choice sequence entry (this is an addition to BITT). The nullary expression `wDepth` reduces to $|w|$ in one computation step. It is used to realize an axiom of the theory of choice sequences in Sec. 4.1.2. It is important to note that before introducing this new computation, all computations were *time-invariant computations* in the sense that if a term t computes to a value v in a world w_1, then it will compute to a value computationally equivalent[5] to v in any world $w_2 \succeq w_1$. For example, for numbers, if a term t computes to a number $\underline{n}$ in some world w, then it also computes to $\underline{n}$ in all extensions of w. Such terms are called *time-invariant terms.* It is straightforward to see that `wDepth` is not time-invariant, as it can compute to different numbers in different extensions of a world. For example, if w_1 contains only one choice sequence η for which 4 choices have been made, then the expression `wDepth` reduces to $\underline{4}$ in w_1. Now, adding another choice to η gives us a world $w_2 \succeq w_1$ in which `wDepth` reduces to $\underline{5}$. This operator is said to be *weakly monotonic* in the sense that if it returns $\underline{k}$ in w_1, and $w_2 \succeq w_1$, then it can only return a value $\underline{k'} \geq \underline{k}$ in w_2. We next introduce types capturing the concept of time-invariance.

2.4 Space Squashing and Time Squashing

OpenTT includes a *squashing* mechanism, which we use (among other things) to validate some of the axioms in Sec. 4 and 5. It erases the evidence that a type is inhabited by squashing it down to a single constant inhabitant using set types [14, pp.60]: $\downarrow T = \{x : \texttt{True} \mid T\}$. The only member of this type is the constant $\star$, which is `True`'s single inhabitant. The constant $\star$ inhabits $\downarrow T$ if T is true/inhabited, but we do not keep the proof that it is true. See Appx. C or [39] for more details on squashing.

In addition to the space squashing operator OpenTT also features another form of squashing called *time squashing.* As discussed in Sec. 2.3, some computations are *time-invariant*, while others, such as `wDepth`, are not. These two kinds of computations have different properties,[6] and this distinction should be captured at the level of types. To this end, OpenTT includes type constructors such as the time-squashing operator $\rotatebox[origin=c]{-90}{$\rightsquigarrow$}$. Given a type T, one can build the type $\rotatebox[origin=c]{-90}{$\rightsquigarrow$}T$, that in addition to T's members also contains terms that behave like members of T at a particular instant of time (in a particular world).

[5] For a precise definition of computational equivalence, see [25].

[6] E.g., if t is a time-invariant term that computes to a number $\underline{m}$ less than $\underline{n}$ in a world w, then t will also be less than $\underline{n}$ in all $w' \succeq w$. However, if t is a non-time-invariant number, t might be less than $\underline{n}$ in some extensions of w, and larger in others.

For the purpose of this paper, we only focus on a particular form of time-squashing for numbers, omitting the general construction.[7] Accordingly, OpenTT features a $\mathbb{N}_{\lightning}$ type of non-time-invariant (or time-squashed) numbers. While $\mathbb{N}$ is required to only be inhabited by time-invariant terms, $\mathbb{N}_{\lightning}$ is not, and allows for terms (such as `wDepth`) to compute to different numbers in different world extensions. For example, $\mathbb{N}_{\lightning}$ is allowed to be inhabited by a term t that computes to $\underline{3}$ in some world w, and to $\underline{4}$ in some world $w' \succeq w$. This distinction between $\mathbb{N}$ and $\mathbb{N}_{\lightning}$ will be critical in the validation of one of the choice sequence axioms in Sec. 4.1.2, where we make use of the depth of worlds which is not time-invariant.

In addition to the time-squashed $\mathbb{N}_{\lightning}$ type, OpenTT features a less than relation $t_1 <_{\lightning} t_2$ on time-squashed numbers, whose semantics is described in Sec. 3. Although similar to the $t_1 < t_2$ type, as for $\mathbb{N}_{\lightning}$, $t_1 <_{\lightning} t_2$ differs by not requiring t_1 and t_2 to be time-invariant.

3 Open Bar Realizability Model

This section presents a novel Beth-style model, called the open bar model, used below to validate OpenTT, which as mentioned above contains both a theory of choice sequences and a weak version of the classical LEM. As is standard in Beth models (or Kripke models [32, 33]), formulas are interpreted w.r.t. worlds. Using Beth models such as the one used in [8], a syntactic expression T is given meaning at a world w if there exists a collection B of worlds that covers all possible extensions of w, such that T corresponds to a legal type in all worlds in B. Such a collection is called *a bar* of w. In these models one has to construct such bars to prove that expressions are types or that types are inhabited. For example, to prove that choice sequences have type $\mathbb{N} \to \mathbb{N}$, given a choice sequence η and a number $\underline{n}$, one must exhibit a bar where $\eta(\underline{n})$ indeed computes to a number.

In this paper we take a different approach, one that avoids having to build bars altogether, and only requires building individual extensions of worlds. Intuitively, instead of requiring that a property P be true at a bar of a given world w, we require that for each extension w' of w, P holds for some extension of w'. Therefore, a major distinction between standard Beth models and our model is that in the former the semantics of a logical formula is computed based on the interpretation of that formula at a bar for the current world, while the latter only requires that in any possible extension of the current world there is always a further extension where the formula is given some meaning. Thus, our model only requires exhibiting *open bars* in the sense that not all infinite extensions of the current world necessarily have a finite prefix in the bar. Therefore, open bars are derivable from "standard" bars, but the converse does not hold. For the proof that choice sequences have type $\mathbb{N} \to \mathbb{N}$, this means that given an extension w' of the current world w, one must exhibit a further extension w'' where $\eta(\underline{n})$ computes to a number, which can be done by constructing w'' in which η contains at least $n + 1$ choices.[8] As mentioned, in standard Beth models, in addition to this construction one has to also construct the bar. Thus, the notion of open bars seems to provide a more relaxed connection between truth and constructions than in the traditional Beth-like interpretation of intuitionistic logic, where one must *construct* bars to establish validity. By not having to make the full construction, the open bar model provides some middle ground between classical and intuitionistic logic. Furthermore, note that in a standard Beth model, depending on how the bar is defined, it is not always possible to constructively exhibit a point in the

[7] See `per_qtime` in `per/per.v` for further details on $\lightning$'s sematics.

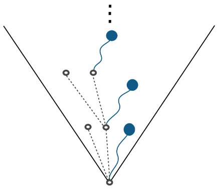
[8] See `rules/rules_choice1.v` for a proof of this statement.

bar, whereas in the open bar model, the existence of the open bar directly gives a point at the open bar. This makes the construction of building bars from other bars generally simpler.

We start by introducing the concept of open bars, which is used below to interpret types.

▶ **Definition 4** (Open Bars). *Let w be a world and f be a (metatheoretical) predicate on worlds. We say that f is true at an* open bar *of w if:*

$$\begin{aligned} &\mathcal{O}(w,f) = \forall_{\mathtt{EXT}}(w, \lambda w'.\exists_{\mathtt{EXT}}(w', \lambda w''.\forall_{\mathtt{EXT}}(w'', f))) \\ &\textit{where} \quad \forall_{\mathtt{EXT}}(w,f) = \forall w'.\ w' \geq w \Rightarrow f(w') \\ &\qquad\qquad \exists_{\mathtt{EXT}}(w,f) = \exists w'.\ w' \geq w \wedge f(w') \end{aligned}$$



Informally, an open bar can be thought of as an object such as the one depicted above. There, the large solid blue nodes indicate worlds which we already know to be at the bar, while the small hollow nodes indicate worlds not yet at the bar from which the open bar provides a way to obtain worlds at the bar. For example, given the root of the tree, the open bar might give us the lowest solid blue world w. Given a world w', such as the one left to w, where different choices have been made from w, we can ask the bar to produce another world at the bar compatible with w' (i.e., that extends w'), and we might get the middle solid blue world.

The open bar semantics bears resemblance to the well known double negation translation [23] in standard Kripke models [32, 33]. Informally, in Kripke interpretations, $A \to B$ is interpreted as follows: $[\![A \to B]\!]_w = \forall_{\mathtt{EXT}}(w, \lambda w'.[\![A]\!]_{w'} \Rightarrow [\![B]\!]_{w'})$. In such a semantics, the formula $\neg\neg A$ is then interpreted as $\forall_{\mathtt{EXT}}(w, \lambda w'.\neg\forall_{\mathtt{EXT}}(w', \lambda w''.\neg[\![A]\!]_{w''}))$, which is classically equivalent to $\forall_{\mathtt{EXT}}(w, \lambda w'.\exists_{\mathtt{EXT}}(w', \lambda w''.[\![A]\!]_{w''}))$. Nonetheless, our interpretation has two benefits over such a double negation translation: it is fully constructive, and it internalizes this double-negation/open-bar operator within the semantics, thereby avoiding having to use it explicitly in the theory. Note that this correspondence is unique to the *open* bar models, and does not hold in BITT's closed-bar model.

We now use open bars to provide meaning to OpenTT's types. As was done for similar theories [3, 4, 17, 6, 8], types are interpreted here by Partial Equivalence Relations (PERs) on closed terms. This PER semantics can be seen as an inductive-recursive definition of (see [20, 16] for similar construction methods):[9] (1) an inductive relation $T_1 {\equiv_w} T_2$ that expresses type equality; (2) a recursive function $t_1 {\equiv_w} t_2 {\in} T$ that expresses equality in a type. The inductive definition $T_1 {\equiv_w} T_2$ has one constructor per OpenTT type plus one additional constructor giving meaning to a type at a world w, based on its interpretation at an open bar of w (see Def. 6). Therefore, the recursive function $t_1 {\equiv_w} t_2 {\in} T$ has as many cases as there are constructors for $T {\equiv_w} T'$. The rest of this section presents some of these constructors and

[9] Due to the limited support for induction-recursion in Coq, our formalization instead combines these two definitions into a single inductive definition following the method described in [4, 13], which results in the same theory, however defined in a slightly more convoluted way that the one defined here.

cases that illustrate key aspects of the new semantics. For simplicity we present them as equivalences, which are derivable from the formal definition. The others are defined similarly in Appx. A or in `per/per.v`. We first define some useful abstractions.

▶ **Definition 5.** *A term t is said to* inhabit *or* realize *a type T at w if $t{\equiv_w}t{\in}T$. We further use the following notations:* $\mathtt{inh}(w,T)$ *for* $\exists t.\ t{\equiv_w}t{\in}T$*;* $a \Downarrow_w b$ *for 'a computes to b w.r.t. w', i.e., the reflexive and transitive closure of* $\mapsto$*; and* $a \Ddownarrow_w b$ *for* $\forall_{\mathtt{EXT}}(w, \lambda w'.a \Downarrow_{w'} b)$ *which captures that a is time-invariant.*[10]

As mentioned above, a key aspect of our open bar model is that it is defined to be closed under open bars, allowing interpreting all types and their PERs in terms of open bars.

▶ **Definition 6** (Open Bar Closure)**.** *OpenTT's semantics is closed under open bars as follows:*

$$
\begin{array}{lcl}
T_1{\equiv_w}T_2 & \iff & \mathcal{O}(w, \lambda w'.\exists T_1', T_2'.\ T_1 \Ddownarrow_{w'} T_1' \wedge T_2 \Ddownarrow_{w'} T_2' \wedge T_1'{\equiv_{w'}}T_2') \\
t_1{\equiv_w}t_2{\in}T & \iff & \mathcal{O}(w, \lambda w'.\exists T'.\ T \Ddownarrow_{w'} T' \wedge t_1{\equiv_{w'}}t_2{\in}T')
\end{array}
$$

Let us now turn to the semantics of key types of OpenTT under the open bar semantics. We start with demonstrating the $\mathbb{N}$ type which is in the core types of CTT.

▶ **Definition 7** (Time-Invariant Numbers)**.** *The $\mathbb{N}$ type is interpreted as follows:*

$$
\mathbb{N}{\equiv_w}\mathbb{N} \iff \mathtt{True} \qquad t{\equiv_w}t'{\in}\mathbb{N} \iff \mathcal{O}(w, \lambda w'.\exists \underline{n}.\ t \Ddownarrow_{w'} \underline{n} \wedge t' \Ddownarrow_{w'} \underline{n})
$$

Note the use of $\Ddownarrow$ above, since such numbers are required to be time-invariant (see Sec. 2.4).

In the next definition the time-invariant constraint is relaxed, allowing inhabitants of $\mathbb{N}_{\lightning}$ to compute to different numbers in different world extensions. For example, a term that computes to $\underline{3}$ in the current world w and to $\underline{4}$ in all (strict) extensions of w, inhabits $\mathbb{N}_{\lightning}$ but not $\mathbb{N}$. While $\mathbb{N}$ is a subtype of $\mathbb{N}_{\lightning}$, in the sense that all equal members of $\mathbb{N}$ are equal members of $\mathbb{N}_{\lightning}$, the converse does not hold. For example, `wDepth` is in $\mathbb{N}_{\lightning}$ but not in $\mathbb{N}$.

▶ **Definition 8** (Time-Squashed Numbers)**.** *The $\mathbb{N}_{\lightning}$ type is interpreted as follows:*

$$
\begin{array}{l}
\mathbb{N}_{\lightning}{\equiv_w}\mathbb{N}_{\lightning} \iff \mathtt{True} \qquad t{\equiv_w}t'{\in}\mathbb{N}_{\lightning} \iff \mathcal{O}(w, \lambda w'.\mathtt{sameNats}(w', t, t')) \\
\textit{where } \ \mathtt{sameNats}(w, t, t') = \exists k.\ t \Downarrow_w \underline{k} \wedge t' \Downarrow_w \underline{k}
\end{array}
$$

As mentioned in Sec. 2.4, in addition to the $\mathbb{N}_{\lightning}$ type, OpenTT also provides a "less-than" operator on such numbers, which we interpret as follows:

▶ **Definition 9** (Time-Squashed Less-Than)**.** *The $t_1 <_{\lightning} t_2$ type is interpreted as follows:*

$$
\begin{array}{lcl}
t_1 <_{\lightning} t_2{\equiv_w}t_1' <_{\lightning} t_2' & \iff & \mathcal{O}(w, \lambda w'.\mathtt{sameNats}(w', t_1, t_1') \ \wedge \ \mathtt{sameNats}(w', t_2, t_2')) \\
t{\equiv_w}t'{\in}t_1 <_{\lightning} t_2 & \iff & \mathcal{O}(w, \lambda w'.\exists k_1, k_2.\ t_1 \Downarrow_{w'} \underline{k_1} \wedge t_2 \Downarrow_{w'} \underline{k_2} \wedge k_1 < k_2)
\end{array}
$$

Note that given t_1 and t_2 in $\mathbb{N}_{\lightning}$ that compute to $\underline{3}$ and $\underline{4}$ respectively in *some* world, one cannot derive $t_1 <_{\lightning} t_2$ as t_1 and t_2 could keep alternating between $\underline{3}$ and $\underline{4}$ such that t_2 computes to $\underline{4}$ when t_1 computes to $\underline{3}$, and vice versa. Though general rules for inferring such inequalities can be formalized[11], in what follows we only need a concrete instance of $t_1 <_{\lightning} t_2$

[10] We here omit some technical details; see `ccomputes_to_valc_ext` in `per/per.v` for the full definition.

[11] Technically, our formalization includes both weakly monotonically increasing and decreasing numbers (denoted here $\mathbb{N}_{\lightning}^{\wedge}$ and $\mathbb{N}_{\lightning}^{\vee}$, respectively) allowing one to derive $t_1 <_{\lightning} t_2$ in w when $t_1 \in \mathbb{N}_{\lightning}^{\vee}$, $t_2 \in \mathbb{N}_{\lightning}^{\wedge}$, and t_2 computes to a number larger than t_1 in w.

in which $t_1 \in \mathbb{N}$ and $t_2 = \texttt{wDepth} \in \mathbb{N}_{\lightning}$ (see Sec. 4.1.2, which makes use of $\texttt{wDepth} \in \mathbb{N}_{\lightning}$ to capture the modulus of continuity of a predicate at a choice sequence). In this case such alternations are avoided since wDepth is weakly monotonically increasing.

OpenTT also includes a type of free choice sequences, interpreted as follows.

▶ **Definition 10** (Choice Sequences). *The* Free *type is interpreted as follows:*

$$\texttt{Free}{\equiv_w}\texttt{Free} \iff \texttt{True} \qquad t{\equiv_w}t'{\in}\texttt{Free} \iff \mathcal{O}(w, \lambda w'.\exists\eta.\ t \Downarrow_{w'} \eta \wedge t' \Downarrow_{w'} \eta)$$

As mentioned in Sec. 2.1, OpenTT includes a $t\#T$ type, which states that the term t is a sealed member of T. For example $\texttt{True}\#\mathbb{U}_i$, $\texttt{False}\#\mathbb{U}_i$, and $\mathbb{N}\#\mathbb{U}_i$ are all inhabited types, whereas $(\eta \in \texttt{Free})\#\mathbb{U}_i$ is not inhabited because this type mentions the choice sequence η. Note that $t\#T$ and $\texttt{synSealed}(t)$ did not appear in BITT.

▶ **Definition 11** (Free From Definitions). *The $a\#A$ type is interpreted as follows:*

$$\begin{aligned} a\#A{\equiv_w}b\#B &\iff A{\equiv_{w'}}B \wedge a{\equiv_{w'}}b{\in}A \\ t{\equiv_w}t'{\in}a\#A &\iff \mathcal{O}(w, \lambda w'.\exists x.\ a{\equiv_{w'}}x{\in}A \wedge \texttt{synSealed}(x)) \end{aligned}$$

As mentioned above, the other type operators of OpenTT are interpreted in a similar fashion. This semantics of OpenTT satisfies the following properties, which are the standard properties expected for such a semantics [3, 17], including the monotonocity and locality properties expected for a possible-world semantics [51, 21, 19] – heremonotonicity refers to types, and not to computations.[12]

▶ **Proposition 12** (Type System Properties). *The $T_1{\equiv_w}T_2$ and $a{\equiv_w}b{\in}T$ relations satisfy the following properties (where free variables are universally quantified):*

transitivity:	$T_1{\equiv_w}T_2 \Rightarrow T_2{\equiv_w}T_3 \Rightarrow T_1{\equiv_w}T_3$	$t_1{\equiv_w}t_2{\in}T \Rightarrow t_2{\equiv_w}t_3{\in}T \Rightarrow t_1{\equiv_w}t_3{\in}T$
symmetry:	$T_1{\equiv_w}T_2 \Rightarrow T_2{\equiv_w}T_1$	$t_1{\equiv_w}t_2{\in}T \Rightarrow t_2{\equiv_w}t_1{\in}T$
computation:	$T{\equiv_w}T \Rightarrow T \Downarrow_w T' \Rightarrow T{\equiv_w}T'$	$t{\equiv_w}t{\in}T \Rightarrow t \Downarrow_w t' \Rightarrow t{\equiv_w}t'{\in}T$
monotonicity:	$T_1{\equiv_w}T_2 \Rightarrow w' \succeq w \Rightarrow T_1{\equiv_{w'}}T_2$	$t_1{\equiv_w}t_2{\in}T \Rightarrow w' \succeq w \Rightarrow t_1{\equiv_{w'}}t_2{\in}T$
locality:	$\mathcal{O}(w, \lambda w'.T_1{\equiv_{w'}}T_2) \Rightarrow T_1{\equiv_w}T_2$	$\mathcal{O}(w, \lambda w'.t_1{\equiv_{w'}}t_2{\in}T) \Rightarrow t_1{\equiv_w}t_2{\in}T$

Using these properties, it follows that OpenTT is consistent w.r.t. the open bar model.

▶ **Theorem 13** (Soundness & Consistency). *OpenTT's inference rules are all sound w.r.t. the open bar model, which entails that OpenTT is consistent.*[13]

4 A Theory of Choice Sequences

This section focuses on OpenTT's inference rules that provide an axiomatization of a theory of choice sequences. This theory includes two variants of the Axiom of Open Data (Sec. 4.1.1 and 4.1.2), a density axiom (Sec. 4.2), and a discreteness axiom (Sec. 4.3). We focus our attention on the variants of the Axiom of Open Data that captures a form of continuity which is the core essence of choice sequences, as those where not handled in BITT.

[12] See `per/nuprl_props.v` for proofs of these properties.

[13] See `rules.v` and `per/weak_consistency.v` for more details.

4.1 The Axiom of Open Data (AOD)

The Axiom of Open Data (AOD) is perhaps the seminal axiom in the theory of choice sequences. It is a continuity axiom that states that the validity of properties of free choice sequences (with certain side conditions) can only depend on finite initial segments of these sequences. Let P be a sealed predicate on free choice sequences of numbers (i.e., $P\#(\texttt{Free} \rightarrow \mathbb{U}_i)$ for some universe i), $\mathbb{N}_n$ the type $\{x:\mathbb{N} \mid x < n\}$ of natural number strictly less than n, and $\mathcal{B}_n = \mathbb{N}_n \rightarrow \mathbb{N}$. The Axiom of Open Data can be formalized as follows:

$$\mathbf{\Pi}\alpha{:}\texttt{Free}.P(\alpha) \rightarrow \mathbf{\Sigma}n{:}\mathbb{N}.\mathbf{\Pi}\beta{:}\texttt{Free}.(\alpha =_{\mathcal{B}_n} \beta \rightarrow P(\beta)) \tag{AOD}$$

Since AOD is a form of continuity principle, and the non-squashed Continuity Principle is incompatible with CTT [39, 40] as well as with other computational theories [29, 46, 22], we only attempt to validate a squashed version of AOD. That is, since there is no way to compute the modulus of continuity of P at α, which is preserved over world extensions (as required by the semantics of $\mathbb{N}$), we instead validate versions of AOD where the sum type is squashed. But there are two ways to squash it, as described in Sec. 4.1.1 and 4.1.2.

There are two additional restrictions we impose in order to validate the squashed variants of AOD. First, to validate the axiom we swap α and β in $P(\alpha)$. This has an impact on both the PER of this type and the world w.r.t. which it is validated. Given an inhabitant t of $P(\alpha)$, we can easily build a proof of $P(\beta)$ by swapping α and β in t. This is however a metatheoretical operation. Therefore, in our variants of AOD the $P(\beta)$ is squashed. Second, note that when swapping one needs to swap α and β in all definitions and choice sequences' choices in the world w.r.t. which it is validated, leading to a different world. Therefore, we require that choice sequences cannot occur in definitions and choice sequences' choices to ensure that swapping α and β in a world w leads to an equivalent world if α and β have the same choices. To see why this is necessary take P to be the predicate $P = \lambda y.\{x : \texttt{Free} \mid x =_{\texttt{Free}} y\}$, and the world w to contain the definition $\delta == \alpha$. Then, $P(\alpha)$ is equivalent to $\{x : \texttt{Free} \mid x =_{\texttt{Free}} \alpha\}$ and δ is a member of $P(\alpha)$ in w, while $P(\beta)$ is equivalent to $\{x : \texttt{Free} \mid x =_{\texttt{Free}} \beta\}$ in this world, and therefore δ is not a member of $P(\beta)$ if α and β are two different choice sequences.

Before presenting and validating the variants of AOD, we present a few intermediate results. First, we prove that from $\alpha =_{\mathcal{B}_n} \beta$, we can always construct a world in which α and β contain exactly the same choices.[14]

▶ **Lemma 14** (Intermediate World). *Let w_1 and w_2 be two worlds such that $w_2 \succeq w_1$ and $\texttt{sing}(w_1)$ (see Def. 2). If η_1 and η_2 are two free choice sequences that have the same choices up to $|w_1|$ in w_2, then there must exist a world w, such that $w_2 \succeq w \succeq w_1$, both η_1 and η_2 occur in w, they have the exact same choice in w, and all these choices are numbers.*

Furthermore, the following swapping operator swaps α and β in $P(\alpha)$ to obtain $P(\beta)$.[15]

▶ **Definition 15** (Swapping). *Let $X{\cdot}(\eta_1|\eta_2)$ be a swapping operation that swaps η_1 and η_2 everywhere in X, where X ranges over all the syntactic forms presented above.*

We can then prove that the various relations introduced in Sec. 3 are preserved by the above swapping operator. For example, crucially, we can prove that the $t_1{\equiv_w}t_2{\in}T$ relation, which expresses that t_1 and t_2 are equal members in T, is preserved by swapping.[16]

[14] See Lemma `to_library_with_equal_cs` in `rules/rules_choice_util4.v`.

[15] See for example `swap_cs_term` in `terms/swap_cs.v`, which swaps two choice sequence names in a term.

[16] See `implies_equality_swap_cs` in `rules/rules_choice_util4.v` for the formal statement and proof.

▶ **Lemma 16** (Swapping PERs). *If* $t_1 {\equiv_w} t_2 {\in} T$ *then* $t_1{\cdot}(\eta_1|\eta_2) {\equiv_{w\cdot(\eta_1|\eta_2)}} t_2{\cdot}(\eta_1|\eta_2) {\in} T{\cdot}(\eta_1|\eta_2)$.

4.1.1 The Space-Squashed Axiom of Open Data ($\texttt{AOD}_{\downarrow}$)

The first variant of AOD we validate is the a *space-squashed* one, called $\texttt{AOD}_{\downarrow}$.

▶ **Proposition 17.** *The following rule of OpenTT is valid w.r.t. the open bar model (where H is an arbitrary list of hypotheses):*

$$\frac{}{H \vdash \mathbf{\Pi}\alpha{:}\texttt{Free}.P(\alpha) \to {\downarrow}\mathbf{\Sigma}n{:}\mathbb{N}.\mathbf{\Pi}\beta{:}\texttt{Free}.(\alpha =_{\mathcal{B}_n} \beta \to {\downarrow}P(\beta))}$$

Proof. We here outline the proof, see `rules/rules_ls3_v0.v` for full details. Since the sum type is $\downarrow$-squashed, a realizer for this formula can simply be $\lambda\alpha, x.\star$ (see Sec. 2.4). Let P be a sealed predicate on free choice sequences, α a free choice sequence, and instantiate n with $|w|$, the depth of the current world w. From $\alpha =_{\mathcal{B}_n} \beta$, we get that α and β have the same choices up to $|w|$ in the extension w' of w, and we have to show that $P(\beta)$ is true in w'. Lem. 14 entails that α and β have exactly the same choices in some world w'' between w and w'. Using Lem. 16 we swap α and β in $P(\alpha)$ and w''. Thus, because choice sequences cannot occur in definitions and choices, $P(\beta)$ is valid in a world equivalent to w'' and hence in w'' too.[17] Finally, using monotonicity (Lem. 12), we obtain that $P(\beta)$ is true also in w'. ◀

4.1.2 The Time-Squashed Axiom of Open Data ($\texttt{AOD}_{\lightning}$)

Next, we present a *time-squashed* version of `AOD`, where instead of $\downarrow$-squashing the sum type the $\mathbb{N}_{\lightning}$ time-squashed type is used, and $\mathcal{B}_{\lightning n} = \{x : \mathbb{N} \mid x <_{\lightning} n\} \to \mathbb{N}$ Is used instead of $\mathcal{B}_n$.[18]

$$\mathbf{\Pi}\alpha{:}\texttt{Free}.P(\alpha) \to \mathbf{\Sigma}n{:}\mathbb{N}_{\lightning}.\mathbf{\Pi}\beta{:}\texttt{Free}.(\alpha =_{\mathcal{B}_{\lightning n}} \beta \to {\downarrow}P(\beta)) \qquad (\texttt{AOD}_{\lightning})$$

Note that because n is not a member of $\mathbb{N}$ anymore but of $\mathbb{N}_{\lightning}$, we use $\mathcal{B}_{\lightning n}$ instead of $\mathcal{B}_n$ here to state that α and β are equal sequences up to n. If $n \in \mathbb{N}_{\lightning}$ then $x < n$, where $x \in \mathbb{N}$, and $\mathcal{B}_n$ are not types anymore: the semantics of $x < n$ requires both x and n to be time-invariant numbers (see Sec. 2.4). Therefore, we use $x <_{\lightning} n$ here instead, which does not require numbers to be time-invariant as per its semantics presented in Def. 9.

Before diving into the proof of $\texttt{AOD}_{\lightning}$'s validity, we first present a few intermediate results.

▶ **Lemma 18.** *The $\mathbb{N}$ type is a subtype of $\mathbb{N}_{\lightning}$, in the sense that all equal members in $\mathbb{N}$ are also equal members in $\mathbb{N}_{\lightning}$ (which implies that $t_1 <_{\lightning} t_2$ is a type even when $t_1 \in \mathbb{N}$ and $t_2 \in \mathbb{N}_{\lightning}$), and the* `wDepth` *expression is a member of $\mathbb{N}_{\lightning}$ (i.e., it is equal to itself in $\mathbb{N}_{\lightning}$).*[19] *I.e. the following rules are valid in OpenTT.*

$$\frac{H \vdash t_1 =_{\mathbb{N}} t_2}{H \vdash t_1 =_{\mathbb{N}_{\lightning}} t_2} \qquad\qquad \frac{}{H \vdash \texttt{wDepth} =_{\mathbb{N}_{\lightning}} \texttt{wDepth}}$$

[17] See Lemma `member_swapped_css_libs` in `rules/rules_choice_util4.v`.

[18] Note that as in $\texttt{AOD}_{\downarrow}$, $P(\beta)$ is also $\downarrow$-squashed here. We leave for future work to derive a version where $P(\beta)$ is not squashed. Note also that the modulus of continuity n is here in $\mathbb{N}_{\lightning}$. We have validated another version of this axiom in `rules/rules_ls3_v1.v` where $n \in \mathbb{N}^{\wedge}_{\lightning}$, i.e., where n is required to be weakly monotonically increasing, which is true about `wDepth` (see Sec. 2.3 and 2.4).

[19] See `rule_qnat_subtype_nat_true` in `rules/rules_ref.v` and `rule_depth_true` in `rules/rules_qnat.v`.

For $\texttt{AOD}_{\downarrow}$, because its $\mathbf{\Sigma}$ type is ↓-squashed, we did not have to provide a witness for the modulus of continuity of P at α. Instead, we could simply find a suitable metatheoretical number in the proof of its validity, without having to provide an expression from the object theory that computes that number. In the metatheoretical proof, we computed the depth of the current world, which is a metatheoretical number k, and simply used $\underline{k}$, which is a number in the object theory, as an approximation of the modulus of continuity of P at α. The situation is different in $\texttt{AOD}_{\downzigzagarrow}$ because the $\mathbf{\Sigma}$ type is no longer ↓-squashed. We now have to provide an expression from the object theory that computes that modulus of continuity. As mentioned, we use `wDepth`, which is an expression of *OpenTT*, the object theory. Thus, we now have to prove that `wDepth` has the right type, namely, $\mathbb{N}_{\downzigzagarrow}$, which we proved in Lem. 18.

Using these results we prove that $\texttt{AOD}_{\downzigzagarrow}$ is valid w.r.t. the semantics presented in Sec. 3.

▶ **Proposition 19.** *The following rule of OpenTT is valid w.r.t. the open bar model:*

$$\frac{}{H \vdash \mathbf{\Pi}\alpha{:}\texttt{Free}.P(\alpha) \rightarrow \mathbf{\Sigma}n{:}\mathbb{N}_{\downzigzagarrow}.\mathbf{\Pi}\beta{:}\texttt{Free}.(\alpha =_{\mathcal{B}_{\downzigzagarrow n}} \beta \rightarrow \downarrow P(\beta))}$$

Proof. We here outline the proof (which is similar to that of Prop.17), while full details are in `rules/rules_ls3_v2.v`. Since now the sum type is not ↓-squashed, we have to provide a witness for it. The realizer we provide for this formula is: $\lambda\alpha, x.\langle \texttt{wDepth}, \lambda\beta, y.\star\rangle$. Let P be a sealed predicate on free choice sequences, and let α be a free choice sequence. We now have to prove that $\texttt{wDepth} \in \mathbb{N}_{\downzigzagarrow}$, which follows from Lem. 18. Since `wDepth` computes to $|w|$, where w is the current world, we can then use $|w|$ as an approximation of the modulus of continuity of P at α, as in Prop. 17's proof. One difference with Prop. 17's proof is that we have here that $\alpha =_{\mathcal{B}_{\downzigzagarrow n}} \beta$ (which we prove to be a type using Lem. 18) instead of $\alpha =_{\mathcal{B}_n} \beta$. This however still suffices to show that α and β have the same choices up to $|w|$ in the extension w' of w. From here, the proof proceeds just as that of Prop. 17. ◀

4.2 The Density Axiom (DeA)

Another common free choice sequence axiom, sometimes called the *density* axiom [42], states that for any finite sequence of numbers f, there is a free choice sequence that contains f as initial segment (this is Axiom 2.1 in [30, Sec.2], also referred to as LS1 in [49]).

In BITT the following Density Axiom (DeA) was validated: $\mathbf{\Pi}n{:}\mathbb{N}.\mathbf{\Pi}f{:}\mathcal{B}_n.\mathbf{\Sigma}\alpha{:}\texttt{Free}.(f =_{\mathcal{B}_n} \alpha)$ [8]. The proof of its validity was by generating an appropriate choice sequence space that contains the values of the finite sequence f as part of its name. More precisely, given a finite sequence f of n terms in $\mathbb{N}$ from the object theory, BITT includes computations to extract those n numbers, say $\underline{k_1}, \dots, \underline{k_n}$, and build a choice sequence with the metatheoretical list of numbers $[k_1, \dots, k_n]$ as part of its name, and which is used to witness DeA's $\mathbf{\Sigma}$ type. In OpenTT we opted against including such names for two reasons. First, in the open bar model it is possible to validate a squashed version of DeA (where the $\mathbf{\Sigma}$ type is squashed) without including lists of numbers in choice sequence names. This is because the open bar model allows for internal choices to be made (see Prop. 20 below). Moreover, deterministically generating choice sequence names is not preserved by swapping (which would be required for example for Lem. 16 to hold). Given a term t that deterministically generates η_1, it might be that swapping η_1 for η_2 turns η_1 into η_2 and leaves t unchanged, while t does not generate η_2.

Therefore, we do not include metatheoretical lists of numbers as part of choice sequence names in OpenTT and only validate the following ↓-squashed version of DeA, called $\texttt{DeA}_{\downarrow}$.

▶ **Proposition 20.** *The following rule of OpenTT is valid w.r.t. the open bar model:*

$$\frac{}{H \vdash \mathbf{\Pi}n{:}\mathbb{N}.\mathbf{\Pi}f{:}\mathcal{B}_n.\downarrow\mathbf{\Sigma}\alpha{:}\texttt{Free}.(f =_{\mathcal{B}_n} \alpha)}$$

Proof. As this axiom is ↓-squashed, we realize it using $\lambda n, f.\star$. To prove its validity in some world w, assume $n \in \mathbb{N}$ and $f \in \mathcal{B}_n$ in some $w' \succeq w$. We have to exhibit some $w'' \succeq w'$ that contains a free choice sequence that has f as its initial segment. This world w'' can simply be w' augmented with a fresh (w.r.t. w') choice sequence that has f as its initial segment.[20] ◀

Note that the Beth model in [8] requires exhibiting a choice sequence such that DeA holds *at a bar* b of w. Without a mechanism to enforce initial segments, it could be that the choice sequence picked to witness α does not include the correct choices in some of b's branches. This is why BITT features choice sequence names that enforce initial segments. Thanks to open bars, OpenTT is able to do without enforcing initial segments within choice sequence names while still featuring a version of DeA, at the detriment of requiring its $\mathbf{\Sigma}$ type be ↓-squashed. (Troelstra calls the free choice sequences that enforce initial segments *lawless*, and the ones where no initial segment is enforced *proto-lawless* [42, Sec.2.4].)

4.3 The Discreteness Axiom (DiA)

One final common free choice sequence axiom, sometimes called the *discreteness* axiom [37], states that equality between free choice sequences is decidable (it is Axiom 2.2 in [30, Sec.2], also referred to as LS2 in [49]). As for BITT, OpenTT features intensional and extensional versions of the Discreteness Axiom (DiA), which we have proven to be valid w.r.t. the open bar model (we only present the extensional version here due to space constraints).[21]

▶ **Proposition 21.** *The following rule of OpenTT is valid w.r.t. the open bar model (the conclusion is inhabited by* $\lambda\alpha,\beta.\texttt{if}\ \alpha{=}\beta\ \texttt{then}\ \texttt{tt}\ \texttt{else}\ \texttt{ff}$*):*

$$\frac{}{H \vdash \mathbf{\Pi}\alpha,\beta{:}\texttt{Free}.\alpha =_{\mathcal{B}} \beta + \neg\alpha =_{\mathcal{B}} \beta}$$

5 The Law of Excluded Middle

This section demonstrates that OpenTT provides a key axiom from classical logic, namely the Law of Excluded Middle (LEM). Even though various other classical principles could be considered here (and will be considered in future work), we focus on LEM as it is the central axiom differentiating classical logic from intuitionistic logic. Thus, we show that in addition to capturing the intuitionistic concept of choice sequences, OpenTT also includes the following ↓-squashed version of LEM, called $\texttt{LEM}_{\downarrow}$, that is validated w.r.t. the open bar model: $\mathbf{\Pi}P{:}\mathbb{U}_i.{\downarrow}(P{+}\neg P)$.

For BITT, even this weak $\texttt{LEM}_{\downarrow}$ axiom, *that does not have any computational content* (as it is realized by $\lambda P.\star$), is inconsistent [8]. More precisely, $\neg\texttt{LEM}_{\downarrow}$ is valid w.r.t. the Beth metatheory presented in [8]. Intuitively, this is because $\texttt{LEM}_{\downarrow}$ states that there exists a bar of the current world such that either: (1) P is true at the bar, or (2) it is false in all extensions of the bar. This is false (i.e., the negation is true) because, for example, for $P = (\mathbf{\Sigma}n{:}\mathbb{N}.\eta(n) =_{\mathbb{N}} \underline{1})$, where η is a free choice sequence, (1) is false because η could be the sequence that never chooses 1, and (2) is false because there is an extension of the bar where η chooses 1. Stronger versions of this axiom, such as the non-↓-squashed version, are therefore also false. This counterexample for BITT does not serve as a counterexample for OpenTT because given a world w it is always possible to find an extension where η eventually

[20] See `rules/rules_choice1.v` for more details.

[21] See `rules/rules_choice2.v` and `rules/rules_choice5.v` for further details.

holds 1. Hence, OpenTT is more amenable to classical logic than theories based on standard Beth models, such as BITT. As illustrated in Prop. 22's proof below, intuitively, this is due to the fact that the open bar model implements a notion of time which allows to select futures (i.e., extensions), thereby allowing for some internal choices to be made.

▶ **Proposition 22.** *The following rule of OpenTT is valid w.r.t. the open bar model (using LEM in the metatheory).*

$$\frac{}{H \vdash \mathbf{\Pi} P{:}\mathbb{U}_i.{\downarrow}(P{+}\neg P)}$$

Proof. We have to show that for every world w' that extends the current world w, there exists a world w'' that extends w' such that $P{+}\neg P$ is inhabited in all extensions of w''. Let w' be an extension of w. We need to find a $w'' \succeq w'$ that makes the above true. Using classical logic we assume that $\exists_{\texttt{EXT}}(w', \lambda w''.\texttt{inh}(w'', P))$ is either true of false. If it is true, we obtain a $w'' \succeq w'$ at which P is inhabited, and we therefore conclude. Otherwise, we use w', which is a trivial extension of w'. We must now show that $P{+}\neg P$ is inhabited in all extensions of w'. We prove that it is inhabited by $\texttt{inr}(\star)$ by showing that in all $w'' \succeq w'$, P is not inhabited at w''. Assuming that P is inhabited at w'', we get that $\exists_{\texttt{EXT}}(w', \lambda w''.\texttt{inh}(w'', P))$ is true, which contradicts our assumption.[22] ◀

6 Conclusion and Related Work

The paper presents OpenTT, a novel intuitionistic type theory that features both a theory of choice sequences and a variant of the classical Law of Excluded Middle. This was made possible thanks to the open bar model, which internalizes a more relaxed notion of time than traditional Beth models that allows selecting futures. Thus, OpenTT provides a theoretical framework for studying the interplay between intuitionistic and classical logic.

Much work has been done on combining classical and constructive logics. One standard method is to use double negation translations [23] to embed classical logic in constructive logic. Another approach is to mix the two logics within the same framework. For example, PIL [35] mixes both logics through a polarization mechanism. Of particular relevance is Moschovakis's theory that includes choice sequences and is consistent with all classically true arithmetic sentences via a Kripke model [38].

As mentioned in the Introduction, there is a long line of work on providing intuitionistic counterexamples to classically valid axioms using variants of choice sequences. For example, in [15] Markov's Principle is proved to be false in a Martin-Löf type theory extended with a "generic" element, which behaves as a free choice sequence of Booleans. Since we have shown that OpenTT is compatible with a variant of LEM, we plan to investigate the status of other classically valid principles, such as Markov's Principle and the Axiom of Choice.

As for the open bar model, Kripke (and Beth) models are often used to model stateful theories. For example, in [34] the Kripke semantics of function types allows the returned values of functions to extend the state at hand. In contrast, the open bar model allows all computations to extend worlds. Other examples include [1, 2, 11, 10], where Kripke semantics are used to interpret theories with reference cells. We leave the study of other forms of stateful computations for future work.

[22] See `rules/rules_classical.v` for more details.

Unlike Kripke models, Beth models can interpret formulas that only *eventually* hold. The notion of "eventuality" in the open bar model slightly differs from the one in Beth models, and as hinted at in Sec. 3, is related to the "possibility" operator of modal logic [32]. A formal study of these connections is left for future work.

Several forms of choice sequence axioms have been studied in the literature. Some of them are currently time or space squashed in OpenTT. We plan on exploring versions of these axioms that are "less squashed" in the sense that they have more computational content.

Finally, the comprehensive account of choice sequences in OpenTT also opens the door for the exploration of the computational implications of the existence of such entities. For one, Brouwer used choice sequences to define the constructive real numbers as sequences of nested rational intervals. The computational account of choice sequences in OpenTT provides a framework for the formalization of Brouwerian constructive real analysis, and then comparing it to the more standard formalizations.

References

1 Amal J. Ahmed, Andrew W. Appel, and Roberto Virga. A stratified semantics of general references embeddable in higher-order logic. In *LICS*, page 75. IEEE Computer Society, 2002. doi:10.1109/LICS.2002.1029818.

2 Amal Jamil Ahmed. *Semantics of Types for Mutable State*. PhD thesis, Princeton University, 2004.

3 Stuart F. Allen. A non-type-theoretic definition of Martin-Löf's types. In *LICS*, pages 215–221. IEEE Computer Society, 1987.

4 Stuart F. Allen. *A Non-Type-Theoretic Semantics for Type-Theoretic Language*. PhD thesis, Cornell University, 1987.

5 Stuart F. Allen, Mark Bickford, Robert L. Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran. Innovations in computational type theory using Nuprl. *J. Applied Logic*, 4(4):428–469, 2006. doi:10.1016/j.jal.2005.10.005.

6 Abhishek Anand and Vincent Rahli. Towards a formally verified proof assistant. In Gerwin Klein and Ruben Gamboa, editors, *ITP 2014*, volume 8558 of *LNCS*, pages 27–44. Springer, 2014. doi:10.1007/978-3-319-08970-6_3.

7 Evert Willem Beth. *The foundations of mathematics: A study in the philosophy of science*. Harper and Row, 1966.

8 Mark Bickford, Liron Cohen, Robert L. Constable, and Vincent Rahli. Computability beyond church-turing via choice sequences. In Anuj Dawar and Erich Grädel, editors, *LICS 2018*, pages 245–254. ACM, 2018. doi:10.1145/3209108.3209200.

9 Mark Bickford, Liron Cohen, Robert L. Constable, and Vincent Rahli. Open bar — a brouwerian intuitionistic logic with a pinch of excluded middle. Extended version of our CSL 2021 paper avaible at https://vrahli.github.io/articles/open-bar-long.pdf, 2020.

10 Lars Birkedal, Bernhard Reus, Jan Schwinghammer, Kristian Støvring, Jacob Thamsborg, and Hongseok Yang. Step-indexed kripke models over recursive worlds. In Thomas Ball and Mooly Sagiv, editors, *POPL*, pages 119–132. ACM, 2011. doi:10.1145/1926385.1926401.

11 Lars Birkedal, Kristian Støvring, and Jacob Thamsborg. Realisability semantics of parametric polymorphism, general references and recursive types. *Mathematical Structures in Computer Science*, 20(4):655–703, 2010. doi:10.1017/S0960129510000162.

12 Douglas Bridges and Fred Richman. *Varieties of Constructive Mathematics*. London Mathematical Society Lecture Notes Series. Cambridge University Press, 1987. URL: http://books.google.com/books?id=oN5nsPkXhhsC.

13 Venanzio Capretta. A polymorphic representation of induction-recursion, 2004. URL: www.cs.ru.nl/~venanzio/publications/induction_recursion.ps.

14 Robert L. Constable, Stuart F. Allen, Mark Bromley, Rance Cleaveland, J. F. Cremer, Robert W. Harper, Douglas J. Howe, Todd B. Knoblock, Nax P. Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing mathematics with the Nuprl proof development system*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.

15 Thierry Coquand and Bassel Mannaa. The independence of markov's principle in type theory. In Delia Kesner and Brigitte Pientka, editors, *FSCD 2016*, volume 52 of *LIPIcs*, pages 17:1–17:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.FSCD.2016.17.

16 Thierry Coquand, Bassel Mannaa, and Fabian Ruch. Stack semantics of type theory. In *LICS 2017*, pages 1–11. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005130.

17 Karl Crary. *Type-Theoretic Methodology for Practical Programming Languages*. PhD thesis, Cornell University, Ithaca, NY, August 1998.

18 Jacques Dubucs and Michel Bourdeau. *Constructivity and Computability in Historical and Philosophical Perspective*, volume 34. Springer, January 2014. doi:10.1007/978-94-017-9217-2.

19 Michael A. E. Dummett. *Elements of Intuitionism*. Clarendon Press, second edition, 2000.

20 Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *J. Symb. Log.*, 65(2):525–549, 2000. URL: http://projecteuclid.org/euclid.jsl/1183746062.

21 VH Dyson and Georg Kreisel. *Analysis of Beth's semantic construction of intuitionistic logic*. Stanford University. Applied Mathematics and Statistics Laboratories, 1961.

22 Martín H. Escardó and Chuangjie Xu. The inconsistency of a Brouwerian continuity principle with the Curry-Howard interpretation. In *TLCA 2015*, volume 38 of *LIPIcs*, pages 153–164. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.TLCA.2015.153.

23 Gilda Ferreira and Paulo Oliva. On various negative translations. In Steffen van Bakel, Stefano Berardi, and Ulrich Berger, editors, *CL&C*, volume 47 of *EPTCS*, pages 21–33, 2010. doi:10.4204/EPTCS.47.4.

24 Arend Heyting. *Intuitionism: an introduction*. North-Holland Pub. Co., 1956.

25 Douglas J. Howe. Equality in lazy computation systems. In *LICS 1989*, pages 198–203. IEEE Computer Society, 1989.

26 Stephen C. Kleene and Richard E. Vesley. *The Foundations of Intuitionistic Mathematics, especially in relation to recursive functions*. North-Holland Publishing Company, 1965.

27 Alexei Kopylov and Aleksey Nogin. Markov's principle for propositional type theory. In *CSL 2001*, volume 2142 of *LNCS*, pages 570–584. Springer, 2001. doi:10.1007/3-540-44802-0_40.

28 Georg Kreisel. A remark on free choice sequences and the topological completeness proofs. *J. Symb. Log.*, 23(4):369–388, 1958. doi:10.2307/2964012.

29 Georg Kreisel. On weak completeness of intuitionistic predicate logic. *J. Symb. Log.*, 27(2):139–158, 1962. doi:10.2307/2964110.

30 Georg Kreisel. Lawless sequences of natural numbers. *Compositio Mathematica*, 20:222–248, 1968.

31 Georg Kreisel and Anne S. Troelstra. Formal systems for some branches of intuitionistic analysis. *Annals of Mathematical Logic*, 1(3):229–387, 1970. doi:10.1016/0003-4843(70)90001-X.

32 Saul A. Kripke. Semantical analysis of modal logic i. normal propositional calculi. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 9(5-6):67–96, 1963. doi:10.1002/malq.19630090502.

33 Saul A. Kripke. Semantical analysis of intuitionistic logic i. In J.N. Crossley and M.A.E. Dummett, editors, *Formal Systems and Recursive Functions*, volume 40 of *Studies in Logic and the Foundations of Mathematics*, pages 92–130. Elsevier, 1965. doi:10.1016/S0049-237X(08)71685-9.

34 Paul Blain Levy. Possible world semantics for general storage in call-by-value. In Julian C. Bradfield, editor, *CSL 2002*, volume 2471 of *LNCS*, pages 232–246. Springer, 2002. doi:10.1007/3-540-45793-3_16.

35 Chuck Liang and Dale Miller. Kripke semantics and proof systems for combining intuitionistic logic and classical logic. *Ann. Pure Appl. Log.*, 164(2):86–111, 2013. doi:10.1016/j.apal.2012.09.005.

36 Joan R. Moschovakis. An intuitionistic theory of lawlike, choice and lawless sequences. In *Logic Colloquium'90: ASL Summer Meeting in Helsinki*, pages 191–209. Association for Symbolic Logic, 1993.

37 Joan Rand Moschovakis. Choice sequences and their uses, 2015. URL: https://www.math.ucla.edu/~joan/stockholm2015handout.pdf.

38 Joan Rand Moschovakis. Intuitionistic analysis at the end of time. *Bulletin of Symbolic Logic*, 23(3):279–295, 2017. doi:10.1017/bsl.2017.25.

39 Vincent Rahli and Mark Bickford. A nominal exploration of intuitionism. In Jeremy Avigad and Adam Chlipala, editors, *CPP 2016*, pages 130–141. ACM, 2016. Extended version: http://www.nuprl.org/html/Nuprl2Coq/continuity-long.pdf. doi:10.1145/2854065.2854077.

40 Vincent Rahli and Mark Bickford. Validating brouwer's continuity principle for numbers using named exceptions. *Mathematical Structures in Computer Science*, pages 1—-49, 2017. doi:10.1017/S0960129517000172.

41 Vincent Rahli, Liron Cohen, and Mark Bickford. A verified theorem prover backend supported by a monotonic library. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22.*, volume 57 of *EPiC Series in Computing*, pages 564–582. EasyChair, 2018. URL: http://www.easychair.org/publications/paper/hp5j.

42 A. S. Troelstra. Analysing choice sequences. *J. Philosophical Logic*, 12(2):197–260, 1983. doi:10.1007/BF00247189.

43 Anne S. Troelstra. *Choice sequences: a chapter of intuitionistic mathematics*. Clarendon Press Oxford, 1977.

44 Anne S. Troelstra. Choice sequences and informal rigour. *Synthese*, 62(2):217–227, 1985.

45 A.S. Troelstra. *Choice Sequences: A Chapter of Intuitionistic Mathematics*. Clarendon Press, 1977.

46 A.S. Troelstra. A note on non-extensional operations in connection with continuity and recursiveness. *Indagationes Mathematicae*, 39(5):455–462, 1977. doi:10.1016/1385-7258(77)90060-9.

47 Mark van Atten. *On Brouwer*. Wadsworth Philosophers. Cengage Learning, 2004.

48 Mark van Atten and Dirk van Dalen. Arguments for the continuity principle. *Bulletin of Symbolic Logic*, 8(3):329–347, 2002. URL: http://www.math.ucla.edu/~asl/bsl/0803/0803-001.ps.

49 Dirk van Dalen. An interpretation of intuitionistic analysis. *Annals of mathematical logic*, 13(1):1–43, 1978.

50 Wim Veldman. Understanding and using Brouwer's continuity principle. In *Reuniting the Antipodes — Constructive and Nonstandard Views of the Continuum*, volume 306 of *Synthese Library*, pages 285–302. Springer Netherlands, 2001. doi:10.1007/978-94-015-9757-9_24.

51 Beth E. W. Semantic construction of intuitionistic logic. *Journal of Symbolic Logic*, 22(4):363–365, 1957.

A OpenTT's Semantics

Sec. 3 provided part of OpenTT's semantics. We presented there the semantics of distinguishing features of OpenTT. Let us now present the rest of its semantics. As mentioned in Sec. 3, this semantics has been formalized in Coq, and can be found in per/per.v and per/nuprl.v. Moreover, as the Coq formalization follows a slightly different presentation (as mentioned in

Sec. 3 it combines the inductive relation $T_1{\equiv_w}T_2$ and the recursive function $t_1{\equiv_w}t_2{\in}T$ into a single inductive definition following the method described in [4, 13]). An inductive-recursive formalization of the open bar semantics of OpenTT in Agda can be found in [9, Appx.D].

▶ **Definition 23** (Products). *Product types are interpreted as follows:*

$$\begin{aligned}
&\mathbf{\Pi}x_1{:}A_1.B_1{\equiv_w}\mathbf{\Pi}x_2{:}A_2.B_2\\
&\quad\iff \forall_{\texttt{EXT}}(w,\lambda w'.A_1{\equiv_{w'}}A_2 \wedge \forall a_1,a_2.\ a_1{\equiv_{w'}}a_2{\in}A_1 \Rightarrow B_1[x_1\backslash a_1]{\equiv_{w'}}B_2[x_2\backslash a_2])\\
&f_1{\equiv_w}f_2{\in}\mathbf{\Pi}x{:}A.B \iff \mathcal{O}(w,\lambda w'.\forall a_1,a_2.\ a_1{\equiv_{w'}}a_2{\in}A \Rightarrow f_1(a_1){\equiv_{w'}}f_2(a_2){\in}B[x_1\backslash a_1])
\end{aligned}$$

▶ **Definition 24** (Sums). *Sum types are interpreted as follows:*

$$\begin{aligned}
&\mathbf{\Sigma}x_1{:}A_1.B_1{\equiv_w}\mathbf{\Sigma}x_2{:}A_2.B_2\\
&\quad\iff \forall_{\texttt{EXT}}(w,\lambda w'.A_1{\equiv_{w'}}A_2 \wedge \forall a_1,a_2.\ a_1{\equiv_{w'}}a_2{\in}A_1 \Rightarrow B_1[x_1\backslash a_1]{\equiv_{w'}}B_2[x_2\backslash a_2])\\
&t_1{\equiv_w}t_2{\in}\mathbf{\Sigma}x{:}A.B \iff \mathcal{O}(w,\lambda w'.\exists a_1,a_2,b_1,b_2.\ t_1 \Downarrow_{w'} \langle a_1,b_1\rangle \wedge t_2 \Downarrow_{w'} \langle a_2,b_2\rangle \qquad)\\
&\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ a_1{\equiv_{w'}}a_2{\in}A \wedge b_1{\equiv_{w'}}b_2{\in}B[x_1\backslash a_1]
\end{aligned}$$

▶ **Definition 25** (Universes). *To interpret universes, we need to use parameterized (by a universe level)* $T_1{\equiv_{i,w}}T_2$ *and* $t_1{\equiv_{i,w}}t_2{\in}T$ *relations instead of the ones used so far. We can then define* $T_1{\equiv_w}T_2$ *as* $\exists i.\ T_1{\equiv_{i,w}}T_2$ *and* $t_1{\equiv_w}t_2{\in}T$ *as* $\exists i.\ t_1{\equiv_{i,w}}t_2{\in}T$. *We do not present the full construction here as it is standard. However, let us point out that using the above definitions we can then interpret universes inductively over* i, *resulting in the following interpretations:*

$$\mathbb{U}_i{\equiv_{j,w}}\mathbb{U}_i \iff i < j \qquad\qquad T_1{\equiv_{j,w}}T_2{\in}\mathbb{U}_i \iff T_1{\equiv_{j,w}}T_2$$

▶ **Definition 26** (Equality). *Equality types are interpreted as follows:*

$$\begin{aligned}
&(a_1 = a_2 \in A){\equiv_w}(b_1 = b_2 \in B) \iff A{\equiv_w}B \wedge a_1{\equiv_w}b_1{\in}A \wedge a_2{\equiv_w}b_2{\in}A\\
&t_1{\equiv_w}t_2{\in}(a = b \in A) \iff \mathcal{O}(w,\lambda w'.t_1 \Downarrow_{w'} \star \wedge t_2 \Downarrow_{w'} \star \wedge a{\equiv_{w'}}b{\in}A)
\end{aligned}$$

▶ **Definition 27** (Disjoint Union). *Disjoint union types are interpreted as follows:*

$$\begin{aligned}
&A_1{+}A_2{\equiv_w}B_1{+}B_2 \iff A_1{\equiv_w}B_1 \wedge A_2{\equiv_w}B_2\\
&t_1{\equiv_w}t_2{\in}A{+}B \iff \mathcal{O}(w,\lambda w'.(\exists x,y.\ t_1 \Downarrow_{w'} \texttt{inl}(x) \wedge t_2 \Downarrow_{w'} \texttt{inl}(y) \wedge x{\equiv_{w'}}y{\in}A) \qquad)\\
&\qquad\qquad\qquad\qquad\qquad \vee\ (\exists x,y.\ t_1 \Downarrow_{w'} \texttt{inr}(x) \wedge t_2 \Downarrow_{w'} \texttt{inr}(y) \wedge x{\equiv_{w'}}y{\in}B)
\end{aligned}$$

▶ **Definition 28** (Sets). *Set types are interpreted as follows:*

$$\begin{aligned}
&\{x_1 : A_1 \mid B_1\}{\equiv_w}\{x_2 : A_2 \mid B_2\}\\
&\quad\iff \forall_{\texttt{EXT}}(w,\lambda w'.A_1{\equiv_{w'}}A_2 \wedge \forall a_1,a_2.\ a_1{\equiv_{w'}}a_2{\in}A_1 \Rightarrow B_1[x_1\backslash a_1]{\equiv_{w'}}B_2[x_2\backslash a_2])\\
&t_1{\equiv_w}t_2{\in}\{x : A \mid B\} \iff \mathcal{O}(w,\lambda w'.t_1{\equiv_{w'}}t_2{\in}A \wedge \texttt{inh}(w',B[x\backslash t_1]))
\end{aligned}$$

▶ **Definition 29** (Less Than). *Less than types are interpreted as follows:*

$$\begin{aligned}
&t_1 < t_2{\equiv_w}u_1 < u_2 \iff t_1{\equiv_w}u_1{\in}\mathbb{N} \wedge t_2{\equiv_w}u_2{\in}\mathbb{N}\\
&t_1{\equiv_w}t_2{\in}(u_1 < u2) \iff \mathcal{O}(w,\lambda w'.\exists k_1,k_2.\ t_1 \Downarrow_w \underline{k_1} \wedge t_2 \Downarrow_w \underline{k_2} \wedge k_1 < k_2)
\end{aligned}$$

The time squashing type ↯T is defined using Howe's computational equivalence [25], which is omitted from this paper for space reasons (see [25] for a definition of this relation, as well as `cequiv/cequiv.v`). It turns out that OpenTT is not only closed under computation but more generally under Howe's computational equivalence $\sim$, which we have proved to be a congruence following Howe's method [25]. We define $t_1 \approx_w t_2$ as $\forall_{\texttt{EXT}}(w,\lambda w'.t_1 \sim_w t_2)$.

▶ **Definition 30** (Time Squashing). *Time squashing types are interpreted as follows:*

$$\lightning T \equiv_w \lightning U \iff T \equiv_w U \in \mathbb{N}$$

$$t_1 \equiv_w t_2 \in (\lightning T) \iff \mathcal{O}(w, \lambda w'.\exists u_1, u_2.\ w \sim t_1 u_1 \wedge w \sim t_2 u_2 \wedge t_1 \approx_w t_2 \wedge u_1 \equiv_{w'} u_2 \in T)$$

B OpenTT's Inference Rules

In OpenTT, sequents are of the form $h_1, \ldots, h_n \vdash T \lfloor \textbf{ext } t \rfloor$. Such a sequent denotes that, assuming $h_1, \ldots, h_n$, the term t is a member of the type T, and that therefore T is a type. The term t in this context is called the *extract* or *evidence* of T. Extracts are sometimes omitted when irrelevant to the discussion. In particular, we typically do so when the conclusion T of a sequent is an equality type of the form $a = b \in A$, since equality types can only be inhabited by the constant $\star$, we then typically omit the extract in such sequents. An hypothesis h is of the form $x : A$, where the variable x stands for the name of the hypothesis and A its type. A rule is a pair of a conclusion sequent S and a list of premise sequents, $S_1, \cdots, S_n$ (written as usual using a fraction notation, with the premises on top). Let us now provide a sample of OpenTT's key inference rules for some of its types not discussed above. The reader is invited to check `https://github.com/vrahli/NuprlInCoq/blob/ls3/` for a complete list of rules, as well as [14], from which OpenTT borrowed most of its rules for its standard types.

B.1 Products

The following rules are the standard $\mathbf{\Pi}$-elimination rule, $\mathbf{\Pi}$-introduction rule, type equality for $\mathbf{\Pi}$ types, and λ-introduction rule, respectively.

$$\frac{H, f : \mathbf{\Pi} x{:}A.B, J \vdash a \in A \quad H, f : \mathbf{\Pi} x{:}A.B, J, z : f(a) \in B[x \backslash a] \vdash C \lfloor \textbf{ext } e \rfloor}{H, f : \mathbf{\Pi} x{:}A.B, J \vdash C \lfloor \textbf{ext } e[z \backslash \star] \rfloor}$$

$$\frac{H, z : A \vdash B[x \backslash z] \lfloor \textbf{ext } b \rfloor \quad H \vdash A \in \mathbb{U}_i}{H \vdash \mathbf{\Pi} x{:}A.B \lfloor \textbf{ext } \lambda z.b \rfloor}$$

$$\frac{H \vdash A_1 = A_2 \in \mathbb{U}_i \quad H, y : A_1 \vdash B_1[x_1 \backslash y] = B_2[x_2 \backslash y] \in \mathbb{U}_i}{H \vdash \mathbf{\Pi} x_1{:}A_1.B_1 = \mathbf{\Pi} x_2{:}A_2.B_2 \in \mathbb{U}_i}$$

$$\frac{H, z : A \vdash t_1[x_1 \backslash z] = t_2[x_2 \backslash z] \in B[x \backslash z] \quad H \vdash A \in \mathbb{U}_i}{H \vdash \lambda x_1.t_1 = \lambda x_2.t_2 \in \mathbf{\Pi} x{:}A.B}$$

Note that the last rule requires to prove that A is a type because the conclusion requires to prove that $\mathbf{\Pi} x{:}A.B$ is a type, and the first hypothesis only states that B is a type family over A, but does not ensures that A is a type.

The following rule is the standard function extensionality rule:

$$\frac{H, z : A \vdash f_1(z) = f_2(z) \in B[x \backslash z] \quad H \vdash A \in \mathbb{U}_i}{H \vdash f_1 = f_2 \in \mathbf{\Pi} x{:}A.B}$$

The following captures that PERs are closed under β-reductions:

$$\frac{H \vdash t[x \backslash s] = u \in T}{H \vdash (\lambda x.t)\ s = u \in T}$$

B.2 Sums

The following rules are the standard $\mathbf{\Sigma}$-elimination rule, $\mathbf{\Sigma}$-introduction rule, type equality for the $\mathbf{\Sigma}$ type, and pair-introduction rule, respectively.

$$\frac{H, p : \mathbf{\Sigma}x{:}A.B, a : A, b : B[x\backslash a], J[p\backslash\langle a, b\rangle] \vdash C[p\backslash\langle a, b\rangle] \ \lfloor \mathbf{ext}\ e \rfloor}{H, p : \mathbf{\Sigma}x{:}A.B, J \vdash C \ \lfloor \mathbf{ext}\ \mathtt{let}\ a, b = p\ \mathtt{in}\ e \rfloor}$$

$$\frac{H \vdash a \in A \quad H \vdash b \in B[x\backslash a] \quad H, z : A \vdash B[x\backslash z] \in \mathbb{U}_i}{H \vdash \mathbf{\Sigma}x{:}A.B \ \lfloor \mathbf{ext}\ \langle a, b\rangle \rfloor}$$

$$\frac{H \vdash A_1 = A_2 \in \mathbb{U}_i \quad H, y : A_1 \vdash B_1[x_1\backslash y] = B_2[x_2\backslash y] \in \mathbb{U}_i}{H \vdash \mathbf{\Sigma}x_1{:}A_1.B_1 = \mathbf{\Sigma}x_2{:}A_2.B_2 \in \mathbb{U}_i}$$

$$\frac{H, z : A \vdash B[x\backslash z] \in \mathbb{U}_i \quad H \vdash a_1 = a_2 \in A \quad H \vdash b_1 = b_2 \in B[x\backslash a_1]}{H \vdash \langle a_1, b_1\rangle = \langle a_2, b_2\rangle \in \mathbf{\Sigma}x{:}A.B}$$

The following rule states that PERs are closed under spread-reductions:

$$\frac{H \vdash u[x\backslash s; y\backslash t] = t_2 \in T}{H \vdash \mathtt{let}\ x, y = \langle s, t\rangle\ \mathtt{in}\ u = t_2 \in T}$$

B.3 Equality

The following rules are the standard equality-introduction rule:[23], equality-elimination rule (which states that equality types are inhabited by the $\star$ constant), hypothesis rule, symmetry and transitivity rules, respectively.

$$\frac{H \vdash A = B \in \mathbb{U}_i \quad H \vdash a_1 = b_1 \in A \quad H \vdash a_2 = b_2 \in B}{H \vdash a_1 = a_2 \in A = b_1 = b_2 \in B \in \mathbb{U}_i}$$

$$\frac{H, z : a = b \in A, J[z\backslash\star] \vdash C[z\backslash\star] \ \lfloor \mathbf{ext}\ e \rfloor}{H, z : a = b \in A, J \vdash C \ \lfloor \mathbf{ext}\ e \rfloor}$$

$$\frac{}{H, x : A, J \vdash x \in A}$$

$$\frac{H \vdash b = a \in T}{H \vdash a = b \in T} \qquad \frac{H \vdash a = c \in T \quad H \vdash c = b \in T}{H \vdash a = b \in T}$$

The following rule allows fixing the extract of a sequent:

$$\frac{H \vdash T \ \lfloor \mathbf{ext}\ t \rfloor}{H \vdash t \in T}$$

The following rule allows rewriting with an equality in an hypothesis:

$$\frac{H, x : B, J \vdash C \ \lfloor \mathbf{ext}\ t \rfloor \quad H \vdash A = B \in \mathbb{U}_i}{H, x : A, J \vdash C \ \lfloor \mathbf{ext}\ t \rfloor}$$

[23] The actual rule is slightly more general as it allows a_1 and b_1 to be "computationally equivalent" (and similarly for a_2 and b_2). However, since we have not introduced this concept here, we present a simpler version of this rule only.

B.4 Universes

Let i is a lower universe than j. The following rules are the standard universe-introduction rule and the universe cumulativity rule, respectively.

$$\frac{}{H \vdash \mathbb{U}_i = \mathbb{U}_i \in \mathbb{U}_j} \qquad \frac{H \vdash T \in \mathbb{U}_j}{H \vdash T \in \mathbb{U}_i}$$

B.5 Sets

The following rule is the standard set-elimination rule:

$$\frac{H, z : \{x : A \mid B\}, a : A, \boxed{b : B[x\backslash a]}, J[z\backslash a] \vdash C[z\backslash a] \ \lfloor \mathbf{ext}\ e \rfloor}{H, z : \{x : A \mid B\}, J \vdash C \ \lfloor \mathbf{ext}\ e[a\backslash z] \rfloor}$$

Note that we have used a new construct in the above rule, namely the hypothesis $\boxed{b : B[x\backslash a]}$, which is called a hidden hypothesis. The main feature of hidden hypotheses is that their names cannot occur in extracts (which is why we “box” those hypotheses). Intuitively, this is because the proof that B is true is discarded in the proof that the set type $\{x : A \mid B\}$ is true and therefore cannot occur in computations. Hidden hypotheses can be unhidden using the following rule:

$$\frac{H, x : T, J \vdash a = b \in A \ \lfloor \mathbf{ext}\ \star \rfloor}{H, \boxed{x : T}, J \vdash a = b \in A \ \lfloor \mathbf{ext}\ \star \rfloor}$$

which is valid since the extract is $\star$ and therefore does not make use of x.

The following rules are the standard set-introduction rule, type equality for the set type, and introduction rule for members of set types, respectively.

$$\frac{H \vdash a \in A \quad H \vdash B[x\backslash a] \quad H, z : A \vdash B[x\backslash z] \in \mathbb{U}_i}{H \vdash \{x : A \mid B\} \ \lfloor \mathbf{ext}\ a \rfloor}$$

$$\frac{H \vdash A_1 = A_2 \in \mathbb{U}_i \quad H, y : A_1 \vdash B_1[x_1\backslash y] = B_2[x_2\backslash y] \in \mathbb{U}_i}{H \vdash \{x_1 : A_1 \mid B_1\} = \{x_2 : A_2 \mid B_2\} \in \mathbb{U}_i}$$

$$\frac{H, z : A \vdash B[x\backslash z] \in \mathbb{U}_i \quad H \vdash a = b \in A \quad H \vdash B[x\backslash a]}{H \vdash a = b \in \{x : A \mid B\}}$$

B.6 Disjoint Unions

The following rules are the standard disjoint union-elimination rule, disjoint union-introduction rules, type equality for the disjoint union type, and injection-introduction rules, respectively.

$$\frac{\begin{array}{c} H, d : A{+}B, x : A, J[d\backslash \mathtt{inl}(x)] \vdash C[d\backslash \mathtt{inl}(x)] \ \lfloor \mathbf{ext}\ t \rfloor \\ H, d : A{+}B, y : B, J[d\backslash \mathtt{inr}(y)] \vdash C[d\backslash \mathtt{inr}(y)] \ \lfloor \mathbf{ext}\ u \rfloor \end{array}}{H, d : A{+}B, J \vdash C \ \lfloor \mathbf{ext}\ \mathtt{case}\ d\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t \mid \mathtt{inr}(y) \Rightarrow u \rfloor}$$

$$\frac{H \vdash A \ \lfloor \mathbf{ext}\ a \rfloor \quad H \vdash B \in \mathbb{U}_i}{H \vdash A{+}B \ \lfloor \mathbf{ext}\ \mathtt{inl}(a) \rfloor} \qquad \frac{H \vdash B \ \lfloor \mathbf{ext}\ b \rfloor \quad H \vdash A \in \mathbb{U}_i}{H \vdash A{+}B \ \lfloor \mathbf{ext}\ \mathtt{inr}(b) \rfloor}$$

$$\frac{H \vdash A_1 = A_2 \in \mathbb{U}_i \quad H \vdash B_1 = B_2 \in \mathbb{U}_i}{H \vdash A_1{+}B_1 = A_2{+}B_2 \in \mathbb{U}_i}$$

$$\frac{H \vdash a_1 = a_2 \in A \quad H \vdash B \in \mathbb{U}_i}{H \vdash \mathtt{inl}(a_1) = \mathtt{inl}(a_2) \in A{+}B} \qquad \frac{H \vdash b_1 = b_2 \in B \quad H \vdash A \in \mathbb{U}_i}{H \vdash \mathtt{inr}(b_1) = \mathtt{inr}(b_2) \in A{+}B}$$

The following rules state that PERs are closed under decide-reductions:

$$\frac{H \vdash t[x\backslash s] = t_2 \in T}{H \vdash (\mathtt{case}\ \mathtt{inl}(s)\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t \mid \mathtt{inr}(y) \Rightarrow u) = t_2 \in T}$$

$$\frac{H \vdash u[y\backslash s] = t_2 \in T}{H \vdash (\mathtt{case}\ \mathtt{inr}(s)\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t \mid \mathtt{inr}(y) \Rightarrow u) = t_2 \in T}$$

C Squashing

As mentioned in Sec. 2.4, OpenTT includes a *squashing* mechanism, which is used to erase the computational content of a type by turning its PER into a trivial one.[24] More precisely, given a type T, the type $\downarrow T$, defined as $\{x : \mathtt{True} \mid T\}$, is true iff T is true. However, while the type T might have a trivial PER, i.e., it might be inhabited by arbitrarily complex programs, $\downarrow T$ can only be inhabited by $\star$, which is $\mathtt{True}$'s only inhabitant. Indeed, as shown in Def. 28 and Appx. B.5, a member of $\{x : \mathtt{True} \mid T\}$ is a member of $\mathtt{True}$, such that T is true. However, T's realizer is thrown away and is not part of $\{x : \mathtt{True} \mid T\}$'s realizer.

More precisely, one can derive $\downarrow T$ from T because given a member t of T, one can trivially show that that $\star$ is a member of $\downarrow T$. We can capture this by the following derived rule:

$$\frac{H \vdash T\ \lfloor \mathbf{ext}\ t \rfloor}{H \vdash \downarrow T\ \lfloor \mathbf{ext}\ \star \rfloor}$$

However, the opposite is not true in general. One cannot in general derive T from $\downarrow T$ because given the realizer $\star$ of $\downarrow T$, it is not always possible to recover a realizer of T. We can capture this by the following derived rule:

$$\frac{H, z : \downarrow T, \boxed{x : T}, J[z\backslash\star] \vdash C[z\backslash\star]\ \lfloor \mathbf{ext}\ e \rfloor}{H, z : \downarrow T, J \vdash C\ \lfloor \mathbf{ext}\ e \rfloor}$$

To illustrate the point that we cannot in general derive T from $\downarrow T$, let us see how far we can go when trying to prove:

$$x : \downarrow T \vdash T$$

Using the above squash-elimination derived rule, we have to prove:

$$x : \downarrow T, \boxed{z : T} \vdash T$$

However, we are now stuck, as we have in general no way of deriving an extract of T given these hypotheses. The unhiding rule mentioned Appx. B.5 can only be used when the conclusion is an equality type, and the hypothesis rule mentioned in Appx. B.3, requires the z hypothesis to be "visible" (not hidden) in order to use z as a realizer of the conclusion.

[24] See for example [39] for more details on squashing.

Discounted-Sum Automata with Multiple Discount Factors

Udi Boker

The Interdisciplinary Center, Herzliya, Israel
udiboker@idc.ac.il

Guy Hefetz

The Interdisciplinary Center, Herzliya, Israel
ghefetz@gmail.com

Abstract

Discounting the influence of future events is a key paradigm in economics and it is widely used in computer-science models, such as games, Markov decision processes (MDPs), reinforcement learning, and automata. While a single game or MDP may allow for several different discount factors, discounted-sum automata (NDAs) were only studied with respect to a single discount factor. For every integer $\lambda \in \mathbb{N} \setminus \{0, 1\}$, as opposed to every $\lambda \in \mathbb{Q} \setminus \mathbb{N}$, the class of NDAs with discount factor λ (λ-NDAs) has good computational properties: it is closed under determinization and under the algebraic operations min, max, addition, and subtraction, and there are algorithms for its basic decision problems, such as automata equivalence and containment.

We define and analyze discounted-sum automata in which each transition can have a different integral discount factor (integral *NMDAs*). We show that integral NMDAs with an arbitrary choice of discount factors are not closed under determinization and under algebraic operations. We then define and analyze a restricted class of integral NMDAs, which we call *tidy NMDAs*, in which the choice of discount factors depends on the prefix of the word read so far. Tidy NMDAs are as expressive as deterministic integral NMDAs with an arbitrary choice of discount factors, and some of their special cases are NMDAs in which the discount factor depends on the action (alphabet letter) or on the elapsed time.

We show that for every function θ that defines the choice of discount factors, the class of θ-NMDAs enjoys all of the above good properties of integral NDAs, as well as the same complexities of the required decision problems. To this end, we also improve the previously known complexities of the decision problems of integral NDAs, and present tight bounds on the size blow-up involved in algebraic operations on them.

All our results hold equally for automata on finite words and for automata on infinite words.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Automata, Discounted-sum, Quantitative verification, NMDA, NDA

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.12

Related Version A full version of the paper is available at `https://www.idc.ac.il/en/schools/cs/research/documents/thesis-guy-hefetz.pdf` ([24]).

Funding *Udi Boker*: Israel Science Foundation grant 1373/16.

1 Introduction

Discounted summation is a central valuation function in various computational models, such as games (e.g., [39, 2, 17]), Markov decision processes (e.g, [23, 30, 15]), reinforcement learning (e.g, [34, 37]), and automata (e.g, [19, 11, 13, 14]), as it formalizes the concept that an immediate reward is better than a potential one in the far future, as well as that a potential problem (such as a bug in a reactive system) in the far future is less troubling than a current one.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 12; pp. 12:1–12:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A Nondeterministic Discounted-sum Automaton (NDA) is an automaton with rational weights on the transitions, and a fixed rational discount factor $\lambda > 1$. The value of a (finite or infinite) run is the discounted summation of the weights on the transitions, such that the weight in the ith position of the run is divided by λ^i. The value of a (finite or infinite) word is the minimal value of the automaton runs on it. An NDA $\mathcal{A}$ realizes a function from words to real numbers, and we write $\mathcal{A}(w)$ for the value of $\mathcal{A}$ on a word w.

In the Boolean setting, where automata realize languages, closure under the basic Boolean operations of union, intersection, and complementation is desirable, as it allows to use automata in formal verification, logic, and more. In the quantitative setting, where automata realize functions from words to numbers, the above Boolean operations are naturally generalized to algebraic ones: union to min, intersection to max, and complementation to multiplication by -1 (depending on the function's co-domain). Likewise, closure under these algebraic operations, as well as under addition and subtraction, is desirable for quantitative automata, serving for quantitative verification. Determinization is also very useful in automata theory, as it gives rise to many algorithmic solutions, and is essential for various tasks, such as synthesis and probabilistic model checking[1].

NDAs cannot always be determinized [14], they are not closed under basic algebraic operations [6], and basic decision problems on them, such as universality, equivalence, and containment, are not known to be decidable and relate to various longstanding open problems [7]. However, restricting NDAs to an integral discount factor $\lambda \in \mathbb{N}$ provides a robust class of automata that is closed under determinization and under the algebraic operations, and for which the decision problems of universality equivalence, and containment are decidable [6].

Various variants of NDAs are studied in the literature, among which are *functional*, *k-valued*, *probabilistic*, and more [21, 20, 12]. Yet, to the best of our knowledge, all of these models are restricted to have a single discount factor in an automaton. This is a significant restriction of the general discounted-summation paradigm, in which multiple discount factors are considered. For example, Markov decision processes and discounted-sum games allow for multiple discount factors within the same entity [23, 2].

A natural extension to NDAs is to allow for different discount factors over the transitions, providing the ability to model systems in which each action (alphabet letter in the automaton) causes a different discounting, systems in which the discounting changes over time, and more. As integral NDAs provide robust automata classes, whereas non-integral NDAs do not, we look into extending integral NDAs into integral *NMDAs* (Definition 1), allowing multiple integral discount factors in a single automaton.

As automata are aimed at modeling systems, NMDAs significantly extend the system behaviors that can be modeled with discounted-sum automata. For an intuitive example, consider how the value of used vehicles changes over time: It decreases a lot in the first year, slightly less rapidly in the next couple of years, and significantly less rapidly in further years. An NDA cannot model such a behavior, as the discount factor cannot change over time, whereas an NMDA provides the necessary flexibility of the discount factor.

On a more formal level, NMDAs may allow to enhance formal verification of reinforcement learning applications. In the reinforcement learning process, the expected return value is the discounted-summation of the accumulated future rewards. In classic reinforcement learning, the discounted summation uses a single discount factor, whereas novel approaches

[1] In some cases, automata that are "almost deterministic", such as limit-deterministic [36] or good-for-games automata [25, 8] suffice.

in reinforcement learning study how to enhance the process to allow multiple discount factors [28, 22, 37, 32, 27]. This enhancement of reinforcement learning parallels our extension of discounted-sum automata to support multiple discount factors.

We start with analyzing NMDAs in which the integral discount factors can be chosen arbitrarily. Unfortunately, we show that this class of automata does not allow for determinization and is not closed under the basic algebraic operations.

For more restricted generalizations of integral NDAs, in which the discount factor depends on the transition's letter (*letter-oriented* NMDAs) or on the elapsed time (*time-oriented* NMDAs), we show that the corresponding automata classes do enjoy all of the good properties of integral NDAs, while strictly extending their expressiveness.

We further analyze a rich class of integral NMDAs that extends both letter-oriented and time-oriented NMDAs, in which the choice of discount factor depends on the word-prefix read so far (*tidy* NMDAs). We show that their expressiveness is as of deterministic integral NMDAs with an arbitrary choice of discount factors and that for every choice function $\theta : \Sigma^+ \to \mathbb{N} \setminus \{0, 1\}$, the class of θ-NMDAs enjoys all of the good properties of integral NDAs. (See Figure 1.)

Considering closure under algebraic operations, we further provide tight bounds on the size blow-up involved in the different operations (Table 1). To this end, we provide new lower bounds also for the setting of NDAs, by developing a general scheme to convert every NFA to a corresponding NDA of linearly the same size, and to convert some specific NDAs back to corresponding NFAs.

As for the decision problems of tidy NMDAs, we provide a PTIME algorithm for emptiness and PSPACE algorithms for the other problems of exact-value, universality, equivalence, and containment. The complexities are with respect to the automaton (or automata) size, which is considered as the maximum between the number of transitions and the maximal binary representation of any discount factor or weight in it. These new algorithms also improve the complexities of the previously known algorithms for solving the decision problems of NDAs, which were PSPACE with respect to unary representation of the weights.

As general choice functions need not be finitely represented, it might upfront limit the usage of tidy NMDAs. Yet, we show that finite transducers suffice, in the sense that they allow to represent every choice function θ that can serve for a θ-NMDA. We provide a PTIME algorithm to check whether a given NMDA is tidy, as well as if it is a $\mathcal{T}$-NMDA for a given transducer $\mathcal{T}$.

We show all of our results for both automata on finite words and automata on infinite words. Whenever possible, we provide a single proof for both settings. Due to lack of space, some of the full proofs appear in the appendix, while the rest can be found in [24].

2 Discounted-Sum Automata with Multiple Integral Discount Factors

We define a discounted-sum automaton with arbitrary discount factors, abbreviated NMDA, by adding to an NDA a discount factor in each of its transitions. An NMDA is defined on either finite or infinite words. The formal definition is given in Definition 1, and an example in Figure 2.

An *alphabet* Σ is an arbitrary finite set, and a *word* over Σ is a finite or infinite sequence of letters in Σ, with ε for the empty word. We denote the concatenation of a finite word u and a finite or infinite word w by $u \cdot w$, or simply by uw. We define Σ^+ to be the set of all finite words except the empty word, i.e., $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For a word $w = w(0)w(1)w(2)\ldots$, we denote the sequence of its letters starting at index i and ending at index j as $w[i..j] = w(i)w(i{+}1)\ldots w(j)$.

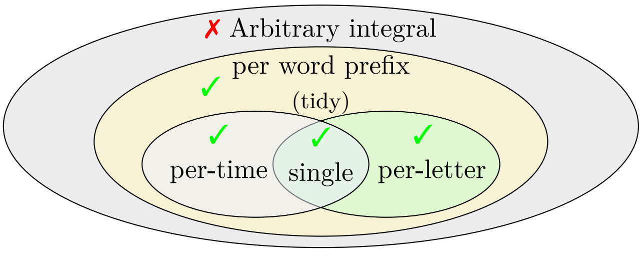


Figure 1 Classes of integral NMDAs, defined according to the flexibility of choosing the discount factors. The class of NMDAs with arbitrary integral factors is not closed under algebraic operations and under determinization. The other classes (for a specific choice function) are closed under both algebraic operations and determinization. Tidy NMDAs are as expressive as deterministic NMDAs with arbitrary integral discount factors.

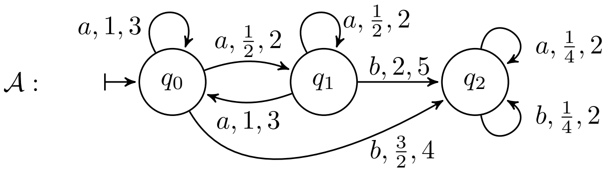


Figure 2 An NMDA $\mathcal{A}$. The labeling on the transitions indicate the alphabet letter, the weight of the transition, and its discount factor.

▶ **Definition 1.** *A nondeterministic discounted-sum automaton with multiple discount factors (NMDA), on finite or infinite words, is a tuple $\mathcal{A} = \langle \Sigma, Q, \iota, \delta, \gamma, \rho \rangle$ over an alphabet Σ, with a finite set of states Q, an initial set of states $\iota \subseteq Q$, a transition function $\delta \subseteq Q \times \Sigma \times Q$, a weight function $\gamma : \delta \to \mathbb{Q}$, and a discount-factor function $\rho : \delta \to \mathbb{Q} \cap (1, \infty)$, assigning to each transition its discount factor, which is a rational greater than one.* [2]

- *A* walk *in $\mathcal{A}$ from a state p_0 is a sequence of states and letters, $p_0, \sigma_0, p_1, \sigma_1, p_2, \cdots$, such that for every i, $(p_i, \sigma_i, p_{i+1}) \in \delta$. For example, $\psi = q_1, a, q_1, b, q_2$ is a walk of the NMDA $\mathcal{A}$ of Figure 2 on the word ab from the state q_1 .*
- *A* run *of $\mathcal{A}$ is a walk from an initial state.*
- *The length of a walk ψ, denoted by $|\psi|$, is n for a finite walk $\psi = p_0, \sigma_0, p_1, \cdots, \sigma_{n-1}, p_n$, and ∞ for an infinite walk.*
- *The i-th transition of a walk $\psi = p_0, \sigma_0, p_1, \sigma_1, \cdots$ is denoted by $\psi(i) = (p_i, \sigma_i, p_{i+1})$.*
- *The* value *of a finite or an infinite walk ψ is $\mathcal{A}(\psi) = \sum_{i=0}^{|\psi|-1} \left(\gamma\big(\psi(i)\big) \cdot \prod_{j=0}^{i-1} \frac{1}{\rho\big(\psi(j)\big)} \right)$. For example, the value of the walk $r_1 = q_0, a, q_0, a, q_1, b, q_2$ (which is also a run) of $\mathcal{A}$ from Figure 2 is $\mathcal{A}(r_1) = 1 + \frac{1}{2} \cdot \frac{1}{3} + 2 \cdot \frac{1}{2 \cdot 3} = \frac{3}{2}$.*
- *The* value *of $\mathcal{A}$ on a finite or infinite word w is $\mathcal{A}(w) = \inf\{\mathcal{A}(r) \mid r$ is a run of $\mathcal{A}$ on $w\}$.*
- *In the case where $|\iota| = 1$ and for every $q \in Q$ and $\sigma \in \Sigma$, we have $|\{q' \mid (q, \sigma, q') \in \delta\}| \leq 1$, we say that $\mathcal{A}$ is* deterministic, *denoted by DMDA, and view δ as a function to states.*
- *When all the discount factors are integers, we say that $\mathcal{A}$ is an* integral *NMDA.*

[2] Discount factors are sometimes defined in the literature as numbers between 0 and 1, under which setting weights are multiplied by these factors rather than divided by them.

In the case where for every $q \in Q$ and $\sigma \in \Sigma$, we have $|\{q' \mid (q,\sigma,q') \in \delta\}| \geq 1$, intuitively meaning that $\mathcal{A}$ cannot get stuck, we say that $\mathcal{A}$ is *complete*. It is natural to assume that discounted-sum automata are complete, and we adopt this assumption, as dead-end states, which are equivalent to states with infinite-weight transitions, break the property of the decaying importance of future events.

Automata $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent*, denoted by $\mathcal{A} \equiv \mathcal{A}'$, if for every word w, $\mathcal{A}(w) = \mathcal{A}'(w)$.

For every finite (infinite) walk $\psi = p_0, \sigma_0, p_1, \sigma_1, p_2, \cdots, \sigma_{n-1}, p_n$ ($\psi = p_0, \sigma_0, p_1, \cdots$), and all integers $0 \leq i \leq j \leq |\psi| - 1$ ($0 \leq i \leq j$), we define the finite sub-walk from i to j as $\psi[i..j] = p_i, \sigma_i, p_{i+1}, \cdots, \sigma_j, p_{j+1}$. For an infinite walk, we also define $\psi[i..\infty] = p_i, \sigma_i, p_{i+1}, \cdots$, namely the infinite suffix from position i. For a finite walk, we also define the target state as $\delta(\psi) = p_n$ and the accumulated discount factor as $\rho(\psi) = \prod_{i=0}^{n-1} \rho\big(\psi(i)\big)$.

We extend the transition function δ to finite words in the regular manner: For a word $u \in \Sigma^*$ and a letter $\sigma \in \Sigma$, $\delta(\varepsilon) = \iota; \delta(u \cdot \sigma) = \bigcup_{q \in \delta(u)} \delta(q, \sigma)$. For a state q of $\mathcal{A}$, we denote by $\mathcal{A}^q$ the automaton that is identical to $\mathcal{A}$, except for having q as its single initial state.

An NMDA may have rational weights, yet it is often convenient to consider an analogous NMDA with integral weights, achieved by multiplying all weights by their common denominator.

▶ **Proposition 2.** *For all constant $0 < m \in \mathbb{Q}$, NMDA $\mathcal{A} = \langle \Sigma, Q, \iota, \delta, \gamma, \rho \rangle$, NMDA $\mathcal{A}' = \langle \Sigma, Q, \iota, \delta, m \cdot \gamma, \rho \rangle$ obtained from $\mathcal{A}$ by multiplying all its weights by m, and a finite or infinite word w, we have $\mathcal{A}'(w) = m \cdot \mathcal{A}(w)$.*

Size. We define the size of $\mathcal{A}$, denoted by $|\mathcal{A}|$, as the maximum between the number of transitions and the maximal binary representation of any discount factor or weight in it. For rational weights, we assume all of them to have the same denominator. The motivation for a common denominator stems from the determinization algorithm (Theorem 8). Omitting this assumption will still result in a deterministic automaton whose size is only single exponential in the size of the original automaton, yet storing its states will require a much bigger space, changing our PSPACE algorithms (Section 4) into EXPSPACE ones.

Algebraic operations. Given automata $\mathcal{A}$ and $\mathcal{B}$ over the same alphabet and a non-negative scalar $c \in \mathbb{Q}$, we define an algebraic operation $op \in \{\min, \max, +, -\}$ on $\mathcal{A}$ and $\mathcal{B}$ as $\mathcal{C} \equiv op(\mathcal{A}, \mathcal{B})$ iff $\forall w. \mathcal{C}(w) = op\big(\mathcal{A}(w), \mathcal{B}(w)\big)$, and $op \in \{c\cdot, -\}$ as $\mathcal{C} \equiv op(\mathcal{A})$ iff $\forall w. \mathcal{C}(w) = op\big(\mathcal{A}(w)\big)$.

Decision problems. Given automata $\mathcal{A}$ and $\mathcal{B}$ and a threshold $\nu \in \mathbb{Q}$, we consider the following properties, with strict (or non-strict) inequalities: *Nonemptiness:* There exists a word w, s.t. $\mathcal{A}(w) < \nu$ (or $\mathcal{A}(w) \leq \nu$); *Exact-value:* There exists a word w, s.t. $\mathcal{A}(w) = \nu$; *Universality:* For all words w, $\mathcal{A}(w) < \nu$ (or $\mathcal{A}(w) \leq \nu$); *Equivalence:* For all words w, $\mathcal{A}(w) = \mathcal{B}(w)$; *Containment:* For all words w, $\mathcal{A}(w) > \mathcal{B}(w)$ (or $\mathcal{A}(w) \geq \mathcal{B}(w)$). [3]

Finite and infinite words. Results regarding NMDAs on finite words that refer to the existence of an equivalent automaton ("positive results") can be extended to NMDAs on infinite words due to Lemma 3 below. Likewise, results that refer to non-existence of an

[3] Considering quantitative containment as a generalization of language containment, and defining the "acceptance" of a word w as having a small enough value on it, we define that $\mathcal{A}$ is contained in $\mathcal{B}$ if for every word w, $\mathcal{A}$'s value on w is at least as big as $\mathcal{B}$'s value. (Observe the $>$ and $\geq$ signs in the definition.)

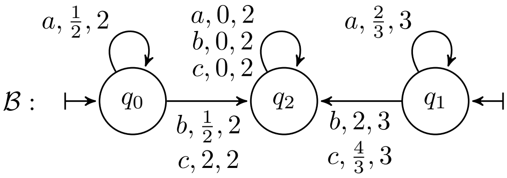


Figure 3 An integral NMDA $\mathcal{B}$ on infinite words that cannot be determinized.

equivalent automaton ("negative results") can be extended from NMDAs on infinite words to NMDAs on finite words. Accordingly, if not stated otherwise, we prove the positive results for automata on finite words and the negative results for automata on infinite words, getting the results for both settings.

▶ **Lemma 3.** *For all NMDAs $\mathcal{A}$ and $\mathcal{B}$, if for all finite word $u \in \Sigma^+$, we have $\mathcal{A}(u) = \mathcal{B}(u)$, then also for all infinite word $w \in \Sigma^\omega$, we have $\mathcal{A}(w) = \mathcal{B}(w)$.*

The proof is a simple extension of the proof of a similar lemma in [6] with respect to NDAs.

Notice that the converse does not hold, namely there are automata equivalent w.r.t. infinite words, but not w.r.t. finite words. (See an example in Figure 4.)

3 Arbitrary Integral NMDAs

Unfortunately, we show that the family of integral NMDAs in which discount factors can be chosen arbitrarily is not closed under determinization and under basic algebraic operations.

▶ **Theorem 4.** *There exists an integral NMDA that no integral DMDA is equivalent to.*

Proof sketch. Consider the integral NMDA $\mathcal{B}$ depicted in Figure 3. We show that for every $n \in \mathbb{N}$, $\mathcal{B}(a^n b^\omega) = 1 - \frac{1}{2^{n+1}}$ and $\mathcal{B}(a^n c^\omega) = 1 + \frac{1}{3^{n+1}}$.

An integral DMDA $\mathcal{D}$ that is equivalent to $\mathcal{B}$ will intuitively need to preserve an accumulated discount factor Π_n and an accumulated weight W_n on every a^n prefix, such that both suffixes of b^ω and c^ω will match the value of $\mathcal{B}$. Since the difference between the required value of each pair $\langle a^n b^\omega, a^n c^\omega \rangle$ is "relatively large", Π_n must have "many" small discount factors of 2 to compensate this difference. But too many discount factors of 2 will not allow to achieve the "delicate" values of $1 + \frac{1}{3^{n+1}}$. In the full proof, we formally analyze the mathematical properties of Π_n, showing that its prime-factor decomposition must indeed contain mostly 2's, "as well as" mostly 3's, leading to a contradiction. ◀

In the following proof that integral NMDAs are not closed under algebraic operations, we cannot assume toward contradiction a candidate deterministic automaton, and thus, as opposed to the proof of Theorem 4, we cannot assume a specific accumulative discount factor for each word prefix. Yet, we analyze the behavior of a candidate nondeterministic automaton on an infinite series of words, and build on the observation that there must be a state that appears in "the same position of the run" in infinitely many optimal runs of the automaton on these words.

▶ **Theorem 5.** *There exist integral NMDAs (even deterministic integral NDAs) $\mathcal{A}$ and $\mathcal{B}$ over the same alphabet, such that no integral NMDA is equivalent to* $\max(\mathcal{A}, \mathcal{B})$*, and no integral NMDA is equivalent to $\mathcal{A} + \mathcal{B}$.*

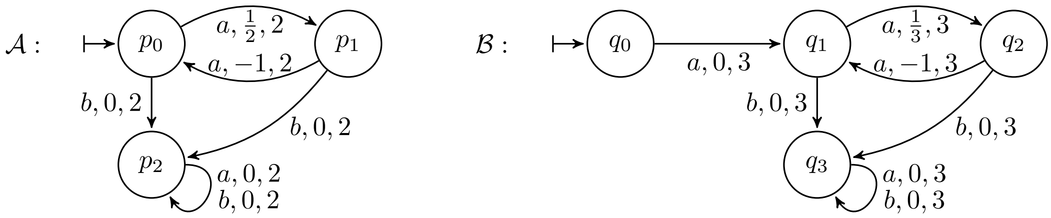


Figure 4 Deterministic integral NDAs that no integral NMDA is equivalent to their max or addition.

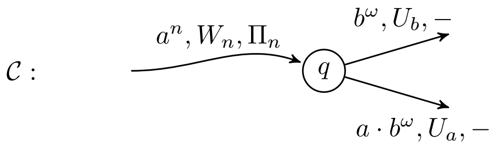


Figure 5 The state q and the notations from the proof of Theorem 5, for two different even $n \in \mathbb{N}$ such that $\delta(r_n[1..n]) = q$. The labels on the walks indicate the input word and the accumulated weight and discount factors.

Proof. Consider the NMDAs $\mathcal{A}$ and $\mathcal{B}$ depicted in Figure 4, and assume towards contradiction that there exists an integral NMDA $\mathcal{C}'$ such that for every $n \in \mathbb{N}$,

$$\mathcal{C}'(a^n b^\omega) = \max(\mathcal{A}, \mathcal{B})(a^n b^\omega) = \Big(\mathcal{A} + \mathcal{B}\Big)(a^n b^\omega) = \begin{cases} \frac{1}{2^n} & n \text{ is odd} \\ \frac{1}{3^n} & n \text{ is even} \end{cases}$$

Let $d \in \mathbb{N}$ be the least common denominator of the weights in $\mathcal{C}'$, and consider the NMDA $\mathcal{C} = \langle \Sigma, Q, \iota, \delta, \gamma, \rho \rangle$ created from $\mathcal{C}'$ by multiplying all its weights by d. Observe that all the weights in $\mathcal{C}$ are integers. According to Proposition 2, for every $n \in \mathbb{N}$, we have $\mathcal{C}(a^n b^\omega) = d \cdot \mathcal{C}'(a^n b^\omega) = \begin{cases} \frac{d}{2^n} & n \text{ is odd} \\ \frac{d}{3^n} & n \text{ is even} \end{cases}$

For every even $n \in \mathbb{N}$, let $w_n = a^n b^\omega$, and r_n a run of $\mathcal{C}$ on w_n that entails the minimal value of $\frac{d}{3^n}$. Since $\mathcal{C}$ is finite, there exists a state $q \in Q$ such that for infinitely many even $n \in \mathbb{N}$, the target state of r_n after n steps is q, i.e, $\delta(r_n[0..n-1]) = q$. We now show that the difference between $U_b = \mathcal{C}^q(b^\omega)$ and $U_a = \mathcal{C}^q(a \cdot b^\omega)$, the weights of the b^ω and $a \cdot b^\omega$ suffixes starting at q, discounted by $\Pi_n = \rho(r_n[0..n-1])$, which is the accumulated discount factor of the prefix of r_n up to q, is approximately $\frac{1}{2^n}$ (See Figure 5 for the notations). Since the weights of the prefixes are constant, for large enough n we will conclude that $m_1 \cdot 2^n \geq \Pi_n$ for some positive constant m_1.

For every such $n \in \mathbb{N}$, let $W_n = \mathcal{C}(r_n[0..n-1])$, and since $\mathcal{C}(r_n) = \frac{d}{3^n}$, we have

$$W_n + \frac{U_b}{\Pi_n} = \frac{d}{3^n} \tag{1}$$

Since the value of every run of $\mathcal{C}$ on $a^{n+1}b^\omega$ is at least $\frac{d}{2^{n+1}}$, we have $W_n + \frac{U_a}{\Pi_n} \geq \frac{d}{2^{n+1}}$. Hence, $\frac{d}{3^n} - \frac{U_b}{\Pi_n} + \frac{U_a}{\Pi_n} \geq \frac{d}{2^{n+1}}$ resulting in $\frac{U_a - U_b}{\Pi_n} \geq d \cdot \left(\frac{1}{2^{n+1}} - \frac{1}{3^n}\right)$. But for large enough n, we have $3^n > 2^{n+2}$, hence we get $\frac{1}{2^{n+1}} - \frac{1}{3^n} > \frac{1}{2^{n+1}} - \frac{1}{2^{n+2}} = \frac{1}{2^{n+2}}$, resulting in $\frac{U_a - U_b}{d} \cdot 2^{n+2} \geq \Pi_n$. And indeed, there exists a positive constant $m_1 = \frac{U_a - U_b}{d} \cdot 2^2$ such that $m_1 \cdot 2^n \geq \Pi_n$.

Now, U_b is a rational constant, otherwise Equation (1) cannot hold, as the other elements are rationals. Hence, there exist $x \in \mathbb{Z}$ and $y \in \mathbb{N}$ such that $U_b = \frac{x}{y}$, and $\frac{1}{3^n} = \frac{W_n \cdot \Pi_n + U_b}{d \cdot \Pi_n} =$

$\frac{W_n \cdot \Pi_n + \frac{x}{y}}{d \cdot \Pi_n} = \frac{W_n \cdot \Pi_n \cdot y + x}{d \cdot y \cdot \Pi_n}$. Since the denominator and the numerator of the right-hand side are integers, we conclude that there exists a positive constant $m_2 = d \cdot y$, such that $m_2 \cdot \Pi_n \geq 3^n$. Eventually, we get $m_1 \cdot m_2 \cdot 2^n \geq 3^n$, for some positive constants m_1 and m_2, and for infinitely many $n \in \mathbb{N}$. But this stands in contradiction with $\lim_{n\to\infty} \left(\frac{2}{3}\right)^n = 0$. ◀

4 Tidy NMDAs

We present the family of "tidy NMDAs" and show that it is as expressive as deterministic NMDAs with arbitrary integral discount factors. Intuitively, an integral NMDA is tidy if the choice of discount factors depends on the word prefix read so far. We further show that for every choice function θ, the class of all θ-NMDAs is closed under determinization and algebraic operations, and enjoys decidable algorithms for its decision problems.

The family of tidy NMDAs contains various other natural subfamilies, such as integral NMDAs in which the discount factors are chosen per letter (action) or per the elapsed time, on which we elaborate at the end of this section. Each of these subfamilies strictly extends the expressive power of integral NDAs.

▶ **Definition 6.** *An integral NMDA $\mathcal{A}$ over an alphabet Σ and with discount-factor function ρ is* tidy *if there exists a function $\theta : \Sigma^+ \to \mathbb{N} \setminus \{0, 1\}$, such that for every finite word $u = \sigma_1 \ldots \sigma_n \in \Sigma^+$, and every run $q_0, \sigma_1, \cdots, q_n$ of $\mathcal{A}$ on u, we have $\rho(q_{n-1}, \sigma_n, q_n) = \theta(u)$.*

In this case we say that $\mathcal{A}$ is a θ-NMDA.

▶ **Definition 7.** *For an alphabet Σ, a function $\theta : \Sigma^+ \to \mathbb{N} \setminus \{0, 1\}$ is a* choice function *if there exists an integral NMDA that is a θ-NMDA.*

For choice functions θ_1 and θ_2, the classes of θ_1-NMDAs and of θ_2-NMDAs are *equivalent* if they express the same functions, namely if for every θ_1-NMDA $\mathcal{A}$, there exists a θ_2-NMDA $\mathcal{B}$ equivalent to $\mathcal{A}$ and vice versa.

For every tidy NMDA $\mathcal{A}$ and finite word u, all the runs of $\mathcal{A}$ on u entail the same accumulated discount factor. We thus use the notation $\rho(u)$ to denote $\rho(r)$, where r is any run of $\mathcal{A}$ on u.

Observe that a general function $\theta : \Sigma^+ \to \mathbb{N}\setminus\{0, 1\}$ might require an infinite representation. Yet, we will show in Theorem 9 that every choice function has a finite representation.

Determinizability

We determinize a tidy NMDA by generalizing the determinization algorithm presented in [6] for NDAs. The basic idea in that algorithm is to extend the subset construction, by not only storing in each state of the deterministic automaton whether or not each state q of the original automaton $\mathcal{A}$ is reachable, but also the "gap" that q has from the currently optimal state q' of $\mathcal{A}$. This gap stands for the difference between the accumulated weights for reaching q and for reaching q', multiplied by the accumulated discounted factor. Since we consider tidy NMDAs, we can generalize this view of gaps to the setting of multiple discount factors, as it is guaranteed that the runs to q and to q' accumulated the same discount factor.

▶ **Theorem 8.** *For every choice function θ and a θ-NMDA $\mathcal{A}$, there exists a θ-DMDA $\mathcal{D} \equiv \mathcal{A}$ of size in $2^{O(|\mathcal{A}|)}$. Every state of $\mathcal{D}$ can be represented in space polynomial in $|\mathcal{A}|$.*

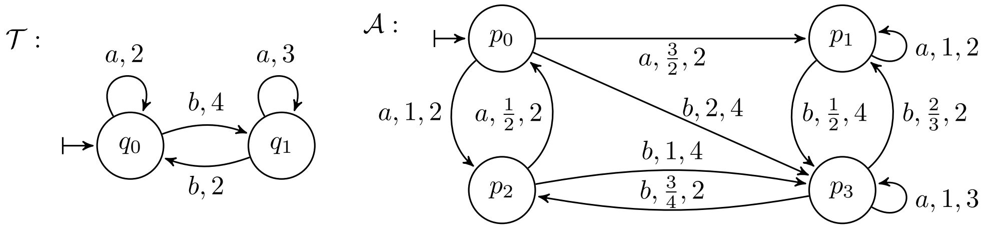


Figure 6 A transducer $\mathcal{T}$ and a $\mathcal{T}$-NMDA.

Representing Choice Functions

We show that, as opposed to the case of a general function $f : \Sigma^+ \to \mathbb{N} \setminus \{0, 1\}$, every choice function θ can be finitely represented by a transducer.

A transducer $\mathcal{T}$ (Mealy machine) is a 6-tuple $\langle P, \Sigma, \Gamma, p_0, \delta, \rho\rangle$, where P is a finite set of states, Σ and Γ are finite sets called the input and output alphabets, $p_0 \in P$ is the initial state, $\delta : P \times \Sigma \to P$ is the total transition function and $\rho : P \times \Sigma \to \Gamma$ is the total output function.

A transducer $\mathcal{T}$ represents a function, to which for simplicity we give the same name $\mathcal{T} : \Sigma^+ \to \Gamma$, such that for every word w, the value $\mathcal{T}(w)$ is the output label of the last transition taken when running $\mathcal{T}$ on w. The size of $\mathcal{T}$, denoted by $|\mathcal{T}|$, is the maximum between the number of transitions and the maximal binary representation of any output in the range of ρ.

Since in this work we only consider transducers in which the output alphabet Γ is the natural numbers $\mathbb{N}$, we omit Γ from their description, namely write $\langle P, \Sigma, p_0, \delta, \rho\rangle$ instead of $\langle P, \Sigma, \mathbb{N}, p_0, \delta, \rho\rangle$. An example of a transducer $\mathcal{T}$ and a $\mathcal{T}$-NMDA is given in Figure 6.

▶ **Theorem 9.** *For every function $\theta : \Sigma^+ \to \mathbb{N} \setminus \{0, 1\}$, θ is a choice function, namely there exists a θ-NMDA, if and only if there exists a transducer $\mathcal{T}$ such that $\theta \equiv \mathcal{T}$.*

Closure under Algebraic Operations

▶ **Theorem 10.** *For every choice function θ, the set of θ-NMDAs is closed under the operations of min, max, addition, subtraction, and multiplication by a rational constant.*

Proof. Consider a choice function θ and θ-NMDAs $\mathcal{A}$ and $\mathcal{B}$.

- *Multiplication by constant $c \geq 0$*: A θ-NMDA for $c \cdot \mathcal{A}$ is straightforward from Proposition 2.
- *Multiplication by -1*: A θ-NMDA for $-\mathcal{A}$ can be achieved by first determinizing $\mathcal{A}$, as per Theorem 8, into a θ-DMDA $\mathcal{D}$ and then multiplying all the weights in $\mathcal{D}$ by -1.
- *Addition*: Considering $\mathcal{A} = \langle \Sigma, Q_1, \iota_1, \delta_1, \gamma_1, \rho_1\rangle$ and $\mathcal{B} = \langle \Sigma, Q_2, \iota_2, \delta_2, \gamma_2, \rho_2\rangle$, a θ-NMDA for $\mathcal{A} + \mathcal{B}$ can be achieved by constructing the product automaton $\mathcal{C} = \langle \Sigma, Q_1 \times Q_2, \iota_1 \times \iota_2, \delta, \gamma, \rho\rangle$ such that $\delta = \big\{\big((q_1, q_2), \sigma, (p_1, p_2)\big) \bigm| (q_1, \sigma, p_1) \in \delta_1 \text{ and } (q_2, \sigma, p_2) \in \delta_2\big\}$, $\gamma\big((q_1, q_2), \sigma, (p_1, p_2)\big) = \gamma_1(q_1, \sigma, p_1) + \gamma_2(q_2, \sigma, p_2)$, $\rho\big((q_1, q_2), \sigma, (p_1, p_2)\big) = \rho_1(q_1, \sigma, p_1) = \rho_2(q_2, \sigma, p_2)$. The latter must hold since both ρ_1 and ρ_2 are compliant with θ.
- *Subtraction*: A θ-NMDA for $\mathcal{A} - \mathcal{B}$ can be achieved by i) Determinizing $\mathcal{B}$ to $\mathcal{B}'$; ii) Multiplying $\mathcal{B}'$ by -1, getting $\mathcal{B}''$; and iii) Constructing a θ-NMDA for $\mathcal{A} + \mathcal{B}''$.
- *min*: A θ-NMDA for $\min(\mathcal{A}, \mathcal{B})$ is straightforward by the nondeterminism on their union.
- *max*: A θ-NMDA for $\max(\mathcal{A}, \mathcal{B})$ can be achieved by i) Determinizing $\mathcal{A}$ and $\mathcal{B}$ to $\mathcal{A}'$ and $\mathcal{B}'$, respectively; ii) Multiplying $\mathcal{A}'$ and $\mathcal{B}'$ by -1, getting $\mathcal{A}''$ and $\mathcal{B}''$, respectively; iii) Constructing a θ-NMDA $\mathcal{C}''$ for $\min(\mathcal{A}'', \mathcal{B}'')$; iv) Determinizing $\mathcal{C}''$ into a θ-DMDA $\mathcal{D}$; and v) Multiplying $\mathcal{D}$ by -1, getting θ-NMDA $\mathcal{C}$, which provides $\max(\mathcal{A}, \mathcal{B})$. ◀

Table 1 The size blow-up involved in algebraic operations on tidy NMDAs.

$c \cdot \mathcal{A}$ (for $c \geq 0$)	$\min(\mathcal{A}, \mathcal{B})$	$\mathcal{A} + \mathcal{B}$	$-\mathcal{A}$	$\max(\mathcal{A}, \mathcal{B})$	$\mathcal{A} - \mathcal{B}$
Linear		Quadratic	Single Exponential		

We analyze next the size blow-up involved in algebraic operations. Most results in Table 1 are straightforward from the constructions presented in the proof of Theorem 10, however the size blow-up of the max operation is a little more involved. At a first glance, determinizing back and forth might look like requiring a double-exponential blow-up, however in this case an optimized procedure for the second determinization can achieve an overall single-exponential blow-up: Determinizing a tidy NMDA that is the union of two DMDAs, in which the transition weights are polynomial in the number of states, is shown to only involve a polynomial size blow-up.

▶ **Theorem 11.** *The size blow-up involved in the* max *operation on tidy NMDAs is at most single-exponential.*

We are not aware of prior lower bounds on the size blow-up involved in algebraic operations on NDAs. For achieving such lower bounds, we develop a general scheme to convert every NFA to a λ-NDA of linearly the same size that defines the same language with respect to a threshold value 0, and to convert some specific λ-NDAs back to corresponding NFAs.

The conversion of an NFA to a corresponding λ-NDA is quite simple. It roughly uses the same structure of the original NFA, and assigns four different transitions weights, depending on whether each of the source and target states is accepting or rejecting.

▶ **Lemma 12.** *For every $\lambda \in \mathbb{N} \setminus \{0, 1\}$ and NFA $\mathcal{A}$ with n states, there exists a λ-NDA $\tilde{\mathcal{A}}$ with $n + 2$ states, such that for every word $u \in \Sigma^+$, we have $u \in L(\mathcal{A})$ iff $\tilde{\mathcal{A}}(u) < 0$. That is, the language defined by $\mathcal{A}$ is equivalent to the language defined by $\tilde{\mathcal{A}}$ and the threshold* 0.

Converting an NDA to a corresponding NFA is much more challenging, since a general NDA might have arbitrary weights. We develop a conversion scheme, whose correctness proof is quite involved, from every NDA $\dot{\mathcal{B}}$ that is equivalent to $-\tilde{\mathcal{A}}$, where $\tilde{\mathcal{A}}$ is generated from an arbitrary NFA as per Lemma 12, to a corresponding NFA $\mathcal{B}$. Notice that the assumption that $\dot{\mathcal{B}} \equiv -\tilde{\mathcal{A}}$ gives us some information on $\dot{\mathcal{B}}$, yet $\dot{\mathcal{B}}$ might a priori still have arbitrary transition weights. Using this scheme, we provide an exponential lower bound on the size blow-up involved in multiplying an NDA by (-1). The theorem holds with respect to both finite and infinite words.

▶ **Theorem 13.** *For every $n \in \mathbb{N}$ and $\lambda \in \mathbb{N} \setminus \{0, 1\}$, there exists a λ-NDA $\mathcal{A}$ with n states over a fixed alphabet, such that every λ-NDA that is equivalent to $-\mathcal{A}$, w.r.t. finite or infinite words, has $\Omega(2^n)$ states.*

Proof sketch. NFA complementation is known to impose an exponential state blow-up [33, 26]. Hence, a conversion of an NFA $\mathcal{A}$ to a λ-NDA $\tilde{\mathcal{A}}$ as per Lemma 12, and a polynomial conversion of every λ-NDA $\dot{\mathcal{B}} \equiv -\tilde{\mathcal{A}}$ to a corresponding NFA $\mathcal{B}$, will show the required lower bound.

We provide such a back-conversion for NDAs whose values on the input words converge to some threshold as the words length grow to infinity, which is the case with every such $\dot{\mathcal{B}}$.

We first construct from $\dot{\mathcal{B}}$ a similar equivalent λ-NDA $\mathcal{B}'$ whose initial states have no incoming transitions. This eliminates the possibility that one run is a suffix of another, allowing to simplify some of our arguments. We then define $\hat{\delta}$ to be the transitions of $\mathcal{B}'$ that participate in some minimal run of $\mathcal{B}'$ on a word whose value is smaller than 0, and $\hat{\hat{\delta}} \subseteq \hat{\delta}$ to have those of them that are the last transition of such runs.

We define the NFA $\mathcal{B}$ to have the states of $\mathcal{B}'$, but only the transitions from $\hat{\delta}$. Its accepting states are clones of the target states of transitions in $\hat{\hat{\delta}}$, but without outgoing transitions. We then prove that $\mathcal{B}$ accepts a word u iff $\mathcal{B}'(u) = -\frac{1}{\lambda^{|u|}}$.

The first direction is easy: if $\mathcal{B}'(u) = -\frac{1}{\lambda^{|u|}}$, we get that all the transitions of a minimal run of $\mathcal{B}'$ on u are in $\hat{\delta}$, and its final transition is in $\hat{\hat{\delta}}$, hence there exists a run of $\mathcal{B}$ on u ending at an accepting state.

For the other direction, we assume towards contradiction that there exists a word u, such that $\mathcal{B}'(u) \neq -\frac{1}{\lambda^{|u|}}$, while there is an accepting run r_u of $\mathcal{B}$ on u. We define the "normalized value" of a run r' of $\mathcal{B}'$ as the value of $\mathcal{B}'$ multiplied by the accumulated discount factor, i.e., $\mathcal{B}'(r') \cdot \lambda^{|r'|}$. According to the special values assigned by $\mathcal{B}'$, whenever the normalized value reaches -1, we have an "accepting" run. We show that r_u and the structure of $\mathcal{B}$ imply the existence of two "accepting" runs $r'_1, r'_2 \in R^-$ that intersect in some state q, such that taking the prefix of r'_1 up to q results in a normalized value $\lambda^k W_1$ that is strictly smaller than the normalized value $\lambda^j W_2$ of the prefix of r'_2 up to q. Since r'_2 is an "accepting" run, the suffix of r'_2 reduces $\lambda^j W_2$ to -1 and therefore it will reduce $\lambda^k W_1$ to a value strictly smaller than -1, and the total value of the run to a value strictly smaller than $-\frac{1}{\lambda^n}$, which is not a possible value of $\mathcal{B}'$.

For showing the lower bound for NDAs that run on infinite words, we properly adjust the proof to consider words of the form $u \cdot \#^\omega$, for a fresh letter $\#$, rather than finite words. ◀

Basic Subfamilies

Tidy NMDAs constitute a rich family that also contains some basic subfamilies that are still more expressive than integral NDAs. Two such subfamilies are integral NMDAs in which the discount factors depend on the transition letter or on the elapsed time.

Notice that closure of tidy NMDAs under determinization and under algebraic operations is related to a specific choice function θ, namely every class of θ-NMDAs enjoys these closure properties (Theorems 8 and 10). Since the aforementioned subfamilies of tidy NMDAs also consist of θ-NMDA classes, their closure under determinization and under algebraic operations follows. For example, the class of NMDAs that assigns a discount factor of 2 to the letter "a" and of 3 to the letter "b" enjoys these closure properties.

Letter-Oriented Discount Factors

Allowing each action (letter) to carry its own discount factor is a basic extension of discounted summation, used in various models, such as Markov decision processes [29, 38].

A θ-NMDA over an alphabet Σ is *letter oriented* if all transitions over the same alphabet letter share the same discount factor; that is, if $\theta : \Sigma^+ \to \mathbb{N} \setminus \{0, 1\}$ coincides with a function $\Lambda : \Sigma \to \mathbb{N} \setminus \{0, 1\}$, in the sense that for every finite word u and letter σ, we have $\theta(u\sigma) = \Lambda(\sigma)$. (See an example in Figure 7.) Notice that every choice function θ for a letter-oriented θ-NMDA can be defined via a simple transducer of a single state, having a self loop over every letter with its assigned discount factor.

We show that letter-oriented NMDAs indeed add expressiveness over NDAs.

▶ **Theorem 14.** *There exists a letter-oriented NMDA that no integral NDA is equivalent to.*

Proof sketch. In the proof we consider the NMDA $\mathcal{A}$ depicted in Figure 7, and assume towards contradiction that there exists an integral λ-DDA $\mathcal{B}$ which is equivalent to $\mathcal{A}$. Since $\mathcal{B}$ is deterministic and has finitely many states, after reading only a letters some cycle will be eventually reached. We analyze the runs on words of the form $a^n b^\omega$ for values of n such

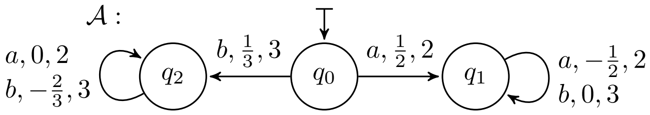


Figure 7 A letter-oriented discounted-sum automaton, for the discount factor function $\Lambda(a) = 2$; $\Lambda(b) = 3$, that no integral NDA is equivalent to.

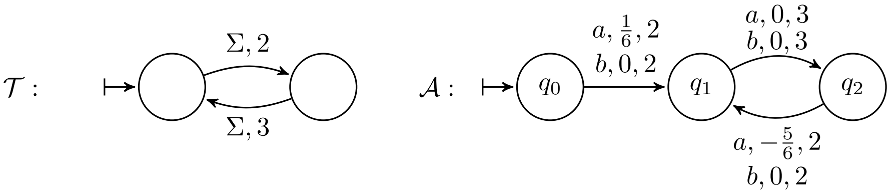


Figure 8 A time-oriented NMDA that no integral NDA is equivalent to, and a transducer that defines its choice function.

that the cycle in $\mathcal{B}$ is not taken at all, such that it is taken once, and such that it is taken twice. Since the accumulated discount factor added by the cycle is a constant equals to λ^i, where i is the length of the cycle, in order for $\mathcal{B}$ to have values of $\mathcal{B}(a^n b^\omega) = \frac{1}{2^n}$ on these words, we conclude that λ must equal 2. We now apply similar analysis regarding words of the form $b^n a^\omega$, for which $\mathcal{B}$ should have values of $\mathcal{B}(b^n a^\omega) = \frac{1}{3^n}$, and a cycle for the b letter, to conclude that λ must equal 3, and reach a contradiction. ◀

Time-Oriented Discount Factors

Allowing the discount factor to change over time is a natural extension of discounted summation, used in various disciplines, such as reinforcement learning [28, 22].

A θ-NMDA over an alphabet Σ is *time oriented* if the discount factor on a transition is determined by the distance of the transition from an initial state; that is, if $\theta : \Sigma^+ \to \mathbb{N}\setminus\{0, 1\}$ coincides with a function $\Lambda : \mathbb{N} \setminus \{0\} \to \mathbb{N} \setminus \{0, 1\}$, in the sense that for every finite word u, we have $\theta(u) = \Lambda\big(|u|\big)$.

For example, the NMDA $\mathcal{A}$ of Figure 8 is time-oriented, as all transitions taken at odd steps, in any run, have discount factor 2, and those taken at even steps have discount factor 3. The transducer $\mathcal{T}$ in Figure 8 represents its choice function.

Time-oriented NMDAs extend the expressiveness of NDAs, as proved for the time-oriented NMDA depicted in Figure 8.

▶ **Theorem 15.** *There exists a time-oriented NMDA that no integral NDA is equivalent to.*

5 Tidy NMDAs – Decision Problems

We show that all of the decision problems of tidy NMDAs are in the same complexity classes as the corresponding problems for discounted-sum automata with a single integral discount factor. That is, the nonemptiness problem is in PTIME, and the exact-value, universality, equivalence, and containment problems are in PSPACE (see Table 2). In the equivalence and containment problems, we consider θ-NMDAs with the same choice function θ. In addition, the problem of checking whether a given NMDA is tidy, as well as whether it is a

Table 2 The complexities of the decision problems of tidy NMDAs.

	Finite words	Infinite words
Non-emptiness ($<$)	PTIME (Theorem 19)	PTIME (Theorem 18)
Non-emptiness ($\leq$)	PTIME (Theorem 20)	
Containment ($>$)	PSPACE-complete (Theorem 24)	PSPACE (Theorem 26)
Containment ($\geq$)		PSPACE-complete (Theorem 25)
Equivalence	PSPACE-complete (Corollary 27)	
Universality ($<$)	PSPACE-complete (Theorem 28)	PSPACE (Theorem 28)
Universality ($\leq$)		PSPACE-complete (Theorem 28)
Exact-value	PSPACE-complete (Theorem 29)	PSPACE (Theorem 29)

θ-NMDA, for a given choice function θ, is decidable in PTIME. The complexities are w.r.t. the automata size (as defined in Section 2), and when considering a threshold ν, w.r.t. its binary representation.

Tidiness

Given an NMDA $\mathcal{A}$, one can check in PTIME whether $\mathcal{A}$ is tidy. The algorithm follows by solving a reachability problem in a Cartesian product of $\mathcal{A}$ with itself, to verify that for every word, the last discount factors are identical in all runs.

▶ **Theorem 16.** *Checking if a given NMDA $\mathcal{A}$ is tidy is decidable in time $O\big(|\mathcal{A}|^2\big)$.*

Given also a transducer $\mathcal{T}$, one can check in polynomial time whether $\mathcal{A}$ is a $\mathcal{T}$-NMDA.

▶ **Theorem 17.** *Checking if a given NMDA $\mathcal{A}$ is a $\mathcal{T}$-NMDA, for a given transducer $\mathcal{T}$, is decidable in time $O\big(|\mathcal{A}| \cdot |\mathcal{T}|\big)$.*

Proof sketch. We construct a nondeterministic weighted automaton that resembles the input NMDA and a deterministic weighted automaton that resembles the input transducer, replacing the original discount factors with weights. We then construct the product of the two automata, setting the transition weights to be the difference between the corresponding weights in the two automata, and check whether the weights on all reachable transitions are zero. ◀

Nonemptiness

We start with nonemptiness with respect to infinite words, for which there is a simple reduction to one-player discounted-payoff games. Notice that it applies to arbitrary NMDAs, not only to tidy ones.

▶ **Theorem 18.** *The nonemptiness problem of NMDAs w.r.t. infinite words is in PTIME for both strict and non-strict inequalities.*

For nonemptiness with respect to finite words, we cannot directly use the aforementioned game solution, as it relies on the convergence of the values in the limit. However, for nonemptiness with respect to strict inequality, we can reduce the finite-words case to the infinite-words case: If there exists an infinite word w such that $\mathcal{A}(w)$ is strictly smaller than the threshold, the distance between them cannot be compensated in the infinity, implying the existence of a finite prefix that also has a value smaller than the threshold; As for the other direction, we add to every state a 0-weight self loop, causing a small-valued finite word to also imply a small-valued infinite word.

▶ **Theorem 19.** *The nonemptiness problem of NMDAs w.r.t. finite words and strict inequality is in PTIME.*

For nonemptiness with respect to finite words and non-strict inequality, we cannot use the construction used in the proof of Theorem 19, since its final part is inadequate: It is possible to have an infinite word with value that equals the threshold, while every finite prefix of it has a value strictly bigger than the threshold. Yet, when considering *integral* NMDAs, we can use a different approach for resolving the problem, applying linear programming to calculate the minimal value of a finite run ending in every state.

▶ **Theorem 20.** *The nonemptiness problem of integral NMDAs w.r.t. finite words and non-strict inequality is in PTIME.*

Exact-Value, Universality, Equivalence, and Containment

We continue with the PSPACE-complete problems, to which we first provide hardness proofs, by reductions from the universality problem of NFAs, known to be PSPACE-complete [31]. Notice that the provided hardness results already stand for integral NDAs, not only to tidy NMDAs.

PSPACE-hardness of the containment problem for NDAs with respect to infinite words and non-strict inequalities is shown in [3]. We provide below more general hardness results, considering the equivalence problem, first with respect to finite words and then with respect to infinite words, as well as the exact-value, universality($\leq$) and universality($<$) problems with respect to finite words.

▶ **Lemma 21.** *The equivalence and universality($\leq$) problems of integral NDAs w.r.t. finite words are PSPACE-hard.*

Proof sketch. Given an NFA A, we construct in polynomial time an NDA $\mathcal{B}$ such that $\mathcal{A}$ is universal if and only if $\mathcal{B}$ gets a value of 0 on all finite words. $\mathcal{B}$ has the same structure as $\mathcal{A}$, and we assign weights on the transitions to guarantee that the value of $\mathcal{B}$ on every word u is at most $\frac{1}{2^{|u|}}$. In addition, we have in $\mathcal{B}$ a new "good" state q_{acc}, and for every original transition to an accepting state q of $\mathcal{A}$, we add in $\mathcal{B}$ a new "good" transition to q_{acc}, such that its weight allows $\mathcal{B}$ to have a value of 0 on a word that reaches q in $\mathcal{A}$. Finally, we add a "bad" transition out of q_{acc}, such that its weight ensures a total positive value, in the case that $\mathcal{B}$ continues the run out of q_{acc}. ◀

▶ **Lemma 22.** *The equivalence and universality($\leq$) problems of integral NDAs w.r.t. infinite words are PSPACE-hard.*

Proof sketch. The reduction from the universality problem of an NFA $\mathcal{A}$ is similar to the one provided in the proof of Lemma 21, with intuitively the following adaptations of the constructed NDA $\mathcal{B}$ to the case of infinite words: We add a new letter # to the alphabet, low-weighted #-transitions from the accepting states, and high-weighted #-transitions from the non-accepting states.

By this construction, the value of $\mathcal{B}$ on an infinite word $u \cdot \# \cdot w$, where u does not contain #, will be 0 if and only if $\mathcal{A}$ accepts u.

Notice that the value of $\mathcal{B}$ on an infinite word that does not contain # is also 0, as it is $\lim_{n\to\infty} \frac{1}{2^n}$. ◀

▶ **Lemma 23.** *The universality($<$) and exact-value problems of integral NDAs w.r.t. finite words are PSPACE-hard.*

We continue with the PSPACE upper bounds. The containment problem of NDAs was proved in [3] to be in PSPACE, using comparators to reduce the problem to language inclusion between Büchi automata. Our approach for the containment problem of NMDAs is different, and it also improves the complexity provided in [3] for NDAs (having a single discount factor), as we refer to binary representation of weights, while [3] assumes unary representation.[4]

Our algorithm for solving the containment problem between θ-NMDAs $\mathcal{A}$ and $\mathcal{B}$ is a nondeterministic polynomial space algorithm that determines the opposite, meaning whether there exists a word w such that $\mathcal{A}(w)-\mathcal{B}(w)<0$ for containment($\geq$) or $\mathcal{A}(w)-\mathcal{B}(w)\leq 0$ for containment($>$), to conclude that the problems are in co-NPSPACE and hence in PSPACE. We perform the determinization of $\mathcal{B}$ on-the-fly into a DMDA $\mathcal{D}$, and simulate on the fly a θ-NMDA for the difference between $\mathcal{A}$ and $\mathcal{D}$. We then non-deterministically guess a run r that witnesses a negative value of the difference automaton, while ensuring that the entire process only uses space polynomial in the size of the input automata. For meeting this space requirement, after each step of the run r, the algorithm maintains a *local data* consisting of the current state of $\mathcal{A}$, the current state of $\mathcal{D}$ and a "normalized difference" between the values of the runs of $\mathcal{A}$ and $\mathcal{D}$ on the word generated so far. When the normalized difference goes below 0, we have that the generated word w is a witness for $\mathcal{A}(w)<\mathcal{D}(w)$, when it gets to 0 we have a witness for $\mathcal{A}(w)=\mathcal{D}(w)$, and when it exceeds a certain *maximal recoverable difference*, which is polynomial in $|\mathcal{A}|+|\mathcal{B}|$, no suffix can be added to w for getting a witness.

▶ **Theorem 24.** *For every choice function θ, the containment problem of θ-NMDAs w.r.t. finite words is PSPACE-complete for both strict and non-strict inequalities.*

The algorithm for determining containment($\geq$) in the infinite-words settings is similar to the one presented for finite words, with the difference that rather than witnessing a finite word w, such that $\mathcal{A}(w)-\mathcal{B}(w)<0$, we witness a finite prefix u (of an infinite word w), such that the normalized difference between $\mathcal{A}(u)$ and $\mathcal{B}(u)$ (taking into account the accumulated discount factor on u) is bigger than some fixed threshold.

▶ **Theorem 25.** *For every choice function θ, the containment problem of θ-NMDAs w.r.t. infinite words and non-strict inequality is PSPACE-complete.*

To find a witness for strict non-containment in the infinite-words setting, we adapt the proof of Theorem 25 by adding an accept condition for detecting convergence of the difference between the two automata values to the threshold value, which is the existence of a cycle with the same normalized difference.

▶ **Theorem 26.** *For every choice function θ, the containment problem of θ-NMDAs w.r.t. infinite words and strict inequality is in PSPACE.*

A PSPACE algorithm for equivalence directly follows from the fact that $\mathcal{A}\equiv\mathcal{B}$ if and only if $\mathcal{A}\geq\mathcal{B}$ and $\mathcal{B}\geq\mathcal{A}$.

▶ **Corollary 27.** *The equivalence problem of tidy NMDAs is PSPACE-complete.*

We continue with the universality problems which are special cases of the containment problems.

[4] Rational weights are assumed to have a common denominator, both by us and by [3], where in the latter it is stated implicitly, by providing the complexity analysis with respect to transition weights that are natural numbers.

▶ **Theorem 28.** *The universality problems of tidy NMDAs are in PSPACE. The universality($<$) w.r.t. finite words, universality($\leq$) w.r.t. finite words, and universality($\leq$) w.r.t. infinite words are PSPACE-complete.*

▶ **Theorem 29.** *The exact-value problem of tidy NMDAs is in PSPACE (and PSPACE-complete w.r.t. finite words).*

6 Conclusions

The measure functions most commonly used in the field of quantitative verification, whether for describing system properties [9, 17, 30], automata valuation schemes [5, 6, 14, 3], game winning conditions [2, 18, 39], or temporal specifications [1, 4, 16, 35], are the limit-average (mean payoff) and the discounted-sum functions.

Limit-average automata cannot always be determinized [14] and checking their (non-strict) universality is undecidable [18]. Therefore, the tendency is to only use deterministic such automata, possibly with the addition of algebraic operations on them [10].

Discounted-sum automata with arbitrary rational discount factors also cannot always be determinized [14] and are not closed under algebraic operations [6]. Yet, with integral discount factors, they do enjoy all of these closure properties and their decision problems are decidable [6]. They thus provide a very interesting automata class for quantitative verification. Yet, they have a main drawback of only allowing a single discount factor.

We define a rich class of discounted-sum automata with multiple integral factors (tidy NMDAs) that strictly extends the expressiveness of automata with a single factor, while enjoying all of the good properties of the latter, including the same complexity of the required decision problems. We thus believe that tidy NMDAs can provide a natural and useful generalization of integral discounted-sum automata in all fields, and especially in quantitative verification of reinforcement learning applications, as novel approaches in this field extend the single discount factor that is used in the calculation of the expected return value to multiple ones [28, 22, 32].

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. Discounting in LTL. In *proceedings of TACAS*, volume 8413 of *LNCS*, pages 424–439, 2014. doi:10.1007/978-3-642-54862-8_37.

2 Daniel Andersson. An improved algorithm for discounted payoff games. In *proceedings of ESSLLI Student Session*, pages 91–98, 2006.

3 Suguman Bansal, Swarat Chaudhuri, and Moshe Y. Vardi. Comparator automata in quantitative verification. In *proceedings of FoSSaCS*, volume 10803 of *LNCS*, pages 420–437, 2018. doi:10.1007/978-3-319-89366-2_23.

4 Udi Boker, Krishnendu Chatterjee, Thomas A. Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. *ACM Trans. Comput. Log.*, 15(4):27:1–27:25, 2014. doi:10.1145/2629686.

5 Udi Boker and Thomas A. Henzinger. Approximate determinization of quantitative automata. In *proceedings of FSTTCS*, volume 18 of *LIPIcs*, pages 362–373, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.362.

6 Udi Boker and Thomas A. Henzinger. Exact and approximate determinization of discounted-sum automata. *Log. Methods Comput. Sci.*, 10(1), 2014. doi:10.2168/LMCS-10(1:10)2014.

7 Udi Boker, Thomas A. Henzinger, and Jan Otop. The target discounted-sum problem. In *proceedings of LICS*, pages 750–761, 2015. doi:10.1109/LICS.2015.74.

8 Udi Boker, Orna Kupferman, and Michal Skrzypczak. How deterministic are good-for-games automata? In *proceedings of FSTTCS*, volume 93 of *LIPIcs*, pages 18:1–18:14, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.18.

9 Krishnendu Chatterjee. Markov decision processes with multiple long-run average objectives. In *proceedings of FSTTCS*, volume 4855 of *LNCS*, pages 473–484. Springer, 2007. doi:10.1007/978-3-540-77050-3_39.

10 Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. In *proceedings of CONCUR*, volume 6269 of *LNCS*, pages 269–283, 2010. doi:10.1007/978-3-642-15375-4_19.

11 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating weighted automata. In *proceedings of FCT*, volume 5699 of *LNCS*, pages 3–13, 2009. doi:10.1007/978-3-642-03409-1_2.

12 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Probabilistic weighted automata. In *proceedings of CONCUR*, volume 5710 of *LNCS*, pages 244–258, 2009. doi:10.1007/978-3-642-04081-8_17.

13 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Expressiveness and closure properties for quantitative languages. *Log. Methods Comput. Sci.*, 6(3), 2010. URL: http://arxiv.org/abs/1007.4018.

14 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010. doi:10.1145/1805950.1805953.

15 Krishnendu Chatterjee, Vojtech Forejt, and Dominik Wojtczak. Multi-objective discounted reward verification in graphs and mdps. In *proceedings of LPAR*, volume 8312 of *LNCS*, pages 228–242, 2013. doi:10.1007/978-3-642-45221-5_17.

16 Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. Model checking discounted temporal properties. *Theor. Comput. Sci.*, 345(1):139–170, 2005. doi:10.1016/j.tcs.2005.07.033.

17 Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Discounting the future in systems theory. In *proceedings of ICALP*, volume 2719, pages 1022–1037, 2003. doi:10.1007/3-540-45061-0_79.

18 Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Torunczyk. Energy and mean-payoff games with imperfect information. In *proceedings of CSL*, volume 6247 of *LNCS*, pages 260–274, 2010. doi:10.1007/978-3-642-15205-4_22.

19 Manfred Droste and Dietrich Kuske. Skew and infinitary formal power series. *Theor. Comput. Sci.*, 366(3):199–227, 2006. doi:10.1016/j.tcs.2006.08.024.

20 Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Finite-valued weighted automata. In *proceedings of FSTTCS*, volume 29 of *LIPIcs*, pages 133–145, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.133.

21 Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Quantitative languages defined by functional automata. *Log. Methods Comput. Sci.*, 11(3), 2015. doi:10.2168/LMCS-11(3:14)2015.

22 Vincent François-Lavet, Raphaël Fonteneau, and Damien Ernst. How to discount deep reinforcement learning: Towards new dynamic strategies. *CoRR*, 2015. URL: http://arxiv.org/abs/1512.02011.

23 Hugo Gimbert and Wieslaw Zielonka. Limits of multi-discounted markov decision processes. In *proceedings of LICS*, pages 89–98, 2007. doi:10.1109/LICS.2007.28.

24 Guy Hefetz. Discounted-sum automata with multiple discount factors. Master's thesis, IDC, Herzliya, Israel, 2020. URL: https://www.idc.ac.il/en/schools/cs/research/documents/thesis-guy-hefetz.pdf.

25 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *proceedings of CSL*, volume 4207 of *LNCS*, pages 395–410, 2006. doi:10.1007/11874683_26.

26 Galina Jirásková. State complexity of some operations on binary regular languages. *Theor. Comput. Sci.*, 330(2):287–298, 2005. doi:10.1016/j.tcs.2004.04.011.

27 Yafim Kazak, Clark W. Barrett, Guy Katz, and Michael Schapira. Verifying deep-rl-driven systems. In *proceedings of NetAI@SIGCOMM*, pages 83–89, 2019. doi:10.1145/3341216.3342218.

28 Tor Lattimore and Marcus Hutter. Time consistent discounting. In *proceedings of ALT*, volume 6925 of *LNCS*, pages 383–397, 2011. doi:10.1007/978-3-642-24412-4_30.

29 Fernando Luque-Vásquez and J. Adolfo Minjárez-Sosa. *Iteration Algorithms in Markov Decision Processes with State-Action-Dependent Discount Factors and Unbounded Costs*, chapter 4, pages 55–69. Operations Research: the Art of Making Good Decisions. IntechOpen, 2017. doi:10.5772/65044.

30 Omid Madani, Mikkel Thorup, and Uri Zwick. Discounted deterministic markov decision processes and discounted all-pairs shortest paths. *ACM Trans. Algorithms*, 6(2):33:1–33:25, 2010. doi:10.1145/1721837.1721849.

31 Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *proceedings of 13th IEEE Symp. on Switching and Automata Theory*, pages 125–129, 1972. doi:10.1109/SWAT.1972.29.

32 Chris Reinke, Eiji Uchibe, and Kenji Doya. Average reward optimization with multiple discounting reinforcement learners. In *proceedings of ICONIP*, volume 10634 of *LNCS*, pages 789–800, 2017. doi:10.1007/978-3-319-70087-8_81.

33 William J. Sakoda and Michael Sipser. Nondeterminism and the size of two way finite automata. In *proceedings of STOC*, pages 275–286, 1978. doi:10.1145/800133.804357.

34 Richard S. Sutton and Andrew G.Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998. URL: http://dl.acm.org/doi/book/10.5555/551283.

35 Takashi Tomita, Shin Hiura, Shigeki Hagihara, and Naoki Yonezaki. A temporal logic with mean-payoff constraints. In *proceedings of ICFEM*, volume 7635 of *LNCS*, pages 249–265. Springer, 2012. doi:10.1007/978-3-642-34281-3_19.

36 Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *proceedings of FOCS*, pages 327–338, 1985. doi:10.1109/SFCS.1985.12.

37 Yufei Wang, Qiwei Ye, and Tie-Yan Liu. Beyond exponentially discounted sum: Automatic learning of return function. *CoRR*, 2019. URL: http://arxiv.org/abs/1905.11591.

38 Xiao Wu and Xianping Guo. Convergence of Markov decision processes with constraints and state-action dependent discount factors. *Sci. China Math.*, 63:167–182, 2020. doi:10.1007/s11425-017-9292-1.

39 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158:343–359, 1996. doi:10.1016/0304-3975(95)00188-3.

A Selected Proofs

This appendix presents some of the omitted proofs. All the full proofs can be found in [24].

Proof of Lemma 12. Given an NFA $\mathcal{A} = \langle \Sigma, Q, \iota, \delta, F \rangle$ and a discount factor $\lambda \in \mathbb{N} \setminus \{0,1\}$, we construct a λ-NDA $\tilde{\mathcal{A}} = \langle \Sigma, Q', \{p_0\}, \delta', \gamma' \rangle$ for which there exists a bijection f between the runs of $\mathcal{A}$ and the runs of $\tilde{\mathcal{A}}$ such that for every run r of $\mathcal{A}$ on a word u,

- r is an accepting run of $\mathcal{A}$ iff $f(r)$ is a run of $\tilde{\mathcal{A}}$ on u with the value $\tilde{\mathcal{A}}\big(f(r)\big) = -\frac{1}{\lambda^{|r|}}$.
- r is a non-accepting run of $\mathcal{A}$ iff $f(r)$ is a run of $\tilde{\mathcal{A}}$ on u with the value $\tilde{\mathcal{A}}\big(f(r)\big) = \frac{1}{\lambda^{|r|}}$.

We first transform $\mathcal{A}$ to an equivalent NFA $\mathcal{A}' = \langle \Sigma, Q', \{p_0\}, \delta', F \rangle$ that is complete and in which there are no transitions entering its initial state, and later assign weights to its transitions to create $\tilde{\mathcal{A}}$.

To construct $\mathcal{A}'$ we add two states to Q, having $Q' = Q \cup \{p_0, q_{hole}\}$, duplicate all the transitions from ι to start from p_0, and add a transition from every state to q_{hole}, namely $\delta' = \delta \cup \big\{(p_0, \sigma, q) \bigm| \exists p \in \iota, (p, \sigma, q) \in \delta\big\} \cup \big\{(q, \sigma, q_{hole}) \bigm| q \in Q', \sigma \in \Sigma\big\}$. Observe that $|Q'| = |Q| + 2$, and $L(\mathcal{A}) = L(\mathcal{A}')$. Next, we assign the following transition weights:

- For every $t = (p_0, \sigma, q) \in \delta'$, $\gamma'(t) = -\frac{1}{\lambda}$ if $q \in F$ and $\gamma'(t) = \frac{1}{\lambda}$ if $q \notin F$.
- For every $t = (p, \sigma, q) \in \delta'$ such that $p \neq p_0$, $\gamma'(t) = \frac{\lambda-1}{\lambda}$ if $p, q \in F$; $\gamma'(t) = \frac{\lambda+1}{\lambda}$ if $p \in F$ and $q \notin F$; $\gamma'(t) = -\frac{\lambda+1}{\lambda}$ if $p \notin F$ and $q \in F$; and $\gamma'(t) = -\frac{\lambda-1}{\lambda}$ if $p, q \notin F$.

By induction on the length of the runs on an input word u, one can show that for every $u \in \Sigma^+$, $\tilde{\mathcal{A}}(u) = -\frac{1}{\lambda^{|u|}}$ if $u \in L(\mathcal{A})$ and $\tilde{\mathcal{A}}(u) = \frac{1}{\lambda^{|u|}}$ if $u \notin L(\mathcal{A})$. ◀

Proof of Theorem 13.

Consider $n \in \mathbb{N}$ and $\lambda \in \mathbb{N} \setminus \{0, 1\}$. By [33, 26] there exists an NFA $\mathcal{A}$ with n states over a fixed alphabet of two letters, such that any NFA for the complement language $\overline{L(\mathcal{A})}$ has at least 2^n states.

Finite words.

Let $\tilde{\mathcal{A}}$ be a λ-NDA that is correlated to $\mathcal{A}$ as per Lemma 12, and assume towards contradiction that there exists a λ-NDA $\dot{\mathcal{B}} = \langle \Sigma, Q_{\dot{\mathcal{B}}}, \iota_{\dot{\mathcal{B}}}, \delta_{\dot{\mathcal{B}}}, \gamma_{\dot{\mathcal{B}}} \rangle$ with less than $\frac{2^n}{4}$ states such that $\dot{\mathcal{B}} \equiv -\tilde{\mathcal{A}}$.

We provide below a conversion opposite to Lemma 12, leading to an NFA for $\overline{L(\mathcal{A})}$ with less than 2^n states, and therefore to a contradiction. The conversion of $\dot{\mathcal{B}}$ back to an NFA builds on the specific values that $\dot{\mathcal{B}}$ is known to assign to words, as opposed to the construction of Lemma 12, which works uniformly for every NFA, and is much more challenging, since $\dot{\mathcal{B}}$ might have arbitrary transition weights. This conversion scheme can only work for λ-NDAs whose values on the input words converge to some threshold as the words length grow to infinity.

For simplification, we do not consider the empty word, since one can easily check if the input NFA accepts it, and set the complemented NFA to reject it accordingly.

By Lemma 12 we have that for every word $u \in \Sigma^+$, $\tilde{\mathcal{A}}(u) = -\frac{1}{\lambda^{|u|}}$ if $u \in L(\mathcal{A})$ and $\tilde{\mathcal{A}}(u) = \frac{1}{\lambda^{|u|}}$ if $u \notin L(\mathcal{A})$. Hence, $\dot{\mathcal{B}}(u) = -\frac{1}{\lambda^{|u|}}$ if $u \notin L(\mathcal{A})$ and $\dot{\mathcal{B}}(u) = \frac{1}{\lambda^{|u|}}$ if $u \in L(\mathcal{A})$. We will show that there exists an NFA $\mathcal{B}$, with less than 2^n states, such that $u \in L(\mathcal{B})$ iff $\dot{\mathcal{B}}(u) = -\frac{1}{\lambda^{|u|}}$, implying that $L(B) = \overline{L(\mathcal{A})}$.

We first construct a λ-NDA $\mathcal{B}' = \langle \Sigma, Q_{\mathcal{B}'}, \iota, \delta, \gamma \rangle$ that is equivalent to $\dot{\mathcal{B}}$, but has no transitions entering its initial states. This construction eliminates the possibility that one run is a suffix of another, allowing to simplify some of our arguments. Formally, $Q_{\mathcal{B}'} = Q_{\dot{\mathcal{B}}} \cup \iota$, $\iota = \iota_{\dot{\mathcal{B}}} \times \{1\}$, $\delta = \delta_{\dot{\mathcal{B}}} \cup \big\{ \big((p, 1), \sigma, q\big) \bigm| (p, \sigma, q) \in \delta_{\dot{\mathcal{B}}} \big\}$, and weights $\gamma(t) = \gamma_{\dot{\mathcal{B}}}(t)$ if $t \in \delta_{\dot{\mathcal{B}}}$ and $\gamma\big((p, 1), \sigma, q\big) = \gamma_{\dot{\mathcal{B}}}(p, \sigma, q)$ otherwise.

Let R^- be the set of all the runs of $\mathcal{B}'$ that entail a minimal value which is less than 0, i.e., $R^- = \{r \bigm| r \text{ is a minimal run of } \mathcal{B}' \text{ on some word and } \mathcal{B}'(r) < 0\}$. Let $\hat{\delta} \subseteq \delta$ be the set of all the transitions that take part in some run in R^-, meaning $\hat{\delta} = \{r(i) \bigm| r \in R^- \text{ and } 0 \leq i < |r|\}$, and $\hat{\hat{\delta}} \subseteq \delta$ the set of all transitions that are the last transition of those runs, meaning $\hat{\hat{\delta}} = \big\{ r\big(|r| - 1\big) \bigm| r \in R^- \big\}$.

We construct next the NFA $\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, \iota, \delta_{\mathcal{B}}, F_{\mathcal{B}} \rangle$. Intuitively, $\mathcal{B}$ has the states of $\mathcal{B}'$, but only the transitions from $\hat{\delta}$. Its accepting states are clones of the target states of the transitions in $\hat{\hat{\delta}}$, but without outgoing transitions. We will later show that the only runs of $\mathcal{B}$ that reach these clones are those that have an equivalent run in R^-. Formally, $Q_{\mathcal{B}} = Q'_{\mathcal{B}} \cup F_{\mathcal{B}}$, $F_{\mathcal{B}} = \big\{ (q, 1) \bigm| \exists p, q \in Q'_{\mathcal{B}} \text{ and } (p, \sigma, q) \in \hat{\hat{\delta}} \big\}$, and $\delta_{\mathcal{B}} = \hat{\delta} \cup \big\{ \big(p, \sigma, (q, 1)\big) \bigm| (p, \sigma, q) \in \hat{\hat{\delta}} \big\}$.

Observe that the number of states in $\mathcal{B}$ is at most 3 times the number of states in $\dot{\mathcal{B}}$, and thus less than 2^n. We will now prove that for every word u, $\mathcal{B}$ accepts u iff $\mathcal{B}'(u) = -\frac{1}{\lambda^{|u|}}$.

The first direction is easy: if $\mathcal{B}'(u) = -\frac{1}{\lambda^{|u|}}$, we get that all the transitions of a minimal run of $\mathcal{B}'$ on u are in $\hat{\delta}$, and its final transition is in $\hat{\hat{\delta}}$, hence there exists a run of $\mathcal{B}$ on u ending at an accepting state.

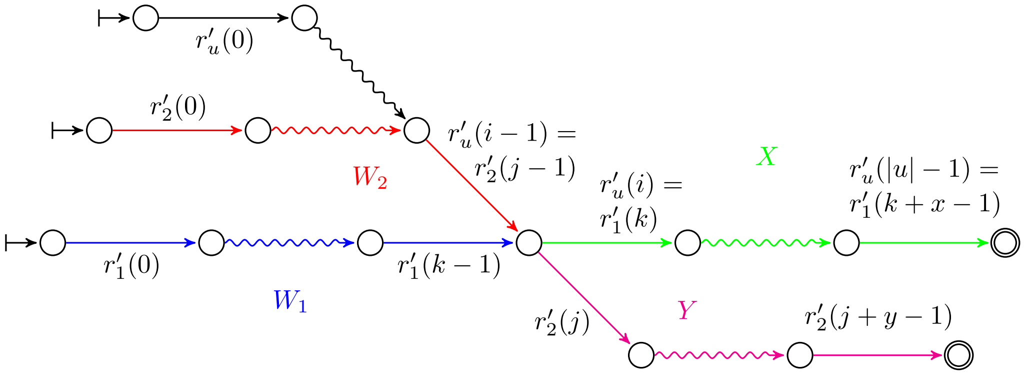


Figure 9 The runs and notations used in the proof of Theorem 13.

For the other direction, assume towards contradiction that there exists a word u, such that $\mathcal{B}'(u) = \frac{1}{\lambda^{|u|}}$, while there is an accepting run r_u of $\mathcal{B}$ on u.

Intuitively, we define the "normalized value" of a run r' of $\mathcal{B}'$ as the value of $\mathcal{B}'$ multiplied by the accumulated discount factor, i.e., $\mathcal{B}'(r') \cdot \lambda^{|r'|}$. Whenever the normalized value reaches -1, we have an "accepting" run. We will show that r_u and the structure of $\mathcal{B}$ imply the existence of two "accepting" runs $r'_1, r'_2 \in R^-$ that intersect in some state q, such that taking the prefix of r'_1 up to q results in a normalized value $\lambda^k W_1$ that is strictly smaller than the normalized value $\lambda^j W_2$ of the prefix of r'_2 up to q. Since r'_2 is an "accepting" run, the suffix of r'_2 reduces $\lambda^j W_2$ to -1 and therefore it will reduce $\lambda^k W_1$ to a value strictly smaller than -1, and the total value of the run to a value strictly smaller than $-\frac{1}{\lambda^n}$, which is not a possible value of $\mathcal{B}'$.

Formally, let $r_u(|u|-1) = \big(p', u(|u|-1), (q', 1)\big)$ be the final transition of r_u. We replace it with the transition $t' = \big(p', u(|u|-1), q'\big)$. The resulting run $r'_u = r_u[0..|u|-2] \cdot t$ is a run of $\mathcal{B}'$ on u, and therefore $\mathcal{B}'(r'_u) \geq \frac{1}{\lambda^{|u|}}$. Since $(q', 1)$ is an accepting state, we get by the construction of $\mathcal{B}$ that t' is in $\hat{\hat{\delta}}$. Consider a run $r'_1 \in R^-$ that shares the maximal suffix with r'_u, meaning that if there exist $r' \in R^-$ and $x > 0$ such that $r'[|r'|-x..|r'|-1] = r'_u[|u|-x..|u|-1]$ then also $r'_1[|r'_1|-x..|r'_1|-1] = r'_u[|u|-x..|u|-1]$.

Recall that all the initial states of $\mathcal{B}'$ have no transitions entering them and $\mathcal{B}'(r'_1) \neq \mathcal{B}'(r'_u)$, hence r'_1 is not a suffix of r'_u and r'_u is not a suffix of r'_1. Let i be the maximal index of r'_u such that $r'_u[i..|u|-1]$ is a suffix of r'_1, but $r'_u[i-1..|u|-1]$ is not a suffix of r'_1. Let k be the index in r'_1 such that $r'_1[k..|r'_1|-1] = r_u[i..|u|-1]$, and let $x = |r'_1| - k$ (see Figure 9).

Since $r'_u(i-1) \in \hat{\delta}$, there exists $r'_2 \in R^-$ and index j such that $r'_2(j-1) = r'_u(i-1)$. Let $y = |r'_2| - j$ (see Figure 9). Consider the run $r'_3 = r'_2[0..j-1] \cdot r'_u[i..|u|-1]$, starting with the prefix of r'_2 up to the shared transition with r'_u, and then continuing with the suffix of r'_u. Observe that $\mathcal{B}'(r'_3) > -\frac{1}{\lambda^{|r'_3|}}$ as otherwise $r'_3 \in R^-$ and has a larger suffix with r'_u than r'_1 has.

Let $W_1 = \mathcal{B}'\big(r'_1[0..k-1]\big)$, $W_2 = \mathcal{B}'\big(r'_2[0..j-1]\big)$, $X = \mathcal{B}'\big(r'_1[k..k+x-1]\big)$ (which is also $\mathcal{B}'\big(r'_u[i..|u|-1]\big)$), and $Y = \mathcal{B}'\big(r'_2[j..j+y-1]\big)$ (see Figure 9). The following must hold:

1. $W_1 + \frac{X}{\lambda^k} = \mathcal{B}'(r'_1) = -\frac{1}{\lambda^{k+x}}$. Hence, $\lambda^k W_1 = -\frac{1}{\lambda^x} - X$.
2. $W_2 + \frac{X}{\lambda^j} = \mathcal{B}'(r'_3) > -\frac{1}{\lambda^{j+x}}$. Hence, $\lambda^j W_2 > -\frac{1}{\lambda^x} - X$, and after combining with the previous equation, $\lambda^j W_2 > \lambda^k W_1$.
3. $W_2 + \frac{Y}{\lambda^j} = \mathcal{B}'(r'_2) = -\frac{1}{\lambda^{j+y}}$. Hence, $\lambda^j W_2 + Y = -\frac{1}{\lambda^y}$

Consider now the run $r'_4 = r'_1[0..k-1] \cdot r'_2[j..j+y-1]$, and combine Items 2 and 3 above to get that $\lambda^k W_1 + Y < -\frac{1}{\lambda^y}$. But this leads to $\mathcal{B}'(r'_4) = W_1 + \frac{Y}{\lambda^k} < -\frac{1}{\lambda^{k+y}} = -\frac{1}{\lambda^{|r'_4|}}$, and this means that there exists a word w of length $k+y$ such that $\mathcal{B}'(w) < -\frac{1}{\lambda^{k+y}}$, contradicting the assumption that $\mathcal{B}' \equiv \dot{\mathcal{B}} \equiv -\tilde{\mathcal{A}}$.

Infinite words.

For showing the lower bound for the state blow-up involved in multiplying an NDA by (-1) w.r.t. infinite words, we add a new letter $\#$ to the alphabet, and correlate every finite word u to an infinite word $u \cdot \#^\omega$. The proof is similar, applying the following modifications:

- The scheme presented in the proof of Lemma 12 now constructs a λ-NDA $\tilde{\mathcal{A}}$ over the alphabet $\Sigma \cup \{\#\}$, adding a 0-weighted transition from every state of $\tilde{\mathcal{A}}$ to q_{hole}. The function f that correlates between the runs of $\mathcal{A}$ and $\tilde{\mathcal{A}}$ is still a bijection, but with a different co-domain, correlating every run r of $\mathcal{A}$ on a finite word $u \in \Sigma^+$ to the run $f(r)$ of $\tilde{\mathcal{A}}$ on the word $u \cdot \#^\omega$.
- With this scheme, we get that $\dot{\mathcal{B}}(u \cdot \#^\omega) = -\frac{1}{\lambda^{|u|}}$ if $u \notin L(A)$ and $\dot{\mathcal{B}}(u \cdot \#^\omega) = \frac{1}{\lambda^{|u|}}$ if $u \in L(A)$, hence replacing all referencing to $\mathcal{B}'(u)$ with referencing to $\mathcal{B}'(u \cdot \#^\omega)$.
- R^- is defined with respect to words of the form $u \cdot \#^\omega$, namely $R^- = \{r \mid u \in \Sigma^+, r$ is a minimal run of $\mathcal{B}'$ on $u \cdot \#^\omega$ and $\mathcal{B}'(r) < 0\}$.
- R^-_p is a new set of all the maximal (finite) prefixes of the runs of R^- without any transitions for the $\#$ letter, meaning $R^-_p = \{r[0..i-1] \mid r \in R^-, r(i-1) = (p, \sigma, q)$ for some $\sigma \in \Sigma$, and $r(i) = (q, \#, s)\}$. $\hat{\delta}$ and $\hat{\hat{\delta}}$ are defined with respect to R^-_p instead of R^-.
- Defining r'_u, we consider a run $r'_t \in R^-$ that is a witness for $t' \in \hat{\hat{\delta}}$, meaning there exists $i \in \mathbb{N}$ for which $r'_t(i) = t'$, and $r'_t(i+1)$ is a transition for the $\#$ letter. Then $r'_u = r_u[0..|u|-2] \cdot t \cdot r'[i+1..\infty] = r_u[0..|u|-2] \cdot r'[i..\infty]$, is a run of $\mathcal{B}'$ on $u \cdot \#^\omega$.
- For choosing r'_1 that "shares the maximal suffix" with r'_u, we take $r'_1 \in R^-$ such that for every $r' \in R^-$ and $x > 0$, if $r'_u[i..\infty]$ is a suffix of r' then it is also a suffix of r'_1.
- For the different runs and their parts, we set $X = \mathcal{B}'\big(r'_1[k..\infty]\big)$, $Y = \mathcal{B}'\big(r'_2[j..\infty]\big)$, $r'_3 = r'_2[0..j-1] \cdot r'_u[i..\infty]$ and $r'_4 = r'_1[0..k-1] \cdot r'_2[j..\infty]$. ◀

Proof of Theorem 24. PSPACE hardness directly follows from Lemmas 21 and 23.

We provide a PSPACE upper bound. Consider a choice function θ, and θ-NMDAs $\mathcal{A} = \langle \Sigma, Q_{\mathcal{A}}, \iota, \delta_{\mathcal{A}}, \gamma_{\mathcal{A}}, \rho_{\mathcal{A}} \rangle$ and $\mathcal{B}$. We have that

$$\forall w. \mathcal{A}(w) > \mathcal{B}(w) \Leftrightarrow \nexists w. \mathcal{A}(w) \le \mathcal{B}(w) \Leftrightarrow \nexists w. \mathcal{A}(w) - \mathcal{B}(w) \le 0$$

and

$$\forall w. \mathcal{A}(w) \ge \mathcal{B}(w) \Leftrightarrow \nexists w. \mathcal{A}(w) < \mathcal{B}(w) \Leftrightarrow \nexists w. \mathcal{A}(w) - \mathcal{B}(w) < 0$$

We present a nondeterministic algorithm that determines the converse of containment, namely whether there exists a word w such that $\mathcal{A}(w) - \mathcal{B}(w) \le 0$ for continament($>$) or $\mathcal{A}(w) - \mathcal{B}(w) < 0$ for continament($\ge$), while using polynomial space w.r.t. $|\mathcal{A}|$ and $|\mathcal{B}|$, to conclude that the problems are in co-NPSPACE and hence in PSPACE.

Let $\mathcal{D} = \langle \Sigma, Q_{\mathcal{D}}, \{p_0\}, \delta_{\mathcal{D}}, \gamma_{\mathcal{D}}, \rho_{\mathcal{D}} \rangle$ be a θ-DMDA equivalent to $\mathcal{B}$, as per Theorem 8. Observe that the size of $\mathcal{D}$ can be exponential in the size of $\mathcal{B}$, but we do not save it all, but rather simulate it on the fly, and thus only save a single state of $\mathcal{D}$ at a time. We will later show that indeed the intermediate data we use in each iteration of the algorithm only requires a space polynomial in $|\mathcal{A}|$ and $|\mathcal{B}|$.

We consider separately non-strict containment ($\ge$) and strict containment ($>$).

Non-strict containment($\ge$).

For providing a word $w \in \Sigma^+$, such that $\mathcal{A}(w) - \mathcal{B}(w) < 0$, we nondeterministically generate on the fly a word w, a run r_w of $\mathcal{A}$ on w, and the single run of $\mathcal{D}$ on w, such that $\mathcal{A}(r_w) - \mathcal{B}(w) = \mathcal{A}(r_w) - \mathcal{D}(w) < 0$. Observe that $\mathcal{A}(w) \le \mathcal{A}(r_w)$, hence the above condition is equivalent to $\mathcal{A}(w) - \mathcal{B}(w) < 0$.

Let $M_{\mathcal{A}}$, $M_{\mathcal{B}}$, and $M_{\mathcal{D}}$ be the maximal absolute weights in $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{D}$, respectively.

We start by guessing an initial state q_{in} of $\mathcal{A}$ and setting a *local data* storage of $\langle q_{in}, p_0, 0\rangle$. The local data will maintain the current state of $\mathcal{A}$ and $\mathcal{D}$ respectively, and a "normalized difference" between the value of the run in $\mathcal{A}$ generated so far and the value of $\mathcal{D}$ on the word generated so far, as formalized below. The algorithm iteratively guesses, given a local data $\langle q, p, d\rangle$, a letter $\sigma \in \Sigma$ and a transition $t = (q, \sigma, q') \in \delta_{\mathcal{A}}(q, \sigma)$, and calculates the *normalized difference* $d' = \rho_{\mathcal{A}}(t)\big(d + \gamma_{\mathcal{A}}(t) - \gamma_{\mathcal{D}}(p, \sigma)\big)$ between the values $\mathcal{A}(r_w)$ and $\mathcal{B}(w)$, w.r.r. the word w and the run r_w generated so far. If d' is bigger than the *maximal recoverable difference* $2S$, where $S = M_{\mathcal{A}} + 3M_{\mathcal{B}}$, we abort, if $d' < 0$, we have that the generated word w indeed witnesses that $\mathcal{A}(w) < \mathcal{D}(w)$ (the *accept condition* holds), and otherwise we continue and update the local data to $\langle q', \delta(p, \sigma), d'\rangle$. Observe that by the construction in the proof of Theorem 8, for every weight W in $\mathcal{D}$ we have that $|W| \leq 2T + M_{\mathcal{B}} \leq 3M_{\mathcal{B}}$, where T is the maximal difference between the weights in $\mathcal{B}$. Hence $S > M_{\mathcal{A}} + M_{\mathcal{D}}$ is polynomial w.r.t. $|\mathcal{A}|$ and $|\mathcal{B}|$, and can be calculated in polynomial space w.r.t. $|\mathcal{A}|$ and $|\mathcal{B}|$.

We show by induction on the length of the word w that whenever a word w and a run r_w are generated, the value d in the corresponding local data $\langle q, p, d\rangle$ indeed stands for the normalized difference between $\mathcal{A}(r_w)$ and $\mathcal{D}(w)$, namely

$$d = \rho_{\mathcal{A}}(r_w)\big(\mathcal{A}(r_w) - \mathcal{D}(w)\big) \tag{2}$$

For the base case we have a single-letter word $w = \sigma$, and a single-transition run $r_w = t$. Hence, $d' = \rho_{\mathcal{A}}(t)\big(d + \gamma_{\mathcal{A}}(t) - \gamma_{\mathcal{D}}(p, \sigma)\big) = \rho_{\mathcal{A}}(r_w)\big(0 + \mathcal{A}(r_w) - \mathcal{D}(w)\big) = \rho_{\mathcal{A}}(r_w)\big(\mathcal{A}(r_w) - \mathcal{D}(w)\big)$.

For the induction step, consider an iteration whose initial local data is $\langle q, p, d\rangle$, for a generated word w and run r_w, that guessed the next letter σ and transition t, and calculated the next local data $\langle q', p', d'\rangle$. Then we have $d' = \rho_{\mathcal{A}}(t)\big(d + \gamma_{\mathcal{A}}(t) - \gamma_{\mathcal{D}}(p, \sigma)\big)$. By the induction assumption, we get:

$$\begin{aligned} d' &= \rho_{\mathcal{A}}(t)\Big(\rho_{\mathcal{A}}(r_w)\big(\mathcal{A}(r_w) - \mathcal{D}(w)\big) + \gamma_{\mathcal{A}}(t) - \gamma_{\mathcal{D}}(p, \sigma)\Big) \\ &= \rho_{\mathcal{A}}(r_w)\rho_{\mathcal{A}}(t)\Big(\mathcal{A}(r_w) + \frac{\gamma_{\mathcal{A}}(t)}{\rho_{\mathcal{A}}(r_w)} - \mathcal{D}(w) - \frac{\gamma_{\mathcal{D}}(p, \sigma)}{\rho_{\mathcal{A}}(r_w)}\Big) \\ &= \rho_{\mathcal{A}}(r_w \cdot t)\Big(\mathcal{A}(r_w \cdot t) - \Big(\mathcal{D}(w) + \frac{\gamma_{\mathcal{D}}(p, \sigma)}{\rho_{\mathcal{A}}(r_w)}\Big)\Big), \end{aligned}$$

and since the discount-factor functions of $\mathcal{A}$ and $\mathcal{D}$ both agree with θ, we have

$$d' = \rho_{\mathcal{A}}(r_w \cdot t)\Big(\mathcal{A}(r_w \cdot t) - \Big(\mathcal{D}(w) + \frac{\gamma_{\mathcal{D}}(p, \sigma)}{\rho_{\mathcal{D}}(w)}\Big)\Big) = \rho_{\mathcal{A}}(r_w \cdot t)\big(\mathcal{A}(r_w \cdot t) - \mathcal{D}(w \cdot \sigma)\big),$$

which provides the required result of the induction claim.

Next, we show that the accept condition holds iff there exist a finite word w and run r_w of $\mathcal{A}$ on w such that $\mathcal{A}(r_w) - \mathcal{D}(w) < 0$. Since for every finite word w we have $\rho_{\mathcal{A}}(w) > 0$, we conclude from Equation (2) that if $d' < 0$ was reached for a generated word w and a run r_w, we have that $\mathcal{A}(r_w) - \mathcal{D}(w) < 0$. For the other direction, assume toward contradiction that there exist finite word w and run r_w of $\mathcal{A}$ on w such that $\mathcal{A}(r_w) - \mathcal{D}(w) < 0$, but the algorithm aborts after generating some prefixes $w[0..i]$ and $r_w[0..i]$. Meaning that $\rho_{\mathcal{A}}(r_w[0..i])\big(\mathcal{A}(r_w[0..i]) - \mathcal{D}(w[0..i])\big) > 2M_{\mathcal{A}} + 2M_{\mathcal{D}}$. Let $W_1 = \mathcal{A}(r_w[i+1..|r_w|-1])$ and $W_2 = \mathcal{D}^{\delta_{\mathcal{D}}(w[0..i])}(w[i+1..|r_w|-1])$. Observe that

$$\begin{aligned} 0 &> \mathcal{A}(r_w) - \mathcal{D}(w) > \rho_{\mathcal{A}}\big(r_w[0..i]\big)\big(\mathcal{A}(r_w) - \mathcal{D}(w)\big) \\ &= \rho_{\mathcal{A}}\big(r_w[0..i]\big)\mathcal{A}\big(r_w[0..i]\big) + W_1 - \big(\rho_{\mathcal{A}}(r_w[0..i])\mathcal{D}(w[0..i]) + W2\big) \\ &> 2M_{\mathcal{A}} + 2M_{\mathcal{D}} + W_1 - W_2 \end{aligned}$$

But since all the discount factors applied by θ are greater or equal to 2, we have that $|W_1| \leq 2M_{\mathcal{A}}$ and $|W_2| \leq 2M_{\mathcal{B}}$, leading to a contradiction.

To see that the algorithm indeed only uses space polynomial in $|\mathcal{A}|$ and $|\mathcal{B}|$, observe that the first element of the data storage is a state of $\mathcal{A}$, only requiring a space logarithmic in $|\mathcal{A}|$, the second element is a state of $\mathcal{D}$, requiring by Theorem 8 a space polynomial in $\mathcal{B}$, and the third element is a non-negative rational number bounded by $2S$, whose denominator is the multiplication of the denominators of the weights in $\mathcal{A}$ and $\mathcal{D}$, and as shown in the proof of Theorem 8, also of the multiplication of the denominators of the weights in $\mathcal{A}$ and $\mathcal{B}$, thus requires a space polynomial in $|\mathcal{A}|$ and $|\mathcal{B}|$. Finally, in order to compute this third element, we calculated a weight of a transition in $\mathcal{D}$, which only requires, by the proof of Theorem 8, a space polynomial in $|\mathcal{B}|$.

Strict Containment(>).

The algorithm is identical to the one used for the containment($\geq$) problem with changing the accept condition $d' < 0$ to $d' \leq 0$. This condition is met iff there exists a finite word w such that $\mathcal{A}(w) - \mathcal{B}(w) \leq 0$. The proof is identical while modifying "< 0" to "≤ 0" in all of the equations. ◀

Reachability in Distributed Memory Automata

Benedikt Bollig
CNRS, LSV, ENS Paris-Saclay, Université Paris-Saclay, Gif-sur-Yvette, France
bollig@lsv.fr

Fedor Ryabinin
IMDEA Software Institue, Madrid, Spain
fedor.ryabinin@imdea.org

Arnaud Sangnier
IRIF, Université de Paris, CNRS, France
sangnier@irif.fr

Abstract

We introduce Distributed Memory Automata, a model of register automata suitable to capture some features of distributed algorithms designed for shared-memory systems. In this model, each participant owns a local register and a shared register and has the ability to change its local value, to write it in the global memory and to test atomically the number of occurrences of its value in the shared memory, up to some threshold. We show that the control-state reachability problem for Distributed Memory Automata is PSPACE-complete for a fixed number of participants and is in PSPACE when the number of participants is not fixed a priori.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Distributed algorithms, Atomic snapshot objects, Register automata, Reachability

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.13

Related Version A full version of the paper is available at `https://hal.archives-ouvertes.fr/hal-02983089`.

Funding Partly supported by ANR FREDDA (ANR-17-CE40-0013).

1 Introduction

Distributed algorithms are nowadays building blocks of modern systems in almost all computer-aided areas. One can find them in ad-hoc networks, telecommunication protocols, cache-coherence protocols, swarm robotics, or biological models. Such systems often consist of small components that solve subtasks such as mutual exclusion, leader election, or spanning trees [9, 12].

One way to classify distributed algorithms is according to how processes communicate with each other. Among the most popular classes are message-passing algorithms or shared-memory systems. In the latter case, processes write to a global memory that can be read by other processes. An important instance of a global memory are atomic snapshot objects, where every process has a dedicated global memory cell it can write to and, as the name suggests, can "snapshot" the current state of *all* global memory cells. Snapshot objects are exploited in renaming algorithms whose aim is to assign to every process a unique id from a small[1] namespace [6]. In a snapshot algorithm, every process may choose a value that is currently not in the global memory, and write it in its local memory. These two steps are non-atomic so that, in principle, other processes may simultaneously choose the same

[1] but unbounded, as it may depend on the number of processes

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 13; pp. 13:1–13:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

value. A process may then examine the snapshot (for example, check whether it contains its local value) and decide how to proceed (for example, overwrite its global memory cell by the contents of its local memory cell).

In view of their widespread use, distributed algorithms are often subject to strong correctness requirements. However, they are inherently difficult to verify. One reason is that they are usually designed for an unbounded number of participants manipulating data from an unbounded domain. That is, we have to deal with two sources of infinity during their analysis. In this paper, we take a further step towards the modeling and verification of algorithms involving atomic snapshot objects.

The Model. We introduce *distributed memory automata (DMAs)*, which feature some of the above-mentioned communication primitives of snapshot objects. Our model is based on *register automata*, which have been used as a general formal model of systems that involve (unbounded or infinite) data. Register automata go back to the work of Kaminski and Francez [10] and have recently sparked new interest leading to extensions with various applications [2, 4, 7, 13]. In a network of a DMA, every process is equipped with two registers, one representing its local memory cell, and one representing its global memory cell that every other process can read. Just like register automata, we allow registers to carry data values, i.e., values from an infinite domain (such as process identifiers), albeit comparison is only possible wrt. equality. Both, write and read operations, are restricted though. A process can perform three types of actions, which are all inspired by snapshot algorithms. It may (i) write a new value, currently not present in any global register, into its local register, (ii) copy the value from its local into its global register, and (iii) test how often its local value already occurs in the overall global memory. Note that (i) and (iii) indeed correspond to a scan operation followed by a test in atomic-snapshot algorithms. Variants of register automata have already been used to model distributed algorithms, but in a round-based setting with peer-to-peer communication [1, 5], whereas DMAs can be classified as asynchronous shared-memory systems.

Parameterized Verification. The vast majority of register-automata models impose a bound on the number of registers. In the execution of a DMA, on the other hand, the number of registers is not fixed in advance: it is *parameterized*. Indeed, distributed algorithms are often characterized by the fact that they run on systems with any number of, a priori identical, processes. Since, in many applications, the number of components varies or is unknown, these algorithms must be working on an architecture of any size. Such systems are called *parameterized*, where the parameter is the number of processes or components. Just like register automata, parameterized verification has had a long history and continues to be an active research area. We refer to [3, 8] for overviews.

In this paper, we consider a simple reachability question for DMAs, which amounts to safety verification (is a "bad" control state reachable?). In general, there are (at least) two ways to analyze parameterized systems. In the "fixed-process case", we know in advance how many processes are involved. This problem often reduces to solving reachability questions in standard models. The parameterized reachability problem, on the other hand, asks whether a given control state is reachable in some execution, involving an arbitrary number of processes. In general, this requires different techniques. Some systems, however, enjoy *cut-off* and *monotonicity* properties. In that case, the number of processes that allow for reaching a given state can be found by solving finitely many fixed-process instances [3].

Results for Distributed Memory Automata. In the fixed-process case, a standard argument allows us to restrict the problem to a bounded number of data values and to show membership in PSPACE. We also provide a matching lower bound. The PSPACE-complete intersection emptiness problem for a collection of finite-state automata is an evident starting point [11]. However, the reduction turns out to be subtle due to the fact that all processes in a DMA look the same. In particular, we have to use guards in a nested fashion to "separate" these processes so that each of them can simulate a different finite automaton.

In the case of parameterized reachability, we show that control-state reachability is in PSPACE, too, leaving tightness of this upper bound as an open problem. The proof proceeds in two steps. We first show PSPACE membership of a "subproblem", which we name *train reachability*. As a model of shared ressources with a parameterized number of processes, it is of independent interest. This algorithm is then called repeatedly within a saturation procedure that allows us to gradually compute the set of all reachable control states.

Outline. The paper is organized as follows. In Section 2, we define our model of DMAs. In Section 3, we consider the case of a fixed number of processes, for which control-state reachability is PSPACE-complete. We then move on to the case of a parameterized number of processes. The proof spans over two sections: In Section 4, we introduce and solve parameterized train reachability. This is exploited, in Section 5, to show decidability, and PSPACE membership, for parameterized reachability in DMAs. Missing proofs can be found in the long version of the paper, available at `https://hal.archives-ouvertes.fr/hal-02983089`.

2 Reachability in Distributed Memory Automata

We start with a few preliminary definitions. For $n \in \mathbb{N}$, we let $[0,n] := \{0,\ldots,n\}$ and $[1,n] := \{1,\ldots,n\}$. For a set A, a natural number $n \geq 1$, a tuple $\mathbf{a} \in A^n$, and $i \in [1,n]$, we let $\mathbf{a}[i]$ refer to the i-th component of $\mathbf{a}$. For $d \in A$, we let $|\mathbf{a}|_d = |\{i \in [1,n] \mid \mathbf{a}[i] = d\}|$ denote the number of occurrences of d in $\mathbf{a}$. Accordingly, we write $d \in \mathbf{a}$ if $|\mathbf{a}|_d \geq 1$, and $d \notin \mathbf{a}$ if $|\mathbf{a}|_d = 0$.

Suppose we have a system with $n \geq 1$ processes. Processes are referred to by their index $p \in [1,n]$. In the global memory, every process has a dedicated memory cell, holding a natural number (which may be a process identifier, a sequence number, etc.). Thus, the state of the global memory is a tuple $\mathbf{M} \in \mathbb{N}^n$. Similarly, every process has a local memory cell. The contents of all local memory cells is also described by a tuple $\boldsymbol{\ell} \in \mathbb{N}^n$. A process p can take a snapshot of the global memory $\mathbf{M}$ and examine its contents. More precisely, p can

- test how often its local value $\boldsymbol{\ell}[p]$ occcurs in $\mathbf{M}$, up to some threshold,
- modify its local memory cell by assigning it *some* new value that is currently not present in *the whole of* $\mathbf{M}$, or
- modify its global memory cell by assigning it its local value (and thus overwriting the old value of $\mathbf{M}[p]$).

Accordingly, $\mathcal{T} = \{=_t, <_t, >_t \mid t \in \mathbb{N}\}$ is the set of *tests* and $\Sigma = \{\mathsf{new}, \mathsf{write}\} \cup \mathcal{T}$ the set of *actions*. For $k \in \mathbb{N}$ and $\bowtie_t \in \mathcal{T}$ with $\bowtie \in \{=,<,>\}$, we write $k \models \bowtie_t$ if $k \bowtie t$. We are now prepared to define distributed memory automata.

▶ **Definition 1.** *A* distributed memory automaton (DMA) *is a tuple* $\mathcal{A} = (S, \iota, \Delta, F)$ *where* S *is the finite set of* states, $\iota \in S$ *is the* initial state, $\Delta \subseteq S \times \Sigma \times S$ *is the finite set of* transitions, *and* F *is the set of* final states.

For a test $\bowtie_t \in \mathcal{T}$, we let $|\bowtie_t| = \max\{1, t\}$. Moreover, $|\mathsf{new}| = |\mathsf{write}| = 1$. The size of $\mathcal{A}$ is defined as $|\mathcal{A}| := |S| + \sum_{(s,\sigma,s')\in\Delta} |\sigma|$. Note that we assume a unary encoding of tests.

For $n \geq 1$, an n-configuration (shortly a configuration) is a tuple $\gamma = (\mathbf{s}, \boldsymbol{\ell}, \mathbf{M}) \in S^n \times \mathbb{N}^n \times \mathbb{N}^n$. Given a process $p \in [1, n]$, we consider that $\mathbf{s}[p]$ is the current state of p, $\boldsymbol{\ell}[p]$ is the content of its local memory, and $\mathbf{M}[p]$ is the entry of p in the global memory. We use *states*(γ) to denote the set $\{\mathbf{s}[p] \mid p \in [1, n]\}$ and $|\gamma|$ to represent the number of processes n of the configuration γ.

We say that γ is *initial* if, for all $p \in [1, |\gamma|]$, we have $\mathbf{s}[p] = \iota$ and $\boldsymbol{\ell}[p] \notin \mathbf{M}$, and for all $p, q \in [1, |\gamma|]$, $\boldsymbol{\ell}[p] = \boldsymbol{\ell}[q]$ implies $p = q$. Hence, in an initial configuration, each process has a different value in its local register and none of these values appears in the shared memory. Moreover, configuration γ is called *final* if $\mathbf{s}[p] \in F$ for some $p \in [1, |\gamma|]$, i.e., if one of its processes is in a state of F.

Let $\mathbb{C}_{\mathcal{A},n}$ be the set of n-configurations and $\mathbb{C}_{\mathcal{A}} := \bigcup_{n\geq 1} \mathbb{C}_{\mathcal{A},n}$ be the set of all configurations. We define a global transition relation $\Longrightarrow_{\mathcal{A}} \subseteq \mathbb{C}_{\mathcal{A}} \times (\Sigma \times \mathbb{N}) \times \mathbb{C}_{\mathcal{A}}$. Suppose $\gamma = (\mathbf{s}, \boldsymbol{\ell}, \mathbf{M})$ and $\gamma' = (\mathbf{s}', \boldsymbol{\ell}', \mathbf{M}')$ are two configurations and let $\sigma \in \Sigma$ and $p \in [1, |\gamma|]$. We let $\gamma \xRightarrow{(\sigma,p)}_{\mathcal{A}} \gamma'$ if the following hold:

- $|\gamma| = |\gamma'|$ and
- $(\mathbf{s}[p], \sigma, \mathbf{s}'[p]) \in \Delta$,
- $\mathbf{s}[q] = \mathbf{s}'[q]$ and $\boldsymbol{\ell}[q] = \boldsymbol{\ell}'[q]$ and $\mathbf{M}[q] = \mathbf{M}'[q]$ for all $q \in [1, |\gamma|] \setminus \{p\}$,
- if $\sigma = \mathsf{new}$, then $\boldsymbol{\ell}'[p] \notin \mathbf{M}$ and $\mathbf{M} = \mathbf{M}'$,
- if $\sigma = \mathsf{write}$, then $\boldsymbol{\ell}[p] = \boldsymbol{\ell}'[p] = \mathbf{M}'[p]$,
- if $\sigma \in \mathcal{T}$, then $\boldsymbol{\ell} = \boldsymbol{\ell}'$ and $\mathbf{M} = \mathbf{M}'$ and $|\mathbf{M}|_{\boldsymbol{\ell}[p]} \models \sigma$.

We write $\Longrightarrow_{\mathcal{A}}$ for the union of all relations $\xRightarrow{(\sigma,p)}_{\mathcal{A}}$ and denote by $\Longrightarrow^*_{\mathcal{A}}$ the reflexive and transitive closure of $\Longrightarrow_{\mathcal{A}}$. Note that if $\gamma \Longrightarrow_{\mathcal{A}} \gamma'$ then there exists $n \geq 1$ such that $\gamma, \gamma' \in \mathbb{C}_{\mathcal{A},n}$. In fact, the transition relation $\Longrightarrow_{\mathcal{A}}$ does not change the number of involved processes. If we have $(\mathbf{s}, \boldsymbol{\ell}, \mathbf{M}) \xRightarrow{(\mathsf{new},p)}_{\mathcal{A}} (\mathbf{s}', \boldsymbol{\ell}', \mathbf{M}')$ with $\boldsymbol{\ell}'[p] = d$, we will sometimes write $(\mathbf{s}, \boldsymbol{\ell}, \mathbf{M}) \xRightarrow{(\mathsf{new}(d),p)}_{\mathcal{A}} (\mathbf{s}', \boldsymbol{\ell}', \mathbf{M}')$ to provide explicitly the new local value. A *run* ρ of $\mathcal{A}$ is a finite sequence of the form $\gamma_0 \xRightarrow{(\sigma_0,p_0)}_{\mathcal{A}} \gamma_1 \xRightarrow{(\sigma_1,p_1)}_{\mathcal{A}} \gamma_2 \cdots \xRightarrow{(\sigma_{k-1},p_{k-1})}_{\mathcal{A}} \gamma_k$ where $\gamma_i \in \mathbb{C}_{\mathcal{A}}$ for all $i \in [0, k]$ and γ_0 is initial. It is said to be final if γ_k is final.

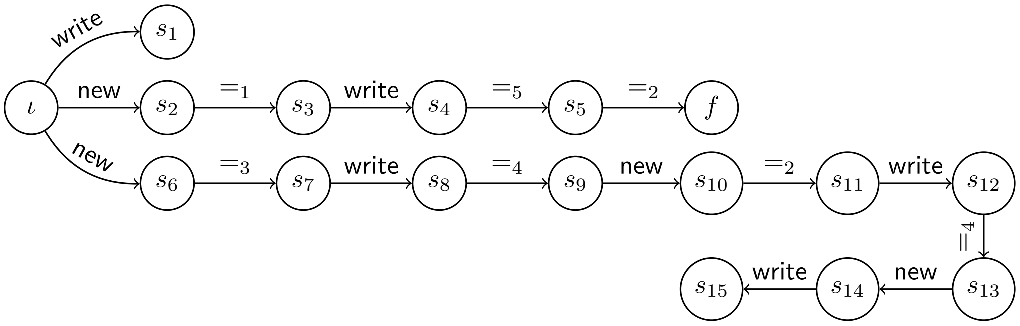


Figure 1 An example DMA.

▶ **Example 2.** In the example presented in Figure 1, the final state f is reachable and we shall see in the development of the paper how we can prove this, since it is not obvious at first sight. We present here an execution to reach s_9 with four processes. Assume that the initial configuration is $\big([\iota, \iota, \iota, \iota], [0, 1, 2, 3], [4, 4, 4, 4]\big)$. From this configuration, if one process performs a write going to s_1, then the system will not be able to reach s_9, because no other processes will be able to choose the same value (with a new) since the value is written in the global memory and the consecutive test $=_4$ (necessary to reach s_9) will never be available. Instead, to reach s_9, we perform the following step: $\big([\iota,\iota,\iota,\iota],[0,1,2,3],[4,4,4,4]\big) \xRightarrow{(\mathsf{new},2)}_{\mathcal{A}} \big([\iota,s_2,\iota,\iota],[0,0,2,3],[4,4,4,4]\big)$. Here the second process can choose the same local value as the first one since it is not yet written in the memory. Thanks to the sequence $\xRightarrow{(\mathsf{new},3)}_{\mathcal{A}}\xRightarrow{(\mathsf{new},4)}_{\mathcal{A}}\xRightarrow{(\mathsf{write},1)}_{\mathcal{A}}$, we reach the configuration $\big([s_1,s_2,s_2,s_6],[0,0,0,0],[0,4,4,4]\big)\big)$, from which we can perform the transition sequence $\xRightarrow{(=_1,2)}_{\mathcal{A}}\xRightarrow{(=_1,3)}_{\mathcal{A}}\xRightarrow{(\mathsf{write},2)}_{\mathcal{A}}\xRightarrow{(\mathsf{write},3)}_{\mathcal{A}}$ to reach the configuration $\big([s_1,s_4,s_4,s_6],[0,0,0,0],[0,0,0,4]\big)$ from which it is possible to perform $\xRightarrow{(=_3,4)}_{\mathcal{A}}\xRightarrow{(\mathsf{write},4)}_{\mathcal{A}}\xRightarrow{(=_4,4)}_{\mathcal{A}}$ making the fourth process reach s_9. Note that we could build a similar execution with 5 processes to reach the configuration $\big([s_1,s_4,s_4,s_4,s_9],[0,0,0,0,0],[0,0,0,0,0]\big)$ by adding an extra process that behaves as process two but writes its value after the last process reaches s_9. We have then five times the value 0 in the global memory. But from this configuration, it is not possible to reach f since, to pass the sequence of transitions $(s_4,=_5,s_5),(s_5,=_2,f)$, at least three processes have to delete the value 0 from their global memory and this is not possible.

The main problem we study in the paper is the reachability problem, in which we check whether a state of a given DMA can be reached without specifying the number of processes. In other words, the number of processes is a parameter that needs to be instantiated.

Reachability

I: DMA $\mathcal{A}$

Q: $\gamma \Longrightarrow^*_{\mathcal{A}} \gamma'$ for some initial $\gamma \in \mathbb{C}_{\mathcal{A}}$ and some final $\gamma' \in \mathbb{C}_{\mathcal{A}}$?

In order to understand the above problem, it is important to also know how to solve the respective problem where the number of processes is imposed.

Fixed-Reachability

I: DMA $\mathcal{A}$ and $n \geq 1$ (encoded in unary)

Q: $\gamma \Longrightarrow^*_{\mathcal{A}} \gamma'$ for some initial $\gamma \in \mathbb{C}_{\mathcal{A},n}$ and some final $\gamma' \in \mathbb{C}_{\mathcal{A},n}$?

Hence, Reachability consists in checking the existence of a final run and Fixed-Reachability seeks for a final run with an initial n-configuration.

3 Considering a fixed number of processes

In this section, we show that Fixed-Reachability is Pspace-complete.

First we explain how we obtain the upper bound. We consider a DMA $\mathcal{A} = (S,\iota,\Delta,F)$ and a fixed number of processes $n \geq 1$. Note that, for any configuration $\gamma = (\mathbf{s},\boldsymbol{\ell},\mathbf{M}) \in S^n \times \mathbb{N}^n \times \mathbb{N}^n$, the number of different values in the local memory $\boldsymbol{\ell}$ and in the global memory $\mathbf{M}$ is at most $2n$. Hence, if there is a run $\gamma_0 \xRightarrow{(\sigma_0,p_0)}_{\mathcal{A}} \gamma_1 \xRightarrow{(\sigma_1,p_1)}_{\mathcal{A}} \gamma_2 \cdots \xRightarrow{(\sigma_{k-1},p_{k-1})}_{\mathcal{A}} \gamma_k$ such that $\gamma_i \in \mathbb{C}_{\mathcal{A},n}$ for all $i \in [0,k]$ and γ_0 is initial and γ_k is final, then there is a run $\gamma'_0 \xRightarrow{(\sigma_0,p_0)}_{\mathcal{A}} \gamma'_1 \xRightarrow{(\sigma_1,p_1)}_{\mathcal{A}} \gamma'_2 \cdots \xRightarrow{(\sigma_{k-1},p_{k-1})}_{\mathcal{A}} \gamma'_k$ such that $\gamma'_i \in S^n \times [0,2n]^n \times [0,2n]^n$ for all $i \in [0,k]$ and γ'_0 is initial and γ'_k is final. In fact, the set of values $[0,2n]^n$ is enough to define an initial configuration in $\mathbb{C}_{\mathcal{A},n}$ since we can pick $2n$ different values. Since there are $2n+1$ different values in $[0,2n]$, when performing an action new, it is always possible to pick a value in $[0, 2n]$ that appears neither in the local memory nor in the global memory. To solve Fixed-Reachability for n processes, we then check whether a final configuration is reachable from an initial one in the graph where the set of vertices is $S^n \times [0, 2n]^n \times [0, 2n]^n$ and the edges are defined by the transition relation $\Longrightarrow_{\mathcal{A}}$. This graph having an exponential number of vertices, the search can be performed in NPspace, i.e., in Pspace thanks to Savitch's theorem. Note that we could obtain the same upper bound by reducing our problem to the non-emptiness problem for non-deterministic register automata with $2n$ registers and S^n as a set of states and then use the fact that the non-emptiness problem for such automata is in Pspace [7]. The $2n$ registers will correspond to the local and global memory and the different actions of the DMA can be simulated by a register automaton.

▶ **Proposition 3.** *Fixed-Reachability is in Pspace.*

To show the lower bound, we do a reduction from the intersection emptiness problem of many non-deterministic finite state automata. A non-deterministic finite state automaton (FSA) A over a finite alphabet Λ is a tuple (Q, q_ι, δ, F) where Q is a finite set of states, $q_\iota \in Q$ is an initial state, $\delta \subseteq Q \times \Lambda \times Q$ is the transition relation and $F \subseteq Q$ is the set of accepting states. A finite word $w = w_0 w_1 \dots w_{k-1}$ in Λ^* is accepted by A if there exists a sequence of states $(q_i)_{0 \leq i \leq k}$ such that $q_0 = q_\iota$, $q_k \in F$, and $(q_i, w_i, q_{i+1}) \in \delta$ for all $i \in [0, k-1]$. We denote by $\mathcal{L}(A)$ the language of A, i.e., the set of words $\{w \in \Lambda^* \mid w \text{ is accepted by } A\}$. The emptiness intersection problem asks, given m FSA $A_1, \dots, A_m$ over the alphabet Λ, whether $\bigcap_{1 \leq i \leq m} \mathcal{L}(A_i) = \emptyset$. This problem is known to be Pspace-complete [11].

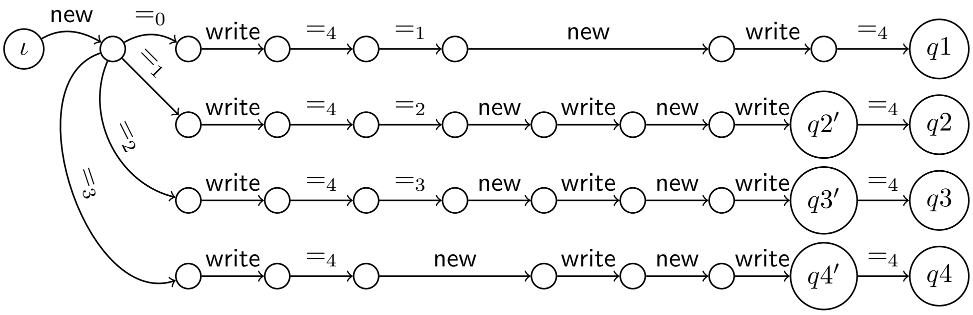
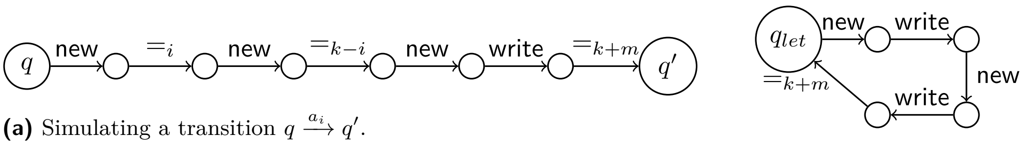


Figure 2 Gadget to isolate 4 processes.

In order to reduce the intersection emptiness problem for FSA to Fixed-Reachability, we first need a gadget to bring different processes to different parts of the DMA so that each of these processes can simulate a particular finite automaton. This gadget is necessary since in DMA all processes begin in the same initial state. An example of this gadget for four processes is depicted in Figure 2. At the beginning, all the processes are in the initial state ι and we claim that if a process reaches the state $q1$ then there is one process in $q2$ or in $q2'$ (because at this stage we cannot force the transition labeled with the test $=_4$ leading to $q2$ to be taken), one process in $q3$ or in $q3'$ and one process in $q4$ or in $q4'$. In fact, if one process is in $q1$, then it has to first write its local value and the only way to do this is to take the upper branch of the DMA and, after writing, wait for the other processes to write their value in order to pass the test $=_4$. Because of this test, all the processes have to choose the same value with the first new. One way to pass the test $=_4$ for the first process is that all the processes take the upper branch as follows: they all choose the same new value, then they all pass the test $=_0$ then they all write their value and they all pass the test $=_4$. However this execution will then stop because of the following test $=_1$ which could not be taken because, at this stage, none of the processes can rewrite its value in the global memory. The same reasoning can be iterated to show that the only way to pass the test $=_1$ in the upper branch is to have one process per branch, the first one writes its value, then the second one can pass the first test $=_1$ in the second branch and writes the same value, the third one passes the test $=_2$ in the third branch and writes its value and the last one can pass the test $=_3$ in the last branch and writes its value. Each process can then pass, in its branch, the test $=_4$ but only the fourth process can perform a new followed by a write to overwrite its value in the global memory (the other ones have to wait because of the tests $=_3, =_2, =_1$). Hence the fourth process overwrites its value, then the third one, then the second one and finally the first process can pass the test $=_1$. After that all the processes can again perform a new and write to choose the same new value and write it to the memory to allow the first process to reach $q1$.

We consider now an instance of the intersection emptiness problem with m FSA $A_i = (Q_i, q_\iota^{(i)}, \delta_i, F_i)$ for $i \in [1, m]$ working over the finite alphabet $\Lambda = \{a_1, a_2, \ldots, a_k\}$. Without loss of generality, we can assume that, for each $i \in [1, m]$, the set $F_i = \{q_f^{(i)}\}$ is a singleton and furthermore the only way to reach this state is to read the letter a_k that is not present in any other transitions. Hence all the words accepted by A_i end with a_k and if an automaton reads a word until its last letter a_k, then the automaton accepts this word.



(a) Simulating a transition $q \xrightarrow{a_i} q'$.

(b) Simulating the k letters.

Figure 3 Encoding intersection emptiness of finite automata into DMA.

To check whether $\bigcap_{1 \le i \le m} \mathcal{L}(A_i) = \emptyset$, we build a DMA and consider $m + k$ processes. The first m processes simulate the automata $(A_i)_{1 \le i \le m}$ and the k last processes simulate the read letters. First we use the gadget presented previously to separate these $m + k$ processes in different parts of the DMA. For $i \in [1, m]$, the i-th process will be brought to the initial state $q_\iota^{(i)}$ of each NFA whereas the last k processes are brought to the state q_{let} leading to the part of the DMA depicted in Figure 3b.

We show then on Figure 3a how we simulate each transition $q \xrightarrow{a_i} q'$ of the finite state automata in the DMA. A process $p \in [1, m]$, in order to simulate the transition $q \xrightarrow{a_i} q'$, first takes a new value and waits until this value appears i times in the global memory. At this stage only the k last processes are able to write, so i of these last processes take the same new value and write it to the global memory. There possibly remain at most $k - i$ processes that did not take the same new value. But the process p then takes a new value and it has to appear $k - i$ times in the global memory, so the $k - i$ processes that did not write their value to the memory can do it now. Finally, after this, each process can take a new value and write it to the global memory and if they all have taken the same new value, they can all pass the test $=_{k+m}$. This ensures that all the processes simulating the automata have read the same letter and, moreover, that the different processes are synchronized. For instance, imagine that a process simulating the automaton takes the transitions $\xrightarrow{=_1}\xrightarrow{\mathsf{new}}\xrightarrow{=_{k-1}}$ and another one at the same stage of the simulation goes through $\xrightarrow{=_2}\xrightarrow{\mathsf{new}}\xrightarrow{=_{k-2}}$. This is possible: a process $p1$ simulating a letter writes its value to the memory allowing the test $=_1$, then a second process simulating a letter writes the same value to the memory allowing the test $=_2$, then the $k - 2$ remaining last processes take the same new value and so does the process $p1$

(by taking the third transition labelled by new in the loop starting in q_{let}), then the $k-2$ last processes write their value allowing the test $=_{k-2}$ and finally the process $p1$ writes its value allowing the test $=_{k-1}$. But after this, the different processes are blocked because $p1$ cannot take a new value anymore and write it to allow the test $=_{k+m}$ for which all the processes need to choose the same new value and write it to the global memory.

To finalize our reduction we choose $\{q_f^{(1)}\}$ as the set of final states of the DMA. Since the size of the DMA we build is polynomial in the size of the m automata, we can deduce the lower bound for Fixed-Reachability.

▶ **Theorem 4.** *Fixed-Reachability is Pspace-complete.*

4 The parameterized train problem

We introduce in this section a simpler parameterized problem whose resolution will help in solving the reachability problem in DMA.

4.1 Definition

As for DMA, we will use here the set of tests $\mathcal{T} := \{=_t, <_t, >_t \mid t \in \mathbb{N}\}$. Our problem consists in modelling a set of passengers who can enter a train and leave it. Each passenger enters the train at most once and has the ability to test how many passengers are in the train and to change its state accordingly. Furthermore, there is a distinguished passenger, called the *controller*.

▶ **Definition 5.** *A* train automaton *is a tuple* $TA = (S, \iota^c, \iota, S_{out}, S_{in}, \Delta, s_f)$ *where* S *is the finite set of* states *partitioned into* $S = S_{out} \uplus S_{in} \uplus \{s_f\}$*,* $\iota^c \in S_{out}$ *is the* initial state *for the controller,* $\iota \in S_{out}$ *is the* initial state *for the passengers,* s_f *is the final state, and* $\Delta \subseteq (S_{out} \times \mathcal{T} \times S_{out}) \cup (S_{in} \times \mathcal{T} \times S_{in}) \cup (S_{out} \times \{\mathbf{E}\} \times S_{in}) \cup (S_{in} \times \{\mathbf{Q}\} \times \{s_f\})$ *is the finite set of* transitions*.*

Intuitively, when a passenger (or the controller) is in a state from S_{out} or in s_f, he stands outside the train, and when he is in S_{in}, he is inside the train. A passenger enters the train thanks to the action $\mathbf{E}$. He can leave the train with action $\mathbf{Q}$ and, in doing so, enters the state s_f from which he cannot perform any test or action. We now detail the semantics induced by TA.

For $n \geq 1$, an *n-train configuration* is a pair $\theta = (\mathbf{s}, c) \in S^n \times \mathbb{N}$ such that $\mathbf{s}[1]$ is the controller state and $c = |\{p \in [1, n] \mid \mathbf{s}[p] \in S_{in}\}|$. Note that we identify the controller with the first passenger. Formally, we could get rid of the c since we can deduce it from $\mathbf{s}$, but it eases the writing of our results to keep it. A *train configuration* is an n-train configuration for some $n \geq 1$. For an n-train configuration θ, we denote by $|\theta| = n$ its size. We say that θ is *initial* if $\mathbf{s}[1] = \iota^c$, $\mathbf{s}[p] = \iota$ for all $p \in [2, |\theta|]$, and $c = 0$. We define a transition relation $\rightarrow_{TA}$ as follows. Let $\theta = (\mathbf{s}, c)$ and $\theta' = (\mathbf{s}', c')$ be two train configurations, $a \in \mathcal{T} \cup \{\mathbf{E}, \mathbf{Q}\}$, and $p \in [1, |\theta|]$. We let $\theta \xrightarrow{(a,p)}_{TA} \theta'$ if $|\theta| = |\theta'|$, $\mathbf{s}[p'] = \mathbf{s}'[p']$ for all $p' \in [1, |\theta|] \setminus \{p\}$, $(\mathbf{s}[p], a, \mathbf{s}'[p]) \in \Delta$, and the following hold:

- if $a = \mathbf{E}$ then $c' = c + 1$ (passenger p enters the train),
- if $a = \mathbf{Q}$ then $c' = c - 1$ (passenger p leaves the train), and
- if $a \in \mathcal{T}$ then $c = c'$ and $c \models a$.

We write $\theta \rightarrow_{TA} \theta'$ if there exist $a \in \mathcal{T} \cup \{\mathbf{E}, \mathbf{Q}\}$ and $p \in [1, |\theta|]$ such that $\theta \xrightarrow{(a,p)}_{TA} \theta'$. An execution of TA is a finite sequence $\rho = \theta_0 \xrightarrow{(a_0,p_0)}_{TA} \theta_1 \xrightarrow{(a_1,p_1)}_{TA} \theta_2 \ldots \xrightarrow{(a_{k-1},p_{k-1})}_{TA} \theta_k$ (or $\rho = \theta_0 \rightarrow_{TA} \theta_1 \rightarrow_{TA} \theta_2 \ldots \rightarrow_{TA} \theta_k$ if we do not need the action and test labellings).

We denote by $\rightarrow^*_{TA}$ the reflexive and transitive closure of $\rightarrow_{TA}$. If $\theta \rightarrow^*_{TA} \theta'$, then we say that there exists an execution from θ to θ' in *TA*. Note that the number of passengers does not change during an execution, just like the number of processes does not change in an execution of a DMA.

The problem we study in this section can be formalized as follows:

TRAIN-REACHABILITY

I: A train automaton $TA = (S, \iota^c, \iota, S_{out}, S_{in}, \Delta, s_f)$ and a state $s \in S$

Q: Are there an initial train configuration θ and a configuration $\theta' = (\mathbf{s}', c')$ such that $\theta \rightarrow^*_{TA} \theta'$ and $\mathbf{s}'[p] = s$ for some $p \in [1, |\theta|]$?

We let TrainReach(*TA*) denote the set of states $s \in S$ such the answer to the TRAIN-REACHABILITY with *TA* and s is positive.

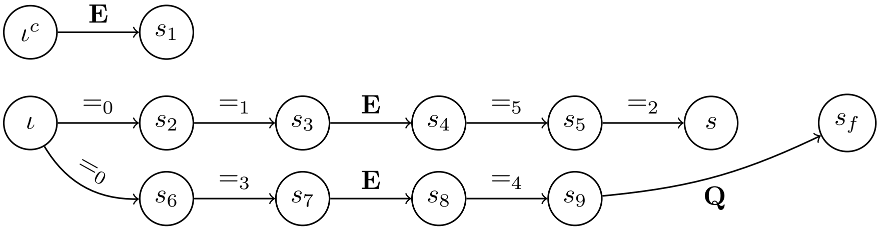


Figure 4 An example of train automaton.

▶ **Example 6.** In Figure 4, we have drawn a train automaton inspired (we shall see the connection later) from the DMA given in Figure 1. In this train automaton, the state s is not reachable. In fact, to reach it, the controller would have to go to state s_1 and at least two passengers to s_4. But then, there are at least three passengers in the train that cannot leave it anymore. Hence, the test $=_2$ can never be satisfied.

Train automata will help us to simulate part of the executions of DMAs where all the processes except one (the controller) begin by choosing a new value identical to the one of the controller (the idea being that this value corresponds to the identity of the train). Then, when a process performs a write, this corresponds to a passenger entering the train. Moreover, when, thanks to a sequence of actions, it overwrites its value in the global memory, this corresponds to a passenger leaving the train. This also explains why we need a controller in train automata: it helps to simulate a process which did not perform a new. Since, initially, all the processes have a different value in their global memory, there can be, for each value d, at most one process which did not perform a new(d) and has d in its local register.

4.2 Bounding the number of passengers

We will see here that in order to solve TRAIN-REACHABILITY, we can bound the number of passengers present in the train at any moment. Consider a train automaton $TA = (S, \iota^c, \iota, S_{out}, S_{in}, \Delta, s_f)$. We let $cap \in \mathbb{N}$ be the maximal constant appearing in the transitions of Δ. Hence we have $t \leq cap$ for all $(s, \bowtie_t, s') \in \Delta$. Given an n-train configuration $\theta = (\mathbf{s}, c)$ and a bound $b \in \mathbb{N}$, we say that θ is *b-bounded* if $c \leq b$. An execution $\theta_0 \rightarrow_{TA} \theta_1 \rightarrow_{TA} \theta_2 \ldots \rightarrow_{TA} \theta_k$ is called *b-bounded* if θ_i is b-bounded for all $i \in [0, k]$.

Finally, we introduce a relation $\preceq$ beween two train configurations $\theta = (\mathbf{s}, c)$ and $\theta' = (\mathbf{s}', c')$ defined as follows: $\theta \preceq \theta'$ if $|\theta| = |\theta'|$ and $c = c'$ and for all $p \in [1, |\theta|]$, if $\mathbf{s}[p] \neq \mathbf{s}'[p]$ then $\mathbf{s}[p] = s_f$ and $\mathbf{s}'[p] \in S_{out}$. In other words, if a passenger is not in the same state in θ

and in θ', it means he is in its final state in θ and he is out of the train in θ'. We need a first technical result stating that the relation $\preceq$ is a simulation relation for $\rightarrow_{TA}$. The result of this lemma is a direct consequence of the definition of $\preceq$ and of the fact that, in TA, when the controller or a passenger is in its final state, he cannot do anything anymore.

▶ **Lemma 7.** *If $\theta_1 \preceq \theta'_1$ and $\theta_1 \xrightarrow{(a,p)}_{TA} \theta_2$ then there exists a configuration θ'_2 such that $\theta_2 \preceq \theta'_2$ and $\theta'_1 \xrightarrow{(a,p)}_{TA} \theta'_2$.*

The following lemma shows us how to bound locally the capacity of the train. The idea is that if the capacity of the train goes above $cap + 2$, it is not necessary to make more passengers enter the train to satisfy the subsequent tests before the capacity goes back to a value smaller than $cap + 2$.

▶ **Lemma 8.** *Let $M > cap$. If there is an execution $\theta_0 \rightarrow_{TA} \theta_1 \rightarrow_{TA} \ldots \rightarrow_{TA} \theta_k$ with $\theta_i = (\mathbf{s}_i, c_i)$ for all $i \in [0, k]$ and such that $c_0 = c_K = M$ and $c_i = M + 1$ for all $i \in [1, k-1]$, then there is an M-bounded execution from θ_0 to some θ' with $\theta_k \preceq \theta'$.*

Proof. Let $\rho = \theta_0 \xrightarrow{(a_0,p_0)}_{TA} \theta_1 \xrightarrow{(a_1,p_1)}_{TA} \theta_2 \ldots \xrightarrow{(a_{k-1},p_{k-1})}_{TA} \theta_k$ be an execution with $\theta_i = (\mathbf{s}_i, c_i)$ for all $i \in [0, k]$ and such that $c_0 = c_k = M$ and $c_i = M + 1$ for all $i \in [1, k-1]$. By definition of the transition relation $\rightarrow_{TA}$ and of cap, we have necessarily $a_0 = \mathbf{E}$ and $a_{k-1} = \mathbf{Q}$ and $a_i = >_t$ with $M > cap \geq t$ for all $i \in [1, k-2]$. We distinguish two cases:

1. **Case** $p_0 = p_{k-1}$, i.e., it is the same process that enters and leaves the train. In that case, we let that process never enter the train and we consider the execution $\theta_0 \rightarrow_{TA} \theta'_1 \ldots \rightarrow_{TA} \theta'_\ell = (\mathbf{s}'_\ell, c'_\ell)$, obtained from ρ by deleting all the transitions (a, p) with $p = p_0$. During this execution the number of passengers in the train remains the same and is equal to $c_0 = M$ and, for all $p \in [1, |\theta_0|] \setminus \{p_0\}$, we have $\mathbf{s}'_\ell[p] = \mathbf{s}_k[p]$ and $\mathbf{s}'_\ell[p_0] = \mathbf{s}_0[p_0]$. Since $\mathbf{s}_0[p_0] \in S_{out}$ (because at the first step of ρ the passenger p_0 enters the train) and $\mathbf{s}_k[p_0] = s_f$ (because in the last step of ρ, passenger p_0 leaves the train), we deduce that $\theta_k \preceq \theta'_\ell$.
2. **Case** $p_0 \neq p_{k-1}$. In that case, we reorder the execution ρ as follows. First we execute all the transitions (a, p) with $p = p_{k-1}$ leading to a configuration $\theta'' = (\mathbf{s}'', c'')$ such that $\mathbf{s}''[p] = \mathbf{s}_0[p]$ for all $p \in [1, |\theta_0|] \setminus \{p_{k-1}\}$ and $\mathbf{s}''[p_{k-1}] = \mathbf{s}_k[p_{k-1}] = s_f$ and $c'' = M - 1$. Then from θ'' we execute, in the same order, the remaining transition of ρ (the first being labelled with $(\mathbf{E}, p_0)$) which leads exactly to the configuration θ_k. Hence we obtain an M-bounded execution from θ_0 to θ_k. ◀

Using iteratively this last lemma allows us to bound the number of passengers in the train to reach a specific control state s.

▶ **Proposition 9.** *Let $s \in S$. Let θ be an initial configuration and $p \in [1, |\theta|]$. If there is an execution from θ to some configuration $\theta' = (\mathbf{s}', c')$ with $\mathbf{s}'[p] = s$, then there is a $(cap + 2)$-bounded execution from θ to some configuration $\theta'' = (\mathbf{s}'', c'')$ with $\mathbf{s}''[p] = s$.*

4.3 Solving Train-Reachability

We shall see now how Proposition 9 allows us to build a finite abstract graph in which the reachability problem provides us with a solution for TRAIN-REACHABILITY. We consider a train automaton $TA = (S, \iota^c, \iota, S_{out}, S_{in}, \Delta, s_f)$ and, as in the previous section, we let $cap \in \mathbb{N}$ be the maximal constant appearing in the transitions of Δ. In order to solve our reachability problem, we build a graph of abstract configurations which keep track of the states of the controller, of the states in S_{out} that can be reached, and of the number of people in the train up to $cap+2$. As we shall see, such an abstract graph will suffice to obtain a witness for TRAIN-REACHABILITY thanks to the Proposition 9 and to the following Copycat Lemma.

▶ **Lemma 10** (Copycat Lemma). *Let $s \in S_{out}$ and $M > 0$. Assume an M-bounded execution from an initial train configuration θ_0 to a configuration $\theta = (\mathbf{s}, c)$ with $\mathbf{s}[p] = s$ for some $p \in [2, |\theta_0|]$. Then, for all $b \geq 0$, there exists an M-bounded execution from θ'_0 to $\theta' = (\mathbf{s}', c)$ where θ'_0 is the initial train configuration with $|\theta'_0| = |\theta_0| + b$, $\mathbf{s}'[p] = \mathbf{s}[p]$ for all $p \in [1, |\theta_0|]$, and $\mathbf{s}'[p] = s$ for all $p \in [|\theta_0|+1, |\theta_0|+b]$.*

Proof. Let $\rho = \theta_0 \xrightarrow{(a_0,p_0)}_{TA} \theta_1 \xrightarrow{(a_1,p_1)}_{TA} \theta_2 \ldots \xrightarrow{(a_{k-1},p_{k-1})}_{TA} \theta_k$ be an execution with $\theta_i = (\mathbf{s}_i, c_i)$ for all $i \in [0,k]$ and $\mathbf{s}_k[p] \in S_{out}$ for $p \in [2, |\theta_0|]$. Since, in TA, a passenger can never go to a state in S_{out} once he has entered the train, $p_i = p$ implies $a_i \in \mathcal{T}$ for all $i \in [0, k-1]$. In other words, all the actions performed by passenger p along ρ are tests. Hence from θ'_0, we can reproduce ρ and each time we have $p_i = p$, passengers $|\theta_0|+1$ to $|\theta_0|+b$ take the same transition as passenger p. As a consequence, at the end of this run, all these passengers will be in the same state as passenger p, and extending ρ in such a way is possible because the actions of passenger p never change the capacity of the train, as they are just tests. ◀

An abstract train configuration ξ of TA is a triple (s^c, Out, In) where $s^c \in S$, $Out \subseteq S_{out} \cup \{s_f\}$ and $In \in \mathbb{N}^{S_{in}}$ is a multiset of elements of S_{in} such that $\sum_{s \in S_{in}} In(s) \leq cap+1$ if $s^c \in S_{in}$ and $\sum_{s \in S_{in}} In(s) \leq cap+2$ otherwise. Given an abstract configuration $\xi = (s^c, Out, In)$, we define $inside(\xi) \in [0, cap+2]$ describing the number of passengers in the train: it is equal to $\sum_{s \in S_{in}} In(s)$ if $s^c \notin S_{in}$ and $1 + \sum_{s \in S_{in}} In(s)$ otherwise. Indeed, by definition, we have $inside(\xi) \leq cap+2$ for all abstract train configurations ξ. The initial abstract train configuration ξ_ι is then equal to $(\iota^c, \{\iota\}, In_\iota)$ with $In_\iota(s) = 0$ for all $s \in S_{in}$. We denote by Ξ the set of abstract train configurations of TA. Note that by definition Ξ is finite.

We define now a transition relation $\rightsquigarrow$ between abstract configurations. Let $\xi_1 = (s^c_1, Out_1, In_1)$ and $\xi_2 = (s^c_2, Out_2, In_2)$ be two abstract train configurations and $\delta = (s, a, s') \in \Delta$ and $mc = \{\top, \bot\}$. The value mc indicates whether the controller moves ($\top$) or another passenger ($\bot$). We have $\xi_1 \overset{\delta,mc}{\rightsquigarrow} \xi_2$ if one of the following cases holds:

1. $mc = \top$ and $s = s^c_1$ and $s' = s^c_2$ and $Out_1 = Out_2$ and $In_1 = In_2$ and if $a = \mathbf{E}$ then $inside(\xi_1) < cap+2$ and if $a \in \mathcal{T}$ then $inside(\xi_1) \models a$ (move of the controller);
2. $mc = \bot$ and $s^c_1 = s^c_2$ and $s \in Out$ and $a \in \mathcal{T}$ and $inside(\xi_1) \models a$ and $Out_2 = Out_1 \cup \{s'\}$ and $In_2 = In_1$ (move of a passenger outside the train);
3. $mc = \bot$ and $s^c_1 = s^c_2$ and $s \in S_{in}$ and $In_1(s) > 0$ and $a \in \mathcal{T}$ and $inside(\xi_1) \models a$ and $Out_2 = Out_1$ and
   - $In_2(s) = In_1(s) - 1$ and $In_2(s') = In_1(s') + 1$ if $s \neq s'$,
   - $In_2(s) = In_1(s)$ if $s = s'$

   and $In_2(s'') = In_1(s'')$ for all $s'' \in S_{in} \setminus \{s, s'\}$ (move of a passenger in the train);
4. $mc = \bot$ and $s^c_1 = s^c_2$ and $s \in Out$ and $a = \mathbf{E}$ and $inside(\xi_1) < cap+2$ and $Out_2 = Out_1$ and $In_2(s') = In_1(s') + 1$ and $In_2(s'') = In_1(s'')$ for all $s'' \in S_{in} \setminus \{s'\}$ (a passenger enters the train);
5. $mc = \bot$ and $s^c_1 = s^c_2$ and $s \in S_{in}$ and $In_1(s) > 0$ and $a = \mathbf{Q}$ and $Out_2 = Out_1 \cup \{s_f\}$ and $In_2(s) = In_1(s) - 1$ and $In_2(s'') = In_1(s'')$ for all $s'' \in S_{in} \setminus \{s\}$ (a passenger leaves the train).

We write $\xi_1 \rightsquigarrow \xi_2$ if there exist $\delta \in \Delta$ and $mc = \{\top, \bot\}$ such that $\xi_1 \overset{\delta,mc}{\rightsquigarrow} \xi_2$, and we denote by $\rightsquigarrow^*$ the reflexive and transitive closure of $\rightsquigarrow$.

We shall now see how we can reduce Train-Reachability to a reachability query in the transition system $(\Xi, \rightsquigarrow)$. In other words, we shall prove in which matters our abstraction is sound and complete for Train-Reachability. The results of the two next lemmas need to be combined with the result of Proposition 9 which states that we can restrict our attention to $(cap + 2)$-bounded executions to solve Train-Reachability. First we give the lemma needed to ensure completeness of our abstraction. For this, given an abstract train configuration $\xi = (s^c, Out, In)$, we define $[\![\xi]\!]$, a set of configurations described by ξ. For a train configuration $\theta = (\mathbf{s}, c)$, we let $\theta \in [\![\xi]\!]$ if the following conditions hold:

- $c = inside(\xi)$,
- $\mathbf{s}[1] = s^c$,
- for all $p \in [2, |\theta|]$, if $\mathbf{s}[p] \in S_{out} \cup \{s_f\}$ then $\mathbf{s}[p] \in Out$, and
- $In(s) = |\{p \in [2, |\theta|] \mid \mathbf{s}[p] = s\}|$ for all $s \in S_{in}$.

In other words, the control state of the controller is the same in θ and ξ, the states of the passengers in the train are the same in ξ and θ, and all the states present in θ from passengers outside the train are present in Out. This interpretation of abstract configurations allows us to state our first property.

▶ **Lemma 11.** *Let θ and θ' be two configurations such that θ is initial. If there is a $(cap+2)$-bounded execution from θ to θ' then there exists an abstract train configuration ξ' such that $\theta' \in [\![\xi']\!]$ and $\xi_\iota \rightsquigarrow^* \xi'$.*

To ensure the soundness of our method, for an abstract train configuration $\xi = (s^c, Out, In)$, we need to identify in $[\![\xi]\!]$ the configurations for which all the states in Out are present. We say that a configuration $\theta = (\mathbf{s}, c)$ is a witness for ξ if $\theta \in [\![\xi]\!]$ and, for all $s \in Out$, there exists $p \in [2, |\theta|]$ such that $\mathbf{s}[p] = s$. This new notion combined with the result of the Copycat Lemma 10 allows us to state the following property of our abstraction.

▶ **Lemma 12.** *Let $\xi' \in \Xi$. If $\xi_\iota \rightsquigarrow^* \xi'$ then there exist an initial configuration θ and $\theta' \in [\![\xi']\!]$ such that there is a $(cap + 2)$-bounded execution from θ to θ' and θ' is a witness for ξ'.*

Now to solve Train-Reachability for the train automaton TA and a state $s \in S$, thanks to Proposition 9, we know it is enough to consider only $(cap + 2)$-bounded executions. Lemmas 11 and 12 tell us that we have to seek in the graph $(\Xi, \rightsquigarrow)$ a path between ξ_ι and an abstract train configuration $\xi = (s^c, Out, In)$ such that $s = s^c$ or $s \in Out$ or $In(s) > 0$. Note that by definition $|\Xi| \leq |S^c| \cdot 2^{|S_{out}|+1} \cdot |S_{in}|^{cap+2}$ hence the size of $(\Xi, \rightsquigarrow)$ is exponential in the size of TA and the transition relation $\rightsquigarrow$ can be built on-the-fly (as it is done in its definition). Using that the reachability problem in a graph can be solved in NLOGspace, we deduce that we can solve Train-Reachability in NPspace (by solving a reachability query in $(\Xi, \rightsquigarrow)$). Thanks to Savitch's theorem we deduce our Pspace upper bound.

▶ **Theorem 13.** *Train-Reachability is in Pspace.*

5 An algorithm for reachability

In this section, we provide an algorithm to solve Reachability using, as an internal procedure, the algorithm proposed in the previous section for Train-Reachability.

We consider a DMA $\mathcal{A} = (S, \iota, \Delta, F)$. Without loss of generality, we assume that in $\mathcal{A}$ when a process p performs a write action, then it will not do so again until it performs a new action. This restriction makes sense, since when it has written its local value once, it does not change anything to the behavior of the global system to rewrite it. One can easily modify $\mathcal{A}$ to respect this property by adding a boolean flag to the states which is set to true after a write and set back to false after a new. Moreover, when an edge labelled with write leaves a state while the newly introduced boolean is true, then write is replaced by the test $>_0$ (which will be necessarily evaluated to true since the global memory contains at least the local value of the process). Before presenting our method to solve Reachability, we state a technical lemma similar to the Copycat Lemma 10, but this time for DMA instead of train automata. The idea here is that we can join two distinct executions of the DMA using the fact that in DMA, the precise values of the data written in the global or local memory do not really matter but only the occurrences of the same values are important.

▶ **Lemma 14** (Copycat Lemma II). *If there exists an execution $\gamma_0 \Longrightarrow^*_{\mathcal{A}} \gamma_1$ with γ_0 initial and $\gamma_1 = (\mathbf{s}_1, \boldsymbol{\ell}_1, \mathbf{M}_1)$ and an execution $\gamma'_0 \Longrightarrow^*_{\mathcal{A}} \gamma'_1$ with γ'_0 initial and $\gamma'_1 = (\mathbf{s}'_1, \boldsymbol{\ell}'_1, \mathbf{M}'_1)$, then there exists an execution $\gamma''_0 \Longrightarrow^*_{\mathcal{A}} \gamma''_1$ with γ''_0 initial and such that $|\gamma''_1| = |\gamma_1| + |\gamma'_1|$ and $\gamma''_1 = (\mathbf{s}''_1, \boldsymbol{\ell}''_1, \mathbf{M}''_1)$ with $\mathbf{s}''_1[p] = \mathbf{s}_1[p]$ for all $p \in [1, |\gamma_1|]$ and $\mathbf{s}''_1[|\gamma_1| + p] = \mathbf{s}'_1[p]$ for all $p \in [1, |\gamma'_1|]$.*

As a consequence of this lemma, if at some point we reach a configuration γ_1 in a DMA, we know that any configuration with as many copies as one may desire of the states of γ_1 is reachable. Our algorithm for Reachability then computes, iteratively, the two following subsets of the set of states S:

- $\mathcal{N}\mathsf{ew}$ is the set of reachable states $s \in S$ from which an action new is feasible. Formally, $s \in \mathcal{N}\mathsf{ew}$ if there exist $\gamma, \gamma' \in \mathbb{C}_{\mathcal{A}}$ such that γ is initial and γ' and $\gamma \Longrightarrow^*_{\mathcal{A}} \gamma'$ and $s \in \mathit{states}(\gamma')$ and $(s, \mathsf{new}, u) \in \Delta$ for some $u \in S$.
- $\mathcal{OW}\mathsf{rite}$ is the set of states $s \in S$ that occur in some execution where the process being in s performs new and eventually write (hence the set of states from which a process can overwrite its value in the global memory). Formally, $s \in \mathcal{OW}\mathsf{rite}$ if there exist a run ρ of $\mathcal{A}$ of the form $\gamma_0 \xRightarrow{(\sigma_0,p_0)}_{\mathcal{A}} \gamma_1 \xRightarrow{(\sigma_1,p_1)}_{\mathcal{A}} \gamma_2 \cdots \xRightarrow{(\sigma_\ell,p_\ell)}_{\mathcal{A}} \gamma_{\ell+1}$ and $p \in [1, |\gamma_0|]$ and $0 \leq j < k \leq \ell$ such that $\gamma_j = (\mathbf{s}_j, \boldsymbol{\ell}_j, \mathbf{M}_j)$ with $\mathbf{s}_j[p] = s$ and $(\sigma_j, p_j) = (\mathsf{new}, p)$ and $(\sigma_k, p_k) = (\mathsf{write}, p)$ and, for all $i \in [j+1, \ell-1]$, if $p_i = p$ then $\sigma_i \notin \{\mathsf{new}, \mathsf{write}\}$.

First, note that $\mathcal{OW}\mathsf{rite} \subseteq \mathcal{N}\mathsf{ew}$. We will see now how to compute these two sets of states and how our method exploits the result of the previous section on the train problem. The intuition to link the reachability in DMA with this latter problem is the following: each process in a DMA is associated to a train whose number is the value stored in its local register. When a process writes its value to the global memory, it enters the corresponding train and it stays in it until it overwrites this value by another one (by entering a new train).

We first explain, given two sets of states $\mathcal{N} \subseteq \mathcal{N}\mathsf{ew}$ and $\mathcal{OW} \subseteq \mathcal{OW}\mathsf{rite}$, how to build a train automaton $\mathit{TA}_{\mathrm{N}}(\mathcal{N}, \mathcal{OW})$ to check whether new states can be added to $\mathcal{N}$. We define $\mathit{TA}_{\mathrm{N}}(\mathcal{N}, \mathcal{OW}) = (S_T, \iota^c_T, \iota_T, S_{out}, S_{in}, \Delta_T, s_f)$ with:

- $S_T = (S \times \{\mathit{out}, \mathit{in}\}) \cup \{\iota_T, s_f\}$,
- $S_{out} = (S \times \{\mathit{out}\}) \cup \{\iota_T\}$,
- $S_{in} = S \times \{\mathit{in}\}$,
- $\iota^c_T = (\iota, \mathit{out})$,
- Δ_T is the set of transitions verifying:
  - $(\iota_T, =_0, (u, \mathit{out})) \in \Delta_T$ for all $u \in S$ such that there is $(s, \mathsf{new}, u) \in \Delta$ with $s \in \mathcal{N}$,
  - $((s, \mathit{out}), \mathbf{E}, (s', \mathit{in})) \in \Delta_T$ for all $(s, \mathsf{write}, s') \in \Delta$,
  - $((s, \mathit{in}), \mathbf{Q}, s_f) \in \Delta_T$ for all $s \in \mathcal{OW}$,
  - $((s, \mathit{out}), a, (s', \mathit{out})), ((s, \mathit{in}), a, (s', \mathit{in}))$ for all $(s, a, s') \in \Delta$ with $a \in \mathcal{T}$.

In a DMA, when a process performs a new, it is always possible that it chooses the initial value of another process that has not been written yet to the global memory. However, for a given value, there is at most one such process since, initially, all the processes have pairwise different values in their local memory. Such a process is represented in $TA_{\mathrm{N}}(\mathcal{N}, \mathcal{OW})$ by the distinguished controller. Hence, the initial state of the controller is (ι, out). All the other processes have to perform a new and are represented by the other passengers. To participate in the train automaton, they have to go through the transitions $(\iota_T, =_0, (u, out))$ such that there is $(s, \mathsf{new}, u) \in \Delta$ with $s \in \mathcal{N}$. The train automaton $TA_{\mathrm{N}}(\mathcal{N}, \mathcal{OW})$ then simulates the DMA with the following rules: When a passenger enters the train with $\mathbf{E}$, the associated process writes its value to the current memory, and when he leaves the train with $\mathbf{Q}$, the associated process has been able to choose a new value and to write it to the global memory, so intuitively it was in a state of $\mathcal{OW}$.

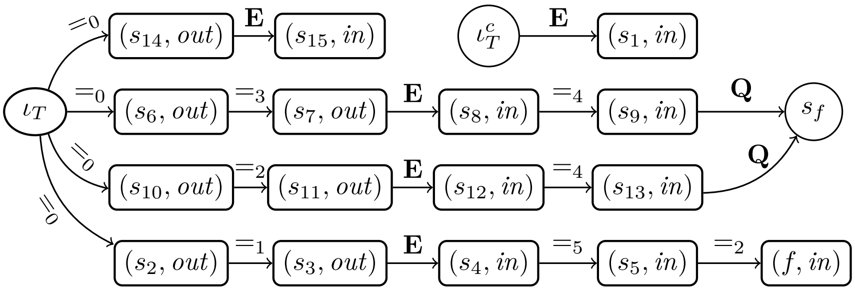


Figure 5 Train Automaton $TA_{\mathrm{N}}(\{\iota, s_9, s_{13}\}, \{\iota, s_9, s_{13}\})$ for the DMA of Figure 1.

▶ **Example 15.** Figure 5 depicts the train automaton $TA_{\mathrm{N}}(\mathcal{N}, \mathcal{OW})$ associated to the DMA of Figure 1 with $\mathcal{N} = \{\iota, s_9, s_{13}\}$ and $\mathcal{OW} = \{\iota, s_9, s_{13}\}$. Thanks to this train automaton, we deduce that f is reachable in the DMA because $(f, in) \in \mathsf{TrainReach}(TA_{\mathrm{N}}(\mathcal{N}, \mathcal{OW}))$. We have indeed the following execution with five passengers (numbered from 1 to 5, where 1 is the controller): First, passengers 2 to 4 move to (s_{10}, out), and passenger 5 moves to (s_2, out). Then, the controller enters the train and arrives in (s_1, in). After that, passenger 5 can go to (s_4, in) entering the train. There are now two passengers in the train, so passengers 2 to 4 can go to (s_{11}, out) and passengers 2 to 3 can enter the train and move to (s_{13}, in) since there will be four passengers in the train. Finally, passenger 4 enters the train. There are now five passengers in the train allowing passenger 5 to move to (s_5, in). After that, passenger 2 in (s_{13}, in) can leave the train, and passenger 4 can move to (s_{13}, in). Now, passengers 3 and 4 from $(s_{13}, in))$ can leave the train bringing the number of passengers to two which allows passenger 5 to reach (f, in).

Thanks to Lemma 14 (Copycat Lemma) and to the semantics of train automata, we deduce the following:

▶ **Lemma 16.** *Let $\mathcal{N} \subseteq \mathsf{New}$, $\mathcal{OW} \subseteq \mathsf{OWrite}$, and $s \in S$. If we have $\{(s, in), (s, out)\} \cap \mathsf{TrainReach}(TA_{\mathrm{N}}(\mathcal{N}, \mathcal{OW})) \neq \emptyset$ and $(s, \mathsf{new}, u) \in \Delta$ for some $u \in S$, then $s \in \mathsf{New}$.*

Hence this last lemma allows us to add new states from New to $\mathcal{N}$. We will now see how to increase the set of states $\mathcal{OW}$. The idea is similar but we give as input a state sn in $\mathcal{N}$ from which we want to check whether an action write can be reached. In the train automaton, we hence have to check which states are reachable from this state sn. For this matter, we use an extra symbol, $\top$ or $\bot$, to track the path coming from sn (this symbol equals $\top$ when the state is reachable from sn). Given two sets of states $\mathcal{N} \subseteq \mathsf{New}$ and $\mathcal{OW} \subseteq \mathsf{OWrite}$ and

$\mathit{sn} \in \mathcal{N}$, we build a train automaton $\mathit{TA}_{\mathrm{OW}}(\mathcal{N}, \mathcal{OW}, \mathit{sn})$ to check whether sn can be added to $\mathcal{OW}$. We let $\mathit{TA}_{\mathrm{OW}}(\mathcal{N}, \mathcal{OW}, \mathit{sn}) = (S_T, \iota^c_T, \iota_T, S_{out}, S_{in}, \Delta_T, s_f)$ with:

- $S_T = (S \times \{\mathit{out}, \mathit{in}\} \times \{\top, \bot\}) \cup \{\iota_T, s_f\}$
- $S_{out} = (S \times \{\mathit{out}\} \times \{\top, \bot\}) \cup \{\iota_T\}$,
- $S_{in} = S \times \{\mathit{in}\} \times \{\top, \bot\}$,
- $\iota^c_T = (\iota, \mathit{out}, \bot)$,
- Δ_T is the set of transitions verifying:
  - $(\iota_T, =_0, (u, \mathit{out}, \bot)) \in \Delta_T$ for all $u \in S$ such that there is $(s, \mathsf{new}, u) \in \Delta$ with $s \in \mathcal{N}$,
  - $(\iota_T, =_0, (u, \mathit{out}, \top)) \in \Delta_T$ for all $u \in S$ such that $(\mathit{sn}, \mathsf{new}, u) \in \Delta$,
  - $((s, \mathit{out}, v), \mathbf{E}, (s', \mathit{in}, v)) \in \Delta_T$ for all $(s, \mathsf{write}, s') \in \Delta$ and $v \in \{\top, \bot\}$,
  - $((s, \mathit{in}, v), \mathbf{Q}, s_f) \in \Delta_T$ for all $s \in \mathcal{OW}$ and $v \in \{\top, \bot\}$,
  - $((s, \mathit{out}, v), a, (s', \mathit{out}, v)), ((s, \mathit{in}, v), a, (s', \mathit{in}, v))$ for all $(s, a, s') \in \Delta$ with $a \in \mathcal{T}$ and all $v \in \{\top, \bot\}$.

Hence in this train automaton, if a state $(s, \mathit{in}, \top)$ or $(s, \mathit{out}, \top)$ is reached, the passenger reaching this state necessarily went through the state $(\mathit{sn}, \mathit{out}, \top)$. We have the following result whose correcteness can be proved the same way as for Lemma 16.

▶ **Lemma 17.** *Let $\mathcal{N} \subseteq \mathcal{N}\mathsf{ew}$, $\mathcal{OW} \subseteq \mathcal{OW}\mathsf{rite}$, and $\mathit{sn} \in \mathcal{N}$. If there exists $s \in S$ such that $(s, \mathit{out}, \top) \in \mathsf{TrainReach}(\mathit{TA}_{\mathrm{OW}}(\mathcal{N}, \mathcal{OW}, \mathit{sn}))$ and such that $(s, \mathsf{write}, u) \in \Delta$ for some $u \in S$, then $\mathit{sn} \in \mathcal{OW}\mathsf{rite}$.*

These two last lemmas give us a technique to compute the sets $\mathcal{N}\mathsf{ew}$ and $\mathcal{OW}\mathsf{rite}$. We present a procedure that computes iteratively two families of sets of states $(\mathcal{N}_i)_{i \in \mathbb{N}}$ and $(\mathcal{OW}_i)_{i \in \mathbb{N}}$ such that $\mathcal{N}_i \subseteq \mathcal{N}_{i+1} \subseteq \mathcal{N}\mathsf{ew}$ and $\mathcal{OW}_i \subseteq \mathcal{OW}_{i+1} \subseteq \mathcal{OW}\mathsf{rite}$ for all $i \in \mathbb{N}$. We set $\mathcal{N}_0 = \mathcal{OW}_0 = \emptyset$ and, for all $i \in \mathbb{N}$:

- $\mathcal{N}_{i+1} = \mathcal{N}_i \ \cup \ \left\{ s \in S \;\middle|\; \begin{array}{l} \{(s, \mathit{in}), (s, \mathit{out})\} \cap \mathsf{TrainReach}(\mathit{TA}_{\mathrm{N}}(\mathcal{N}_i, \mathcal{OW}_i)) \neq \emptyset \text{ and} \\ (s, \mathsf{new}, u) \in \Delta \text{ for some } u \in S \end{array} \right\}$
- $\mathcal{OW}_{i+1} = \mathcal{OW}_i \cup \left\{ \mathit{sn} \in \mathcal{N}_{i+1} \;\middle|\; \begin{array}{c} \exists s \in S. \\ (s, \mathit{out}, \top) \in \mathsf{TrainReach}(\mathit{TA}_{\mathrm{OW}}(\mathcal{N}_{i+1}, \mathcal{OW}_i, \mathit{sn})) \text{ and} \\ (s, \mathsf{write}, u) \in \Delta \text{ for some } u \in S \end{array} \right\}$

Note that, since the set of states S is finite, these computations terminate and, thanks to Theorem 13, we know they are in PSPACE. We define $\mathcal{N} = \bigcup_{i \in \mathbb{N}} \mathcal{N}_i$ and $\mathcal{OW} = \bigcup_{i \in \mathbb{N}} \mathcal{OW}_i$. Due to Lemmas 16 and 17, we have $\mathcal{N} \subseteq \mathcal{N}\mathsf{ew}$ and $\mathcal{OW} \subseteq \mathcal{OW}\mathsf{rite}$. We can also obtain the inclusion in the other directions by reasoning by induction on the length of the executions of the DMA and looking at the processes that can create a new value or can overwrite their value in the global memory in such executions.

▶ **Lemma 18.** *We have $\mathcal{N} = \mathcal{N}\mathsf{ew}$ and $\mathcal{OW} = \mathcal{OW}\mathsf{rite}$.*

Now, to conclude, we can assume w.l.o.g. that, from each of the final states s in F, there is a transition (s, new, s') in Δ (if not we can add one) and hence solving REACHABILITY amounts at verifying whether $F \cap \mathcal{N} \neq \emptyset$. Since, as said earlier, $\mathcal{N}$ and $\mathcal{OW}$ can be computed in PSPACE, this allows us to deduce the following theorem:

▶ **Theorem 19.** *REACHABILITY is in PSPACE.*

6 Conclusion

We have shown that the control-state reachability problem for DMAs is in PSPACE when the number of processes is a parameter and is PSPACE-complete when this number is fixed. The upper-bound for the parameterized case is obtained thanks to an algorithm which uses as a sub-routine a polynomial-space solution for the control-state reachability in train automata. If we could find a better complexity bound, such as P or NP, for TRAIN-REACHABILITY, this bound will also apply to REACHABILITY in DMAs. Similarly, if we find another algorithm to solve REACHABILITY in DMAs with a better upper bound, this would lead to a better solution for TRAIN-REACHABILITY (which can easily be encoded into REACHABILITY for DMAs). In fact, we currently do not have any lower bound for these two problems and the proof to obtain the lower bound for FIXED-REACHABILITY crucially depends on the fact that we know the number of involved processes. In the future, we plan to further study the TRAIN-REACHABILITY problem and some of its extensions to see how the reasoning presented here can be applied to verify concrete distributed algorithms.

References

1 C. Aiswarya, Benedikt Bollig, and Paul Gastin. An automata-theoretic approach to the verification of distributed algorithms. *Inf. Comput.*, 259(Part 3):305–327, 2018.

2 Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. In Erzsébet Csuhaj-Varjú and Zoltán Ésik, editors, *Fundamentals of Computation Theory, 16th International Symposium, FCT 2007*, volume 4639 of *Lecture Notes in Computer Science*, pages 88–99. Springer, 2007.

3 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.

4 Mikolaj Bojanczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011.

5 Benedikt Bollig, Patricia Bouyer, and Fabian Reiter. Identifiers in registers - describing network algorithms with logic. In Mikolaj Bojanczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019*, volume 11425 of *Lecture Notes in Computer Science*, pages 115–132. Springer, 2019.

6 Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. The renaming problem in shared memory systems: An introduction. *Comput. Sci. Rev.*, 5(3):229–251, 2011.

7 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009.

8 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.

9 Wan Fokkink. *Distributed Algorithms: An Intuitive Approach*. MIT Press, 2013.

10 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

11 Dexter Kozen. Lower bounds for natural proof systems. In *FOCS'77*, pages 254–266. IEEE Computer Society, 1977.

12 Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.

13 Nikos Tzevelekos. Fresh-register automata. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011*, pages 295–306. ACM, 2011.

Pregrammars and Intersection Types

Sabine Broda
CMUP, Departamento de Ciência de Computadores, Faculdade de Ciências,
University of Porto, Portugal

Abstract

A representation of intersection types in terms of pregrammars is presented. Pregrammar based rewriting relations, corresponding respectively to type checking and inhabitation are defined and the latter is used to implement a Wajsberg/Ben-Yelles style alternating semi-decision algorithm for inhabitation. The usefulness of the framework is illustrated by revisiting and partially extending standard inhabitation related results for intersection types, as well as establishing new ones. It is shown how the notion of bounded multiset dimension emerges naturally and the relation between the two settings is clarified. A meaningful rank independent superset of the set of rank 2 types is identified for which EXPSPACE-completeness for inhabitation as well as for counting is proved. Finally, a standard result on negatively non-duplicated simple types is extended to intersection types.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Logic; Theory of computation → Lambda calculus

Keywords and phrases Intersection Types, Pregrammars, Inhabitation

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.14

Funding The author was partially supported by CMUP, which is financed by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the project with reference UIDB/00144/2020.

1 Introduction

1.1 Contribution and Related Work

Intersection type systems [12, 13] play an important role in the theory of typed λ-calculus [5]. They characterise exactly the set of strongly normalising terms and are closely related to the model theory of λ-calculus [33, 25, 14]. Applications in programming language theory cover a variety of diverse topics including the design of programming languages [36], program analysis and model checking [30, 31, 34], synthesis [15, 21], and related systems such as refinement and union types [22, 19, 20]. But the enormous expressive power of intersection types also comes with a major drawback. Standard type theoretic problems such as type checking and inhabitation are undecidable in general [5]. For inhabitation, undecidability is known for about twenty years [42]. More recently [37, 38] the problem was shown to be equivalent to the undecidable problem of λ-definability [32, 29]. Over the years attempts have been made in various directions with the purpose of identifying and studying the complexity of decidable fragments or variants of the system. Those include restrictions by rank [28, 27, 43], restrictions of admissible type inference rules [26, 35, 11], calculi of bounded dimension [16, 18], as well as systems of non-idempotent intersection types [10, 9]. In terms of rank the dividing line between decidable and undecidable cases was established in [27, 43] and complexity results were given for the decidable cases. As such, inhabitation was shown to be PSPACE-complete for rank 1, EXPSPACE-complete for rank 2, and undecidable for types of rank ≥ 3. Recently, the notion of dimension was introduced [16, 17] measuring intersection introduction as a resource. The inhabitation problem was shown to be decidable and EXPSPACE-complete in bounded multiset dimension, exposing thereby a rank independent calculus with a decidable inhabitation problem. In particular, it was shown to subsume the same result for rank 2 inhabitation.

The present work borrows some inspiration from [40] and aims to contribute to this line of research providing a simple tool for addressing inhabitation related problems. Lower bounds of specific problems are typically established using reductions from problems for which a bound (resp. undecidability) is already known. For upper bounds the standard procedure is based on the implementation of a Wajsberg/Ben-Yelles style search algorithm [6, 23, 11, 43]. The algorithm looks for inhabitants in β-normal form and operates on a set of pairs, each consisting of a context and a type. Depending on the rule used in each step the algorithm manipulates the set of pairs accordingly. The present work was motivated by the desire to provide a simple formalisation of this transformation process, expounding changes in each execution step in a clear and organised manner. The primary goal was to facilitate reasoning about inhabitation problems for intersection types, as well as to provide a clean framework in which proofs can be presented. Additionally, a graphical representation of intersection types exposes their fine underlying structure contributing to the end of clarification. That insight enables us to add further to the knowledge on decidable fragments of the intersection type system. From a practical point of view, we believe this work, and in particular pregrammar based implementations of a Wajsberg/Ben-Yelles search algorithm, to be of importance for applications in proof/inhabitant search based program synthesis in systems with intersection [21].

Picking up on work in [1, 2, 3] we here extend the notion of pregrammar to the entire system of intersection types. In [1] pregrammars were introduced in the context of (principal) inhabitation of simple types. Later [2] PSPACE upper bounds were (re-)proved for different inhabitation related problems for simple types in the framework of pregrammars. A first attempt to generalise those concepts to apply to intersection types was made in [3] and covered a restricted fragment of rank 2 types, therein denoted by $\mathcal{T}_2^-$. Additionally, the notion of pregrammar was extended to apply to the set of finite intersections of $\mathcal{T}_2^-$-types, which properly contains the set of strict intersection types [44] of rank 2 (but not all rank 2 types). For $\mathcal{T}_2^-$-types it is sufficient to consider a type inference system without the intersection introduction rule ($\cap$I), which was the approach taken in [3]. Intersections of $\mathcal{T}_2^-$-types introduce new intersections at the top level only and could therefore be handled by considering products of pregrammars. Neither considering a system without rule ($\cap$I), nor using products of pregrammars, works if one wants to cover the entire system, for which a different approach has to be taken. In the present work we consider a type inference system for terms in β-normal form, equivalent to that in [43], with rule ($\cap$I) but without an explicit intersection elimination rule ($\cap$E). This enables us to avoid the restriction to strict types, which was the approach taken in [18], and still handle the case of application in our search algorithm strictly syntax driven, contrasting with the $\vdash$-based condition in [43].

1.2 Outline

The paper is organised as follows. Section 2 contains the necessary definitions and results on the system of intersection types and sets the ground for subsequent developments. The definition of pregrammars for intersection types is in Section 3. In Sections 4 and 5 we define sound and complete rewriting relations based on pregrammars, corresponding respectively to type checking and inhabitation. Transformation in contexts during proof search are therein captured in terms of operations (update and replication) on tuples of integers. A Wajsberg/Ben-Yelles style alternating semi-decision algorithm for inhabitation is defined in

Section 6. In the remaining of that section the usefulness of our framework is demonstrated by revisiting and partially extending standard inhabitation related results, as well as establishing new ones. It is shown how the notion of bounded multiset dimension emerges naturally in this setting and the relation between the two is clarified. A meaningful rank independent superset of the set of rank 2 types is identified for which EXPSPACE-completeness is proved. Making use of the aforementioned operations on tuples, EXPSPACE-completess of counting in this fragment is also shown. Finally, a standard result on negatively non-duplicated simple types is extended to intersection types. A compilation of complexity results for different families of types can be found in the conclusions.

It is important to note that the majority of the discussed results can be obtained by the classical method. Nonetheless, it is our belief that the work in this paper constitutes further a step towards a better understanding of inhabitation in the intersection type system by highlighting how intersections at different depth and of different polarity contribute differently to the complexity of the inhabitation problem.

2 Preliminaries

We consider the basic system of intersection types, with no constants nor subtyping, cf. [43]. Type variables are ranged over by $a, b, c, \ldots$ and arbitrary types by lower-case Greek letters. The set of simple types. i.e. types without occurrences of the intersection operator, is denoted by $\mathcal{T}$.

▶ **Definition 1** (Intersection Types). $\mathcal{T}_\cap \ni \sigma, \tau ::= a \mid \sigma \to \tau \mid \sigma_1 \cap \cdots \cap \sigma_n \ (n \geq 2)$.

We write $\sigma \in \tau$, if σ is an occurrence of a subtype of τ, where the notion of occurrence is the usual one, cf. [23, Definition 9A2]. For instance, type $\alpha = (a \to a) \to a \to a$ has three different subtypes α, $a \to a$ and a, which have respectively 1, 2 and 4 occurrences in α. We assume that the intersection operator is associative, and that it binds stronger than $\to$. Furthermore, we always identify maximal subtypes containing intersections, that is, when we write $\tau_1 \cap \cdots \cap \tau_n$, then none of the τ_i is itself an intersection. The notion of rank stems from [28] and is defined by $\mathsf{rank}(\tau) = 0$ if $\tau \in \mathcal{T}$, by $\mathsf{rank}(\sigma \to \tau) = max(1 + \mathsf{rank}(\sigma), \mathsf{rank}(\tau))$ if $\mathsf{rank}(\sigma) + \mathsf{rank}(\tau) > 0$, and by $\mathsf{rank}(\tau_1 \cap \cdots \cap \tau_n) = max(1, \mathsf{rank}(\tau_1), \ldots, \mathsf{rank}(\tau_n))$ for $n \geq 2$. Considering the syntactic tree of a type θ, its rank equals the maximal number of implications, to which an intersection occurs to the left, plus one. In general, given a particular occurrence of a subtype $\tau \in \theta$ we denote by $\mathsf{depth}(\tau, \theta)$ the number of implications in θ to which τ occurs to the left and call it the depth of τ in θ. Then, we have that $\mathsf{rank}(\theta) \geq r \geq 1$ if and only if there exists an intersection at depth $\geq r - 1$. Formally, depth can be defined (top-down) by $\mathsf{depth}(\theta, \theta) = 0$, by $\mathsf{depth}(\sigma, \theta) = \mathsf{depth}(\sigma \to \tau, \theta) + 1$ and $\mathsf{depth}(\tau, \theta) = \mathsf{depth}(\sigma \to \tau, \theta)$, and by $\mathsf{depth}(\tau_i, \theta) = \mathsf{depth}(\tau_1 \cap \cdots \cap \tau_n, \theta)$ for $1 \leq i \leq n$ and $n \geq 2$. Another meaningful notion is the degree of an occurrence $\tau \in \theta$, which measures the number of consecutive implications, ignoring intersections in that counting, to which τ occurs to the right. Formally, we have $\mathsf{deg}(\theta, \theta) = 0$, if $\mathsf{deg}(\sigma \to \tau, \theta) = n$ then $\mathsf{deg}(\sigma, \theta) = 0$ and $\mathsf{deg}(\tau, \theta) = n + 1$, and finally $\mathsf{deg}(\tau_i, \theta) = \mathsf{deg}(\tau_1 \cap \cdots \cap \tau_n, \theta)$ for $1 \leq i \leq n$ and $n \geq 2$. In particular, θ is a strict intersection type [44] if and only if all intersections in θ occur at degree 0.

The notion of polarity of an occurrence $\tau \in \theta$ is defined as usual by: τ occurs positively in τ; if τ occurs positively (resp. negatively) in θ then it occurs positively (resp. negatively) in $\sigma \to \theta$ and in $\tau_1 \cap \cdots \cap \theta \cap \cdots \cap \tau_n$, and negatively (resp. positively) in $\theta \to \sigma$. As an alternative, one can define that τ occurs positively in θ if $\mathsf{depth}(\tau, \theta)$ is even, and negatively otherwise.

$$\frac{\Gamma \cup \{x : \sigma\} \vdash_\cap N : \tau}{\Gamma \vdash_\cap \lambda x.N : \sigma \to \tau}\ (\mathrm{I}{\to}) \qquad \frac{\Gamma \vdash_\cap x N_1 \cdots N_s : \sigma \to \tau \quad \Gamma \vdash_\cap N_{s+1} : \sigma}{\Gamma \vdash_\cap x N_1 \cdots N_s N_{s+1} : \tau}\ (\mathrm{E}{\to})$$

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash_\cap x : \tau}\ (\mathsf{var}) \qquad \frac{\Gamma \vdash_\cap M : \tau_1 \ \cdots \ \Gamma \vdash_\cap M : \tau_n}{\Gamma \vdash_\cap M : \tau_1 \cap \cdots \cap \tau_n}\ (\mathrm{I}\cap) \qquad \frac{\Gamma \vdash_\cap M : \tau_1 \cap \cdots \cap \tau_n}{\Gamma \vdash_\cap M : \tau_i \ \ (1 \leq i \leq n)}\ (\mathrm{E}\cap)$$

Figure 1 Intersection Type Assignment for Terms in Normal Form.

Every type τ can be uniquely written as $\tau = \tau_1 \to \cdots \to \tau_n \to \theta$ $(n \geq 0)$, where θ is a type variable or an intersection $\theta_1 \cap \cdots \cap \theta_m$ $(m \geq 2)$. If $n \geq 1$, then $\tau_1, \ldots, \tau_n$ are called the *arguments* of τ. An occurrence of σ in τ is called a *negative subpremise* of τ iff it is the argument of a positive occurrence of a subtype in τ. We write $\sigma \preceq_\cap \beta$ and say that σ is a *component* of β if and only if $\beta = \beta_1 \cap \cdots \cap \beta_n$ and $\sigma = \beta_i$, for some $i \in [1..n]$ and $n \geq 1$. By our convention on intersections this implies that a component is never an intersection itself.

▶ **Example 2.** Let $\alpha = \alpha_1 \cap \alpha_2$, where $\alpha_1 = 1 \to (0 \to 0) \cap (1 \to 1) \to (1 \to 0) \to 0$ and $\alpha_2 = 1 \to (1 \to 0) \to (0 \to 1) \to 0$, and 0 and 1 denote type variables. This type belongs to the family used in the reduction of the halting problem for bus machines to rank 2 inhabitation in [43], showing thereby EXPSPACE-hardness of the problem. Furthermore, consider $\beta = o \cap \beta_1 \to o$, where $\beta_1 = ((o \to (o \to o) \cap (o \to o)) \to o) \to (o \to o) \cap (o \to o)$. We have $\mathsf{rank}(\alpha) = 2$ and $\mathsf{rank}(\beta) = 4$. Moreover, α is strict, while β is not, since both intersections in β_1 occur at degree 1 in β. Types α and β will be our running examples throughout the paper.

We denote λ-terms by $M, N, \ldots$, which are built from an infinite countable set of term variables $\mathcal{V}$. We identify terms modulo α-equivalence. For type assignment we consider a system equivalent to the one presented in [27, 43], but restricted to β-normal forms, which is the usual choice when addressing inhabitation related problems. Unless stated otherwise, all λ-terms considered in the remainder of this paper are supposed to be in β-normal form.

A (consistent) *context* is a finite set Γ of *declarations* of the form $x : \sigma$, where $x \in \mathcal{V}$ and $\sigma \in \mathcal{T}_\cap$, such that all term variables occurring in Γ are distinct from each other. The *domain* of Γ, denoted by $\mathsf{dom}(\Gamma)$, is the set of term variables occurring in Γ. For $(x : \sigma) \in \Gamma$, let $\Gamma(x) = \sigma$. Furthermore, $\mathsf{Types}(\Gamma) = \{ \sigma \mid x : \sigma \in \Gamma \}$. The rules of the type assignment system are given in Figure 1. Formulas (judgements) in derivations are of the form $\Gamma \vdash_\cap M : \theta$. Symbol $\vdash$ will be used for the inference of formulas in a second equivalent system without explicit $\cap$-elimination, whose rules are given in Figure 2. We say that type $\theta \in \mathcal{T}_\cap$ can be assigned to a normal form M in context Γ, and write $\Gamma \vdash_\cap M : \theta$, if and only if this formula can be obtained by applying the rules in Figure 1 a finite number of times. For the parameters in these rules we suppose that $s \geq 0$, $n \geq 2$ and $i \in [1..n]$. Similarly, we write $\Gamma \vdash M : \theta$ if that formula can be derived by the inference rules in Figure 2. We draw attention to the fact that for $\vdash$, dropping the $\cap$-elimination rule was achieved by, in some sense, incorporating quasi-order $\sqsubseteq$ in [35, Lemma 20] directly into rules (var) and ($\mathrm{E}{\to}$), and facilitates handling types with intersections at degree ≥ 1, i.e. non-strict types.

For $\triangleright \in \{\vdash_\cap, \vdash\}$, it is easy to verify that $\Gamma \triangleright M : \theta$ implies that the set of term variables in Γ contains the set of free variables in M, i.e. $\mathsf{dom}(\Gamma) \supseteq \mathsf{FV}(M)$. A derivation of a formula $\Gamma \triangleright M : \theta$ can be represented as a derivation tree Π, in which all nodes are labelled by formulas, such that $\Gamma \triangleright M : \theta$ is the root of Π, every internal node is obtained from its children by one of the type assignment rules different from (var), and every leaf is labelled with an instance of (var).

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \tau_i \quad (\tau_i \preceq_\cap \tau)}\,(\mathsf{var}) \qquad \frac{\Gamma \cup \{x : \sigma\} \vdash N : \tau}{\Gamma \vdash \lambda x.N : \sigma \to \tau}\,(\mathrm{I}{\to})$$

$$\frac{\Gamma \vdash xN_1 \cdots N_s : \sigma \to \tau \quad \Gamma \vdash N_{s+1} : \sigma}{\Gamma \vdash xN_1 \cdots N_s N_{s+1} : \tau_i \quad (\tau_i \preceq_\cap \tau)}\,(\mathrm{E}{\to}) \qquad \frac{\Gamma \vdash M : \tau_1 \ \cdots \ \Gamma \vdash M : \tau_n}{\Gamma \vdash M : \tau_1 \cap \cdots \cap \tau_n}\,(\mathrm{I}\cap)$$

Figure 2 Type Assignment without an explicit ∩-elimination rule.

▶ **Lemma 3.** *We have $\Gamma \vdash_\cap M : \theta$ if and only if there is a derivation of $\Gamma \vdash M : \theta$, such that for every formula $\Gamma' \vdash N : \tau$ in that derivation, either*

- *τ is an intersection and $\Gamma' \vdash N : \tau_1 \cap \cdots \cap \tau_n$ $(n \geq 2)$ was derived from $\Gamma' \vdash N : \tau_1, \ldots, \Gamma' \vdash N : \tau_n$ by one application of rule (I∩);*
- *or $\Gamma' \vdash N : \tau$ was derived using one of the rules* (var)*, (I→), or(E→).*

From now on we will only consider $\vdash$-derivations satisfying the conditions in Lemma 3. The set of β-normal terms M such that $\Gamma \vdash M : \tau$ is denoted by $\mathsf{Nhabs}(\Gamma, \tau)$. If $\Gamma = \emptyset$, then we also write $\vdash M : \tau$ instead of $\Gamma \vdash M : \tau$ and say that M is an *inhabitant* of type τ. The set of all β-normal inhabitants of τ is denoted by $\mathsf{Nhabs}(\tau) = \mathsf{Nhabs}(\emptyset, \tau)$. The inhabitation problem for intersection types is the problem of deciding if, for a given type $\tau \in \mathcal{T}_\cap$ (input) one has $\mathsf{Nhabs}(\tau) \neq \emptyset$, and denoted by INH.

3 Pregrammars for Intersection Types

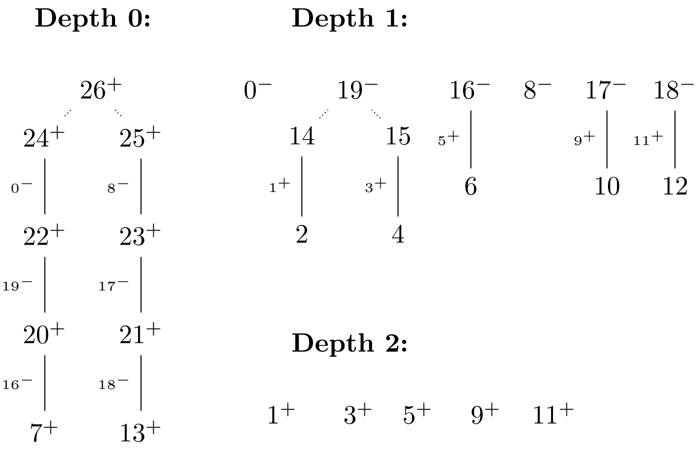
Given $\tau \in \mathcal{T}_\cap$, let $\mathsf{occT}(\tau)$ denote the set of all *occurrences* of subtypes of τ, i.e. $\mathsf{occT}(\tau) = \{ \ \sigma \mid \sigma \in \tau \ \}$. Consider $\mathsf{N}(\tau) = [0..(|\mathsf{occT}(\tau)| - 1)]$ as well as an (arbitrary) bijection $\mathsf{n} : \mathsf{occT}(\tau) \longrightarrow \mathsf{N}(\tau)$. We call $\mathsf{n}(\sigma)$ the identifier of $\sigma \in \tau$. The type of identifier $k \in \mathsf{N}(\tau)$ is $\mathsf{t}(k) = \mathsf{n}^{-1}(k) \in \mathsf{occT}(\tau)$. Relation $\preceq_\cap$ transfers to elements in $\mathsf{N}(\tau)$ in the obvious way, by $n \preceq_\cap m$ iff $\mathsf{t}(n) \preceq_\cap \mathsf{t}(m)$. In order to deal correctly with the correspondence between occurrences of subtypes and occurrences of subterms, polarities have to be taken into account. With this purpose, and whenever convenient, we might superscript an integer n with '+' if n corresponds to a positive occurrence of a subtype and with '−' if it corresponds to a negative subpremise. Integers that correspond to a negative occurrence, which is no subpremise, will not be superscripted. We write $m \equiv_{\mathsf{occT}} n$ if and only if the type occurrences corresponding to m and n, i.e. $\mathsf{t}(m)$ and $\mathsf{t}(n)$, are identical[1], i.e. are (not necessarily different) occurrences of the same subtype. The relation $T(\tau) \subseteq \mathsf{N}(\tau)^3$ is defined by $(n, k, m) \in T(\tau)$, abbreviated by $n \rightarrowtail^k m$, iff $\mathsf{t}(m) = \mathsf{t}(k) \to \mathsf{t}(n)$ for $m, n, k \in \mathsf{N}(\tau)$. We will display relevant information about relations T and $\preceq_\cap$ in the *dependency graph* $\mathcal{G}(\tau)$, whose set of vertices is $\mathsf{N}(\tau)$ and that consists of trees such that:

- If $\mathsf{t}(m) = \mathsf{t}(n_1) \cap \cdots \cap \mathsf{t}(n_k)$, then node m has children $n_1, \ldots, n_k$, connected by dotted edges.
- If $n \rightarrowtail^k m$, then node m has one child n and a firm edge between them labelled with k.

In light of this last observation we define $\mathsf{lab}(n, m) = k$ whenever $n \rightarrowtail^k m$, meaning that the edge between n and m is labelled by k.

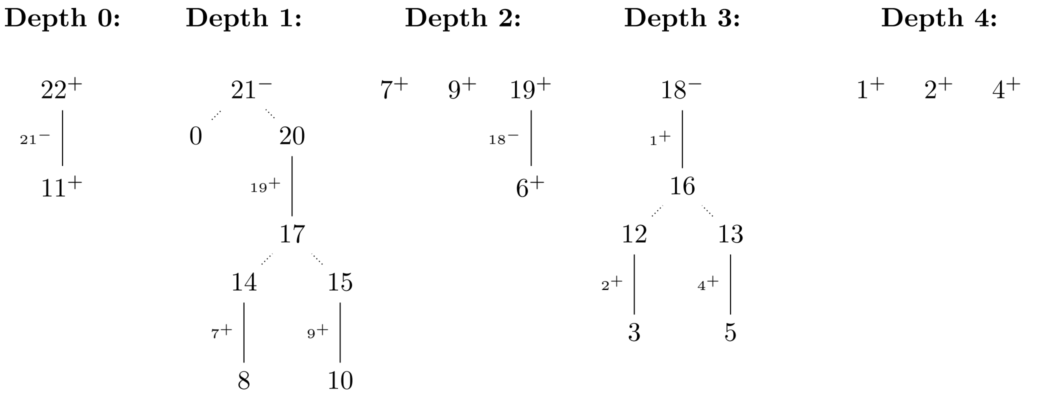
▶ **Example 4.** Consider α from Example 2, let $\mathsf{n}(\alpha) = 26$, $\mathsf{n}(\alpha_1) = 24$, $\mathsf{n}(\alpha_2) = 25$ and the remaining identifiers of subformulas of α_1 and α_2 respectively assigned as follows: $1_0 \to [((0_1 \to 0_2)^{14} \cap (1_3 \to 1_4)^{15})^{19} \to ((1_5 \to 0_6)^{16} \to 0_7)^{20}]^{22}$, and $1_8 \to [(1_9 \to 0_{10})^{17} \to ((0_{11} \to 1_{12})^{18} \to 0_{13})^{21}]^{23}$. The dependency graph of α is the following.

[1] Abusing on the notation, we will occasionally abbreviate this and simply write $\mathsf{t}(m) = \mathsf{t}(n)$.



Degrees of occurrences of subtypes are clearly visible in the dependency graph, since they coincide with the number of firm edges between their identifier and the top node in the subgraph of $\mathcal{G}$ in which they occur. As such, the occurrences corresponding to $22, 23, 2, 4, 6, 10, 12$ have degree 1 (their distance to the top node in the corresponding subgraph is by one firm edge), those corresponding to 20 and 21 are of degree 2 (distance by two firm edges), and those to 7 and 13 are of degree 3. All other occurrences are of degree 0, including the intersections (which correspond respectively to 26 and 19). We conclude that α is strict. On the other hand, its rank is 2 since there is an intersection at depth 1.

For β consider an attribution of identifiers to occurrences of type variable o suggested by $\beta = o_0 \cap \beta_1 \to o_{11}$, where $\beta_1 = ((o_1 \to (o_2 \to o_3) \cap (o_4 \to o_5)) \to o_6) \to (o_7 \to o_8) \cap (o_9 \to o_{10})$. Furthermore, $\mathsf{n}(\beta) = 22$, $\mathsf{n}(o_0 \cap \beta_1) = 21$, and $\mathsf{n}(\beta_1) = 20$. The remaining of this attribution n should be clear from $\mathcal{G}(\beta)$ depicted below. The graph shows clearly that β is not a strict intersection type, since both intersections $\mathsf{t}(17) = \mathsf{t}(14) \cap \mathsf{t}(15)$, as well as $\mathsf{t}(16) = \mathsf{t}(12) \cap \mathsf{t}(13)$ occur at degree 1. Since the depth of $\mathsf{t}(16)$ is 3, we have that β is of rank 4.



Positive and negative identifiers play opposite roles during the process of inhabitant search. A positive identifier m^+ represents a goal of constructing a term of type $\mathsf{t}(m)$, while a negative identifier m^- represents a (possibly available) variable of type $\mathsf{t}(m)$. As such, if one has $\mathsf{t}(m^+) = \mathsf{t}(m_1) \cap \mathsf{t}(m_2)$, then in order to construct a term of type $\mathsf{t}(m)$ one has to find an inhabitant that has simultaneously types $\mathsf{t}(m_1)$ and $\mathsf{t}(m_2)$. On the other hand, $\mathsf{t}(m^-) = \mathsf{t}(m_1) \cap \mathsf{t}(m_2)$ represents an alternative (the variable can be used either with type $\mathsf{t}(m_1)$ or $\mathsf{t}(m_2)$). Similarly, if one has $\mathsf{t}(m^+) = \mathsf{t}(k^-) \to \mathsf{t}(n^+)$, i.e. $n \rightarrowtail^k m$, then one possibility (other than to construct an applicational term using one of the available variables) is to add a variable of type $\mathsf{t}(k^-)$ to the context and search for a term of type $\mathsf{t}(n^+)$. Conversely, an available variable of type $\mathsf{t}(m^-) = \mathsf{t}(k^+) \to \mathsf{t}(n)$, i.e. $n \rightarrowtail^k m^-$, allows to transform the goal of finding a term of type $\mathsf{t}(n)$ into a new goal of finding a term of type $\mathsf{t}(k)$. These concepts are captured in the definition of pregrammars.

▶ **Definition 5** (Pregrammar). *The pregrammar* $\mathsf{pre}(\tau)$ *of* τ *is the smallest set of rules satisfying the following conditions.*

- $m := (n_1, \dots, n_s) \in \mathsf{pre}(\tau)$*, if* $\mathsf{t}(m^+) = \mathsf{t}(n_1) \cap \cdots \cap \mathsf{t}(n_s)$*,* $(s \geq 2)$*.*
- $m := \lambda k.n \in \mathsf{pre}(\tau)$*, if* $n \rightarrowtail^k m^+$*.*
- $m := p_0\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$*, if* $m^+ \equiv_{\mathsf{occT}} c_s$ *for* $s \geq 0$ *and there exists an ascending path from node* c_s *to node* p_0^- *in* $\mathcal{G}(\tau)$*, satisfying*

$$c_s \preceq_\cap p_s \rightarrowtail^{n_s} c_{s-1} \preceq_\cap \dots \preceq_\cap p_1 \rightarrowtail^{n_1} c_0 \preceq_\cap p_0^- .$$

The size of a type τ, denoted by $|\tau|$, is the total number of occurrences of variables and of symbols $\to$ and $\cap$ in τ. Then, the number (of occurrences) of subformulas is $|\mathsf{N}(\tau)| \leq |\tau|$ and consequently the size of $\mathcal{G}(\tau)$, i.e. the total number of nodes and edges, is at most $2|\tau| - 1$. Let $|\tau|_\to$ and $|\tau|_\cap$ denote respectively the number of occurrences of $\to$ and of $\cap$ in τ. Then, the number of rules in $\mathsf{pre}(\tau)$ is at most $|\tau|_\cap + |\tau|_\to + |\tau| \cdot |\tau| < 3|\tau|^2$. Each of the rules contains at most $|\tau|$ identifiers. Hence, the size of $\mathsf{pre}(\tau)$, i.e. the total number of occurrences of identifiers in $\mathsf{pre}(\tau)$, is $|\mathsf{pre}(\tau)| < 3|\tau|^3$.

▶ **Example 6.** Consider α and β from Example 2. Equivalence classes in $\mathsf{N}(\alpha)/\!\equiv_{\mathsf{occT}}$ that are not singletons are $\{0^-, 3^+, 4, 5^+, 8^-, 9^+, 12\}$, $\{1^+, 2, 6, 7^+, 10, 11^+, 13^+\}$, and $\{16^-, 17^-\}$. The pregrammar of α has size $|\mathsf{pre}(\alpha)| = 87$ ($|\alpha| = 27$) and contains thirty-one rules:

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
26 & := & (24, 25) & 20 & := & \lambda 16.7 & 21 & := & \lambda 18.13 \\
24 & := & \lambda 0.22 & 25 & := & \lambda 8.23 & 1, 7, 11, 13 & := & 19\ 1 \mid 16\ 5 \mid 17\ 9 \\
22 & := & \lambda 19.20 & 23 & := & \lambda 17.21 & 3, 5, 9 & := & 19\ 3 \mid 18\ 11 \mid 0 \mid 8
\end{array}$$

Moreover, the equivalence relation $\equiv_{\mathsf{occT}}$ partitions $\mathsf{N}(\beta)$ into eight classes $\{12, 13, 14, 15\}$, $\{0, 1^+, 2^+, 3, 4^+, 5, 6^+, 7^+, 8, 9^+, 10, 11^+\}$, $\{16, 17\}$, $\{18^-\}$, $\{19^+\}$, $\{20\}$, $\{21^-\}$, and $\{22^+\}$. Pregrammar $\mathsf{pre}(\beta)$ contains the following rules.

$$\begin{array}{rcl@{\qquad}rcl}
22 & := & \lambda 21.11 & 1, 2, 4, 6, 7, 9, 11 & := & 21 \mid 21\ 19\ 7 \mid 21\ 19\ 9 \mid 18\ 1\ 2 \mid 18\ 1\ 4 \\
19 & := & \lambda 18.6 & & &
\end{array}$$

For instance, we have rule $6 := 21\ 19\ 9$ in $\mathsf{pre}(\beta)$, because a) $6^+ \equiv_{\mathsf{occT}} 10$, i.e. identifiers 6 and 10 represent the same type, and b) in $\mathcal{G}(\beta)$ there is an ascending path from node (component) 10 to node 21^- whose nodes satisfy $10 \preceq_\cap 10 \rightarrowtail^9 15 \preceq_\cap 17 \rightarrowtail^{19} 20 \preceq_\cap 21^-$, i.e. a variable of type $\mathsf{t}(21)$ applied successively to a term of type $\mathsf{t}(19)$ and to a term of type $\mathsf{t}(9)$ has type $\mathsf{t}(10)$ (which equals $\mathsf{t}(6)$).

4 Type Checking

In the following we describe a rewriting algorithm that, given a type τ and a term M, verifies if $\vdash M : \tau$, i.e. checks if $M \in \mathsf{Nhabs}(\tau)$. During the rewriting process we use objects with the structure of λ-terms, but such that tuples of integers can figure as variable names and are also referred to as *placeholders*. We refer to these objects as *extended terms*. We denote by $N[\vec{k}/x]$ the (extended) term obtained from N by replacing all free occurrences of variable x in N by placeholder $\vec{k}$.

▶ **Definition 7** (Update, Replication). *The* update of $\vec{m} = (m_1, \dots, m_t)$ with $\vec{n} = (n_1, \dots, n_k)$ at position i*, where* $i \in [1..t]$ *and* $t \geq 1$*, is defined by*

$$\vec{m}[\vec{n}/i] = (m_1, \dots, m_{i-1}, n_1, \dots, n_k, m_{i+1}, \dots, m_t).$$

Replicating a value $k \geq 2$ times at position i in $\vec{m}$ *is denoted by* $\vec{m}[i,k]$ *and defined by*

$$\vec{m}[i,k] = (m_1, \dots, m_{i-1}, \underbrace{m_i, \dots, m_i}_{k}, m_{i+1}, \dots, m_t).$$

Given an object E, possibly containing placeholders, let $E[\vec{n}/i]$ (resp. $E[i,k]$) denote the result of applying operation $[\vec{n}/i]$ (resp. $[i,k]$) to all placeholders in E. Given an extended term N, a tuple $\vec{k}$, $i \in \mathbb{N}$ and $x \in \mathcal{V}$, let $N[\vec{k}/x]$ be the result of replacing all free occurrences of x in N by placeholder $\vec{k}$, while $N[\vec{k}/i]$ is obtained by updating all placeholders in N at position i with $\vec{k}$.

▶ **Definition 8** (Type Checking Relation). *Given a type* τ*, an extended term* M*,* $\vec{m} = (m_1, \dots, m_t) \in \mathbb{N}^t$*,* $t \geq 1$*, and* $s \geq 0$*, we write* $(M : \vec{m}) \hookrightarrow (N_1 : \vec{n_1}), \dots, (N_s : \vec{n_s})$*, if one of the following applies.*

- *If* $m_i := \vec{n} \in \mathsf{pre}(\tau)$*, for some* $i \in [1..t]$*,* $\vec{n} = (n_1, \dots, n_k)$*, and* $k \geq 2$*, then* $(M : \vec{m}) \hookrightarrow (M[i,k] : \vec{m}[\vec{n}/i])$.
- *If* $m_i := \lambda k_i . n_i \in \mathsf{pre}(\tau)$*, for all* $i \in [1..t]$*,* $\vec{k} = (k_1, \dots, k_t)$*, and* $\vec{n} = (n_1, \dots, n_t)$*, then* $(\lambda x.N : \vec{m}) \hookrightarrow (N[\vec{k}/x] : \vec{n})$.
- *If* $m_i := k_i\ n_1^i \cdots n_s^i \in \mathsf{pre}(\tau)$*, for all* $i \in [1..t]$*,* $\vec{n_j} = (n_j^1, \dots, n_j^t)$*, for* $j \in [1 \dots s]$*, and* $\vec{k} = (k_1, \dots, k_t)$*, then* $(\vec{k}\ N_1 \cdots N_s : \vec{m}) \hookrightarrow (N_1 : \vec{n_1}), \dots, (N_s : \vec{n_s})$.

The definition of $\hookrightarrow$ *extends, in the usual way, to rewriting of sequences. Then,* $\hookrightarrow^*$ *denotes the reflexive, transitive closure of* $\hookrightarrow$.

▶ **Example 9.** Consider α from Example 2 and $M = \lambda xyz.y(z(yx))$. Then,

$$\begin{aligned}
(M : 26) &\hookrightarrow (\lambda xyz.y(z(yx)) : (24,25)) \hookrightarrow (\lambda yz.y(z(y(0,8))) : (22,23)) \\
&\hookrightarrow (\lambda z.(19,17)(z((19,17)(0,8))) : (20,21)) \hookrightarrow ((19,17)((16,18)((19,17)(0,8))) : (7,13)) \\
&\hookrightarrow ((16,18)((19,17)(0,8)) : (1,9)) \hookrightarrow ((19,17)(0,8) : (5,11)) \hookrightarrow ((0,8) : (3,9)) \hookrightarrow \epsilon.
\end{aligned}$$

The proof of the following theorem, stating correctness of $\hookrightarrow$, is mainly technical and can be found in the appendix.

▶ **Theorem 10.** $\mathsf{Nhabs}(\tau) = \{\, M \mid (M : \mathsf{n}(\tau)) \hookrightarrow^* \epsilon \,\}$.

5 Inhabitation

In this section we define a rewriting relation for type inhabitation.

▶ **Definition 11** (Inhabitation Relation). *Let* $\tau \in \mathcal{T}_\cap$*,* $\vec{m} = (m_1, \dots, m_t) \in \mathbb{N}^t$*,* $t \geq 1$*,* $\vec{V} \subseteq \mathbb{N}^t$*, and* $s \geq 0$*. We write*

$$(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V'} : \vec{n_1}), \dots, (\vec{V'} : \vec{n_s}),$$

if one of the following applies.

1. *If* $m_i := \vec{n} \in \mathsf{pre}(\tau)$*, for some* $i \in [1..t]$*, where* $\vec{n} = (n_1, \dots, n_k)$ *and* $k \geq 2$*, then*

$$(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V}[i,k] : \vec{m}[\vec{n}/i]).$$

2. *If* $m_i := \lambda k_i.n_i \in \mathsf{pre}(\tau)$*, for all* $i \in [1..t]$*, where* $\vec{k} = (k_1, \dots, k_t)$ *and* $\vec{n} = (n_1, \dots, n_t)$*, then*

$$(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} \cup \{\vec{k}\} : \vec{n}).$$

3. *If* $m_i := k_i\ n_1^i \cdots n_s^i \in \mathsf{pre}(\tau)$*, for all* $i \in [1..t]$*, and* $\vec{k} = (k_1, \dots, k_t) \in \vec{V}$*, where* $\vec{n_j} = (n_j^1, \dots, n_j^t)$*,* $(1 \le j \le s)$*, then*

$$(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} : \vec{n_1}), \dots, (\vec{V} : \vec{n_s}).$$

The definition of $\rightsquigarrow$ *extends, in the usual way, to rewriting of sequences. Then,* $\rightsquigarrow^*$ *denotes the reflexive, transitive closure of* $\rightsquigarrow$*.*

▶ **Example 12.** For α from Example 2 we have the following, where $\vec{V} = \{(0,8),(19,17),(16,18)\}$.

$$\begin{aligned}(\emptyset : 26) &\rightsquigarrow (\emptyset : (24,25)) \rightsquigarrow (\{(0,8)\} : (22,23)) \rightsquigarrow (\{(0,8),(19,17)\} : (20,21)) \\ &\rightsquigarrow (\vec{V} : (7,13)) \rightsquigarrow (\vec{V} : (1,9)) \rightsquigarrow (\vec{V} : (5,11)) \rightsquigarrow (\vec{V} : (3,9)) \rightsquigarrow \epsilon,\end{aligned}$$

The following theorem, stating correctness of $\rightsquigarrow$, is proved in the appendix.

▶ **Theorem 13.** $\mathsf{Nhabs}(\tau) \neq \emptyset$ *if and only if* $(\emptyset : \mathsf{n}(\tau)) \rightsquigarrow^* \epsilon$*.*

5.1 Intersections at different depth or degree

We want to examine more closely how intersections at different depth or degree in a type contribute differently to the complexity of type checking and inhabitation. To that end we analyse, where in a grammar rule, their identifiers can occur and what consequences that might have in terms of applications of $\hookrightarrow$ and $\rightsquigarrow$. Let $\mathsf{r}(i)$ stand for an arbitrary identifier of a subtype at depth i. Then each rule in a pregrammar respects one of the following patterns, where $i, j \ge 0$.

(1) $\mathsf{r}(i) := (\mathsf{r}(i), \dots, \mathsf{r}(i))$ (2) $\mathsf{r}(i) := \lambda \mathsf{r}(i+1).\mathsf{r}(i)$ (3) $\mathsf{r}(i) := \mathsf{r}(j)\ \mathsf{r}(j+1) \cdots\ \mathsf{r}(j+1)$

We observe the following:

- if m is the identifier of a positive intersection, then there is exactly one rule for m in $\mathsf{pre}(\tau)$ and that rule respects pattern (1); consequently rule 1 is incompatible with the other two rules in Definition 11;
- if m occurs at depth 0, then M cannot occur on the right side of a rule of pattern (3) in $\mathsf{pre}(\tau)$; consequently a pair $(\vec{V}, \vec{m})$, such that m is one of the coordinates of vector $\vec{m}$ (possibly $\vec{m}$ itself), occurs at most once in a rewriting sequence starting with $(\emptyset : \mathsf{n}(\tau))$; in particular intersections at depth 0 may contribute to the growth of tuples at most once;
- there can be two different rules $m := k\ n_1 \cdots n_s$ and $m' := k\ n'_1 \cdots n'_{s'}$ in $\mathsf{pre}(\tau)$ such that $m = m'$ or $(m \neq m' \wedge s = s')$, only if k is the identifier of a negative subtype and there is at least one (at any degree) intersection in the tree in $\mathcal{G}(\tau)$ that is rooted in k; consequently type checking is deterministic if there are no negative intersections in τ; in terms of inhabitation negative intersections may increase the combinatorics of the problem, but don't seem to cause undecidability on their own[2];

[2] This is in accordance with the fact that there is no gap between ranks 3 and 4, which differ by negative intersections at depth 3 only. Both have undecidable inhabitation problems, shown by Urzyczyn in 1999 for rank 3 and in 2009 for rank 4 [42, 43].

- in principle, there is no relation between depths i and j in a rule of pattern (3); consequently it seems as if negative intersections at different depth contribute to the same extent to the complexity of inhabitation;
- the degree at which an intersection occurs seems to have no particular influence on the problem and similar results should be expected regardless of considering strict intersection types or not.

6 Algorithm $\mathcal{I}$

A Wajsberg/Ben-Yelles style alternating semi-decision algorithm $\mathcal{I}$ for inhabitation of intersection types, following [43], can be implemented based on relation $\rightsquigarrow$. Every rewriting sequence starting in $(\emptyset : \mathsf{n}(\tau))$ corresponds to a unique computation tree Π, whose nodes are labelled with occurrences of pairs $(\vec{V} : \vec{m})$ in the rewriting sequence. The root of Π is $(\emptyset : \mathsf{n}(\tau))$ and each node $(\vec{V} : \vec{m})$, that was rewritten by $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V'} : \vec{n_1}), \ldots, (\vec{V'} : \vec{n_s})$, has s children, respectively labelled with $(\vec{V'} : \vec{n_1}), \ldots, (\vec{V'} : \vec{n_s})$. The rewriting sequence terminates with the empty sequence, i.e. $(\emptyset : \mathsf{n}(\tau)) \rightsquigarrow^* \epsilon$, if and only if in Π all leafs are labelled with $(\vec{V} : \vec{m})$ such that $(\vec{V} : \vec{m}) \rightsquigarrow \epsilon$. In this case the computation tree is called an *accepting computation tree for τ*.

▶ **Definition 14** (Algorithm $\mathcal{I}$). *Algorithm $\mathcal{I}$, starting with $(\emptyset : \mathsf{n}(\tau))$, aims to construct an accepting computation tree for τ, operating in each step on a pair $(\vec{V} : \vec{m})$. The procedure non-deterministically chooses a combination of rules in $\mathsf{pre}(\tau)$, such that $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V'} : \vec{n_1}), \ldots, (\vec{V'} : \vec{n_s})$. If $s = 0$, then the algorithm accepts, otherwise it universally applies to $(\vec{V'} : \vec{n_1}), \ldots, (\vec{V'} : \vec{n_s})$. It rejects, if there is no such combination of rules in $\mathsf{pre}(\tau)$.*

Note that rules 1, 2, and 3 in Definition 11 correspond respectively to rules 1, 2, and 3 in the definition of Urzyczyn's algorithm in [43]. While rule 1 is not compatible with the other two, rules 2 and 3 may both apply at some point. If one gives priority to rule 2 whenever possible, then the algorithm is customised to find only long solutions/inhabitants in the sense of Definition 8 in [27]. We denote this variant by $\mathcal{I}^{\mathsf{long}}$.

In order to guarantee termination, i.e. convert $\mathcal{I}$ into a decision algorithm, we need to restrict the search space, for instance by limiting the maximal height of computation trees. A tree is called *non-repeating* if it has no branch containing two different nodes with the same label. Clearly, there is an accepting computation tree with root $(\emptyset : \mathsf{n}(\tau))$ iff there is a non-repeating accepting computation tree with root $(\emptyset : \mathsf{n}(\tau))$. This fact can be used to establish upper bounds for the complexity of inhabitation for different subfamilies of intersection types. Whenever there is a function $\mathsf{D} : \mathbb{N} \longrightarrow \mathbb{N}$ such that any branch in a computation tree of length $\geq \mathsf{D}(n)$ has a repetition, where $n = |\tau|$ is the size of the input, then $\mathcal{I}$ can be transformed into a decision algorithm $\mathcal{I}_{\mathsf{D}}$, which rejects when operating on nodes of depth $\geq \mathsf{D}(|\tau|)$. In that case we have an upper bound (alternating $O(\mathsf{D}(n))$-time) for deciding the inhabitation problem. This is the usual approach for obtaining upper bounds, cf. [6, 16, 27, 43].

The existence of a measure D as above depends on the maximal number of distinct nodes in a path. Such a value does not exist in general for types of rank ≥ 3 because of the possibility of having continuously growing dimension of nodes in a path. Here the *dimension* of a node labelled with $(\vec{V} : \vec{m})$ is a synonym for the arity of $\vec{m}$ (or of tuples in $(\vec{V} : \vec{m})$) and denoted by $\mathsf{dim}(\vec{V} : \vec{m}) = \mathsf{dim}(\vec{V})$. The dimension of a tree, $\mathsf{dim}(\Pi)$, is the maximal dimension of its nodes. We write $(\vec{V} : \vec{m}) \ll (\vec{V'} : \vec{n})$ if $(\vec{V'} : \vec{n})$ is a descendant of $(\vec{V} : \vec{m})$ in Π. In that case, it follows directly from Definition 11, that either $\mathsf{dim}(\vec{V} : \vec{m}) < \mathsf{dim}(\vec{V'} : \vec{n})$ or $\vec{V} \subseteq \vec{V'}$.

If $\mathsf{dim}(\vec{V} : \vec{m}) = p$ then $\vec{V} \cup \{\vec{m}\} \subseteq \mathsf{N}(\tau)^p$ and the number of tuples of arity $\leq p$ is less than $(|\mathsf{N}(\tau)| + 1)^p$. Since $|\mathsf{N}(\tau)| = |\tau|$, we conclude that the length of a non-repeating branch with a leaf of dimension $p \geq 1$ is $< (|\tau| + 1)^{2p}$. In light of the above we consider the problem of deciding, if for a given type $\tau \in \mathcal{T}_\cap$ (input) there exists an accepting computation tree of dimension $\leq p$, and denote this decision problem by INH_p.

▶ **Proposition 15.** *For each $p \geq 1$ the problem INH_p is* PSPACE*-complete.*

Proof. PSPACE-hardness is a consequence of that result for simple types, whose computation trees have dimension 1. For the upper bound one may consider decision procedure $\mathcal{I}_{\mathsf{D}}$, where $\mathsf{D}(n) = (n+1)^{2p}$. ◀

As foreseeable, the dimension of a computation tree relates to the notion of *dimension*, considered in [16, 18] for strict types. More precisely, to the multiset setting for which we have the following result. The proof of this result relies heavily on notions defined in [16] and is therefore included in the appendix.

▶ **Proposition 16.** *Let τ be a strict intersection type. Then, there exists M such that M can be typed with τ at bounded multiset dimension $\leq p$ if and only if $\tau \in \mathsf{INH}_p$.*

On account of this result, Proposition 15 is in fact a reformulation of Proposition 32 in [16], here extended to the entire set of intersection types.

We want to identify causes for the existing gap between the complexity of the inhabitation problem for different families of intersection types in terms of structural properties of their dependency graphs. The first observation is that the growing of dimension along a path is exclusively due to positive subtypes of the form $\tau_1 \cap \cdots \cap \tau_k$ with $k \geq 2$. Let $\mathcal{T}_\cap^-$ denote the set of intersection types without positive occurrences of intersections, i.e. all intersections occur at an odd depth. Computation trees for types in $\mathcal{T}_\cap^-$ have always dimension 1. Since this fragment falls within the scope of Proposition 15 for $p = 1$, and since $\mathcal{T} \subsetneq \mathcal{T}_\cap^-$, we obtain PSPACE-completeness for $\mathcal{T}_\cap^-$, which is a generalisation of the same result for simple types [39, 41] and includes $\mathcal{T}_2^-$-types considered in [3], i.e. intersection types where all intersections occur at depth 1. More generally we have the following.

▶ **Corollary 17.** *For each $p \geq 1$ the inhabitation problem for types of the form $\tau = \cap_{i=1}^p \tau_i$, where $\tau_i \in \mathcal{T}_\cap^-$ for $i \in [1..p]$, is* PSPACE*-complete.*

Proof. PSPACE-hardness follows from the same result for $\mathcal{T} \subsetneq \mathcal{T}_\cap^-$. Just consider for $\sigma \in \mathcal{T}$ the intersection type $\cap_{i=1}^p \sigma_i$, where each σ_i is a fresh copy of σ (obtained for instance by indexing all variables in σ with i). On the other hand, for $\tau = \cap_{i=1}^p \tau_i$, we have $\mathsf{n}(\tau) := (\mathsf{n}(\tau_1), \ldots, \mathsf{n}(\tau_p))$, but no other rule of that form in $\mathsf{pre}(\tau)$. So rule 1 will be applied exactly once, producing pair $(\emptyset : (\mathsf{n}(\tau_1), \ldots, \mathsf{n}(\tau_p)))$. Hence, any accepting computation tree is of dimension p and it suffices to apply decision procedure $\mathcal{I}_{\mathsf{D}}$, where $\mathsf{D}(n) = n^{2p}$. ◀

Urzyczyn's proof for rank 2 inhabitation in [43] also shows EXPSPACE-completeness for finite intersections of $\mathcal{T}_\cap^-$-types. We denote the rank independent set of these types by

$$\bigcap \mathcal{T}_\cap^- = \{ \cap_{i=1}^p \tau_i \mid p \geq 1 \ \wedge \ \forall i \in [1..p] \ \tau_i \in \mathcal{T}_\cap^- \}.$$

This set properly contains the set of strict rank 2 types.

▶ **Proposition 18.** *The inhabitation problem for $\bigcap \mathcal{T}_\cap^-$ is* EXPSPACE*-complete.*

Proof (from Urzyczyn [43]). The set of strict rank 2 types is a proper subset of $\bigcap \mathcal{T}_{\cap}^{-}$ and EXPSPACE-hardness for rank 2 was shown by reduction from the halting problem for bus machines to strict rank 2 types. For the upper bound one may consider decision procedure $\mathcal{I}_{\mathsf{D}}$, where $\mathsf{D}(n) = n + n^n \cdot n^n$. This stems from the fact that for a $\bigcap \mathcal{T}_{\cap}^{-}$-type τ algorithm $\mathcal{I}$ starting with $(\emptyset : \mathsf{n}(\tau))$ will initially apply rule 1 at most n times, expanding the dimension of nodes up to at most n, where $n = |\tau|$. After that phase rule 1 will no longer apply. Given two nodes of dimension n such that $(\vec{V} : \vec{m}) \ll (\vec{V}' : \vec{m}')$ we have necessarily $\vec{V} \subseteq \vec{V}'$. Thus, there are at most n^n different sets of dimension n in a path of a computation tree, as well as at most n^n different tuples $\vec{m}$ of dimension n. Finally, $n + n^n \cdot n^n = n + 2^{2n \log n} \leq n + 2^{n^2}$, for $n \geq 4$. ◀

In [16, Proposition 24] a not rank-bounded family of types, ranged over by T and U, where

$$T ::= a \mid U \to T \qquad \text{and} \qquad U ::= A \mid (\cap_{i=1}^{n} T_i) \to U$$

was considered. More precisely it was shown that every normal inhabitant of an intersection of the form $\cap_{i=1}^{n} U_i$ can be typed at multiset dimension n. Note that the set of types ranged over by U is a proper subset of $\mathcal{T}_{\cap}^{-}$. On the other hand, the set of types of the form $\cap_{i=1}^{n} U_i$ properly includes the family of types used to show EXPSPACE-hardness for rank 2 [43]. Consequently, the proof of Proposition 18 still works and we have EXPSPACE-hardness also for this proper subclass. We apply this line of reasoning to another rank independent set of types, which is a superset of $\bigcap \mathcal{T}_{\cap}^{-}$ and contains types with positive intersections at depth ≥ 2.

▶ **Definition 19.** *For $m, n \in \mathsf{N}(\tau)$ we define $m \succ n$ iff one of the following holds:*

- $m := (n_1, \ldots, n_s) \in \mathsf{pre}(\tau)$ *and* $n = n_i$ *for some* $i \in [1..s]$*;*
- $m := \lambda k.n \in \mathsf{pre}(\tau)$*;*
- $m := k\ n_1 \cdots n_s \in \mathsf{pre}(\tau)$ *and* $n = n_i$ *for some* $i \in [1..s]$.

Then, $\succ^+$ denotes the transitive closure of $\succ$. A type $\tau \in \mathcal{T}_{\cap}$ is called growth restrained *if there is no identifier m with $m := (n_1, \ldots, n_s) \in \mathsf{pre}(\tau)$ and such that $m \succ^+ m$.*

▶ **Proposition 20.** *Inhabitation for growth restrained types is* EXPSPACE*-complete.*

Proof. If τ has rank 2, then $m := (n_1, \ldots, n_s) \in \mathsf{pre}(\tau)$ implies that $\mathsf{t}(m)$ occurs at depth 0 in τ and consequently m appears nowhere else in $\mathsf{pre}(\tau)$. Hence, τ is growth restrained and EXPSPACE-hardness follows from that result for rank 2. If τ is growth restrained then any rule $m := (n_1, \ldots, n_s) \in \mathsf{pre}(\tau)$ applies at most once in a path of a computation tree for τ, whose dimension can for that reason not exceed n, where $n = |\tau|$. Again, there are at most $(n+1)^n$ tuples of dimension $\leq n$. We conclude that the length of a non-repeating branch with a leaf of dimension $\leq n$ is $< (n+1)^{2n}$ and consider decision procedure $\mathcal{I}_{\mathsf{D}}$, where $\mathsf{D}(n) = (n+1)^{2n}$. But $(n+1)^{2n} = 2^{2n \log(n+1)} < 2^{n^2}$, for $n \geq 6$. ◀

The set of growth restrained types contains properly the set of types in which positive occurrences of intersections are only allowed at depth 0. This set, on the other hand contains properly the set of rank 2 types, as well as $\bigcap \mathcal{T}_{\cap}^{-}$. Since all the aforementioned type classes contain the set of strict rank 2 types, for which Urzyczyn showed EXPSPACE-hardness, one obtains as a corollary EXPSPACE-completeness for all these classes.

Wajsberg/Ben-Yelles style search algorithms are the established vehicle to implement counting algorithms [6, 23, 24, 8]. We follow that direction and use algorithm $\mathcal{I}$ to show that the problem of counting for growth restrained types is EXPSPACE-complete. Counting means to decide, if for a given type τ, the set of normal inhabitants $\mathsf{Nhabs}(\tau)$ is empty, finite or infinite. Following the usual approach we provide limits $\mathsf{d}(|\tau|)$ and $\mathsf{D}(|\tau|)$, such that: (i) $|\mathsf{Nhabs}(\tau)| \neq 0$ iff there is an accepting computation tree of height lower than $\mathsf{D}(|\tau|)$; (ii) $|\mathsf{Nhabs}(\tau)| = \infty$ iff there is an accepting computation tree of height between $\mathsf{d}(|\tau|)$ and $\mathsf{D}(|\tau|)$. Condition (i) is decided by $\mathcal{I}_{\mathsf{D}(|\tau|)}$. For (ii) we use a customised version of $\mathcal{I}_{\mathsf{D}(|\tau|)}$, that in each step remembers the highest depth the algorithm operated on so far and accepts the whole computation only if that value is $\geq \mathsf{d}(|\tau|)$.

▶ **Proposition 21.** *Counting for growth restrained types is* EXPSPACE*-complete.*

Proof. EXPSPACE-hardness follows from Proposition 20. To show that the problem can be solved in exponential space we consider $\mathsf{d}(n) = (n+1)^n$ and $\mathsf{D}(n) = (n+1)^{3n}$. Let τ be a growth restrained type with $|\tau| = n$. The proof of Proposition 20 shows that $\mathsf{Nhabs}(\tau) \neq \emptyset$ iff there is an accepting computation tree for τ of height $\leq (n+1)^{2n} < \mathsf{D}(n)$. It remains to show that $|\mathsf{Nhabs}(\tau)| = \infty$ iff there is an accepting computation tree of height between $\mathsf{d}(n)$ and $\mathsf{D}(n)$. Consider an accepting computation tree Π for τ, which we know to have dimension $\leq n$. Given two nodes $\iota \ll \iota'$, labelled respectively with $(\vec{V} : \vec{m})$ and $(\vec{V}' : \vec{n})$, there is a possibly empty sequence $\zeta_{\iota,\iota'}$ of replications that have been applied on the path from ι to ι' (corresponding to applications of rule 1, Definition 11). Let $\vec{V}\zeta_{\iota,\iota'}$ denote the result of applying the replications in $\zeta_{\iota,\iota'}$ successively to $\vec{V}$. Then, $\vec{V}\zeta_{\iota,\iota'} \subseteq \vec{V}'$. Now, suppose that Π has a branch of length $\geq \mathsf{d}(n) = (n+1)^n$, which is the maximal number of tuples of dimension $\leq n$. Thus, there must be two distinct nodes (of equal dimension) such that $(\vec{V_1} : \vec{m}) \ll (\vec{V_2} : \vec{m})$ and $\vec{V_1} \subseteq \vec{V_2}$. Let Π_1 be the subtree of Π rooted in node ι labelled with $(\vec{V_1} : \vec{m})$. For each other node ι' in Π_1 consider the corresponding sequence $\zeta_{\iota,\iota'}$ of replications from ι to ι'. We denote by $\Pi_1\zeta$ the tree obtained from Π_1 by replacing the label of each node $\iota' = (\vec{V}' : \vec{n})$ by $(\vec{V}' \cup \vec{V_2}\,\zeta_{\iota,\iota'} : \vec{n})$. An accepting tree of bigger height can now be obtained replacing in Π the subtree rooted in $(\vec{V_2} : \vec{m})$ by $\Pi_1\zeta$. $|\mathsf{Nhabs}(\tau)| = \infty$ follows by repetition. This part of the proof corresponds to the Stretching Lemma in [23]. It remains to show that the existence of a tree Π of height $\geq \mathsf{D}(n) = (n+1)^{3n}$ implies that there is a tree of height in $[\mathsf{d}(n), \mathsf{D}(n)[$. This last part corresponds to the Shrinking Lemma [23] and is achieved by successively shortening subtrees in Π by a controlled amount, such that the final result is an accepting tree of height $< \mathsf{D}(n)$, but still $\geq \mathsf{d}(n)$. Consider a branch in Π of length $> \mathsf{D}(n) > \mathsf{d}(n)$. There must be two distinct nodes (a repetition) such that $\iota = (\vec{V} : \vec{m}) \ll (\vec{V} : \vec{m}) = \iota'$ and such that the path from ι to ι' has length $l \leq (n+1)^{2n}$. Now, consider the tree Π' obtained by replacing in Π the subtree rooted in ι by the subtree rooted in ι'. All paths starting in ι are now shortened by length $l \geq (n+1)^{2n}$. However, $(n+1)^{3n} - l \geq (n+1)^n$ and the result is a tree of height $\geq \mathsf{d}(n)$. It remains to repeat this process as long as there are paths of length $> \mathsf{D}(n)$. ◀

Changing the limits in the previous proof to $\mathsf{d}(n) = (n+1)^p$ and $\mathsf{D}(n) = (n+1)^{3p}$ suffices to show a similar result for counting in bounded dimensions.

▶ **Corollary 22.** *For each $p \geq 1$, counting the number of accepting computation trees of dimension $\leq p$ is* PSPACE*-complete.*

Some studies consider fragments where the amount of positive and/or negative occurrences of subtypes is constrained. In [4] it was proved that in the simple type system all inhabitants of a given *negatively non-duplicated* type τ are $\beta\eta$-equivalent. Here, negatively non-duplicated means that every type variable occurs at most once negatively in τ. Bourreaux and Salvati adapted this correspondence to the case of λ-terms containing occurrences of constants in [7]. The following result shows that the same is true for the intersection type system.

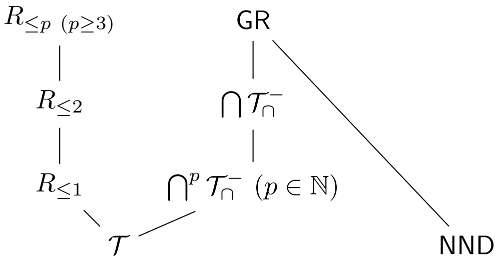

$R_{\leq p}\ (p \geq 3)$
GR
$R_{\leq 2}$
$\bigcap \mathcal{T}_{\cap}^{-}$
$R_{\leq 1}$
$\bigcap^{p} \mathcal{T}_{\cap}^{-}\ (p \in \mathbb{N})$
$\mathcal{T}$
NND

Figure 3 Intersection Type Families.

▶ **Proposition 23.** *Deciding inhabitation for negatively non-duplicated intersection types is in* PSPACE. *Furthermore, if $M, N \in \mathsf{Nhabs}(\tau)$, then $M =_{\beta\eta} N$.*

Proof. We know that τ has a normal inhabitant if and only if it has a long one. Suppose that Π is an accepting non-repeating computation tree for τ obtained by algorithm $\mathcal{I}^{\mathsf{long}}$. We show that Π has height $\leq |\tau|^2$, i.e. Π can be obtained by $\mathcal{I}^{\mathsf{long}}_{\mathsf{D}}$, where $\mathsf{D}(n) = n^2$. If τ is negatively non-duplicating, then for every $m_i^+ \in \mathsf{N}(\tau)$ there is at most one rule of the form $m_i := k_i\ n_1^i \cdots n_s^i \in \mathsf{pre}(\tau)$. Given a pair $(\vec{V} : \vec{m})$, such that neither rule 1 nor rule 2 in Definition 11 apply, there is at most one $\vec{k} = (k_1, \ldots, k_t) \in \vec{V}$ such that $m_i := k_i\ n_1^i \cdots n_s^i \in \mathsf{pre}(\tau)$, for all $i \in [1..|\vec{m}|]$. We conclude that any run of $\mathcal{I}^{\mathsf{long}}$ on $(\emptyset, \mathsf{n}(\tau))$ is deterministic (up to the order in which identifiers in a tuple, to which rule 1 applies, are chosen) and that there is at most one accepting computation tree (up to that order), implying that the number of long inhabitants is finite. Consider any branch $(\vec{V}_1 : \vec{m}_1), \ldots, (\vec{V}_s : \vec{m}_s)$ in Π, where $(\vec{V}_1 : \vec{m}_1) = (\emptyset, \mathsf{n}(\tau))$. We associate a tree π to this branch whose root is labelled by $\mathsf{n}(\tau)$ and such that at depth $i-1$ nodes are labelled by $m_1^i, \ldots, m_{t_i}^i$, where $\vec{m}_i = (m_1^i, \ldots, m_{t_i}^i)$ and have children defined by the following. If $(\vec{V}_{i+1} : \vec{m}_{i+1})$ was obtained by rule 1, because $m_j^i := (n_1, \ldots, n_k) \in \mathsf{pre}(\tau)$, for some $j \in [1..t_i]$, $\vec{n} =$ and $k \geq 2$, then m_j^i has k children labelled respectively with $n_1, \ldots, n_k$. Every other node m_l^i at depth $i-1$ has one child labelled with m_l^i. If $(\vec{V}_{i+1} : \vec{m}_{i+1})$ was obtained by rule 2 or by rule 3, then each node has one child labelled respectively with $k_1, \ldots, k_{t_i}$, for $\vec{k} = (k_1, \ldots, k_{t_i})$ as in Definition 11 (rules 2 and 3). Consider two nodes at different depth in a branch of π, which are labelled with the same identifier m. Since Π is accepting and the rewriting process deterministic (up to order of application of rule 1) all nodes between them are also labelled by m, corresponding to successive applications of rule 1. Otherwise, the algorithm would have entered an infinite loop. But rule 1 can be applied successively at most $|\tau|$ times. On the other hand, there are at most $|\tau|$ different identifiers in $\mathsf{N}(\tau)$. We conclude that $s \leq |\tau|^2$. On the other hand, this means that whenever $(\vec{V}_i : \vec{m}_i) \rightsquigarrow (\vec{V}_i \cup \{\vec{k}\} : \vec{m}_{i+1})$ by rule 2, then $\vec{k} \notin \vec{V}_i$. Consequently, an accepting tree Π corresponds to exactly one long normal inhabitant M and any other normal inhabitant can be obtained from M by η-expansion, cf. proof of Lemma 9 in [27]. ◀

7 Conclusions

We presented a pregrammar based framework for addressing inhabitation related problems in the intersection type system. In that setting types are first represented by dependency graphs, which expose their underlying structure and thereby reveal properties related to (the complexity of) intersection type inhabitation. Rewriting relations corresponding to type checking and inhabitation for normal forms were given. These operate solely on sets of tuples in terms of simple operations, update and replication. After proving the correctness of those relations, which involves some bureaucracy, the method became available to be implemented in a Wajsberg/Ben-Yelles style search algorithm and any forthcoming reasoning could be expressed in terms of rewriting of pairs (consisting of a set of tuples and a tuple). That was illustrated by revisiting and partially extending some well-known problems. An overview of results is given in Table 1. The relation between the different sets of type families, regarding inclusion, is displayed in an Hasse diagram in Figure 3. There $\mathcal{T}$ denotes the set of simple types, $R_{\leq n}$ the set of types of rank at most n, $\bigcap^p \mathcal{T}_\cap^-$ the set of types of the form $\cap_{i=1}^p \tau_i$ for some fixed p, GR the set of groth restrained bypes, and finally NND the set of negatively non-duplicated intersection types. The set of types inhabited in (some) bounded multiset dimension is orthogonal to the remaining families, c.f. [16], and therefore not included in the diagram.

Studying further applications of the method is left for future work. For instance, the amount of complexity caused by negative intersections should be placed under more careful observation. As such, one could wonder about strictly positive fragments.

Table 1 Complexity Results for Inhabitation and Counting.

Problem	**Complexity**
Inhabitation of Simple Types	PSPACE-complete [39, 41]
Counting of Simple Types	PSPACE-complete [24]
Rank 1 Inhabitation	PSPACE-complete [43]
Rank 2 Inhabitation	EXPSPACE-complete [27, 43]
Rank $\geq$ 3 Inhabitation	undecidable [43, 42]
Inhabitation in Bounded Multiset Dimension	EXPSPACE-complete [16]
Inhabitation in Fixed Bounded Multiset Dimension	PSPACE-complete [16]
INH_p for fixed p	PSPACE-complete[3] [Proposition 15]
Counting in Fixed Bounded Multiset Dimension	PSPACE-complete [Corollary 22]
Inhabitation of $\tau = \cap_{i=1}^p \tau_i$ ($\tau_i \in \mathcal{T}_\cap^-$, fixed p)	PSPACE-complete [Corollary 17]
$\bigcap \mathcal{T}_\cap^-$ Inhabitation	EXPSPACE-complete [43, Proposition 18]
Inhabitation for growth restrained types	EXPSPACE-complete [Proposition 20]
Counting for growth restrained types	EXPSPACE-complete [Proposition 21]
Inhabitation for negatively non-duplicated intersection types	PSPACE [Proposition 23]

References

1 S. Alves and S. Broda. Inhabitation machines: determinism and principality. In *Ninth Workshop on Non-Classical Models of Automata and Applications, NCMA 2017*, pages 57–70, 2017.

2 S. Alves and S. Broda. A unifying framework for type inhabitation. In *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, Leibniz International Proceedings in Informatics (LIPIcs), 2018.

[3] The problem is EXPSPACE-complete if p isn't fixed beforehand, corresponding then to inhabitation in bounded multiset dimension.

3 Sandra Alves and Sabine Broda. Pre-grammars and inhabitation for a subset of rank 2 intersection types. *Electr. Notes Theor. Comput. Sci.*, 344:25–45, 2019.

4 T. Aoto. Uniqueness of normal proofs in implicational intuitionistic logic. *Journal of Logic, Language and Information*, 8(2):217–242, 1999.

5 Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.

6 Ch. Ben-Yelles. *Type Assignment in the Lambda-Calculus: Syntax and Semantics*. PhD thesis, University College of Swansea, September 1979.

7 P. Bourreau and S. Salvati. A datalog recognizer for almost affine -cfgs. In *MOL*, pages 21–38, 2011.

8 Sabine Broda and Luís Damas. Counting a type's principal inhabitants. In *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, pages 69–82, 1999.

9 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Inhabitation for non-idempotent intersection types. *Logical Methods in Computer Science*, 14(3), 2018.

10 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017.

11 Martin W. Bunder. The inhabitation problem for intersection types. In *Theory of Computing 2008. Proc. Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), Wollongong, NSW, Australia, January 22-25, 2008. Proceedings*, pages 7–14, 2008.

12 Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. *Arch. Math. Log.*, 19(1):139–156, 1978.

13 Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.

14 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Math. Log. Q.*, 27(2-6):45–58, 1981.

15 Boris Düdder, Moritz Martens, and Jakob Rehof. Staged composition synthesis. In *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 67–86, 2014.

16 Andrej Dudenhefner and Jakob Rehof. Intersection type calculi of bounded dimension. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 653–665, 2017.

17 Andrej Dudenhefner and Jakob Rehof. Typability in bounded dimension. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017.

18 Andrej Dudenhefner and Jakob Rehof. Principality and approximation under dimensional bound. *PACMPL*, 3(POPL):8:1–8:29, 2019.

19 Joshua Dunfield. Elaborating intersection and union types. *J. Funct. Program.*, 24(2-3):133–165, 2014.

20 Joshua Dunfield and Frank Pfenning. Type assignment for intersections and unions in call-by-value languages. In *Foundations of Software Science and Computational Structures, 6th International Conference, FOSSACS 2003 Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, pages 250–266, 2003.

21 Jonathan Frankle, Peter-Michael Osera, David Walker, and Steve Zdancewic. Example-directed synthesis: a type-theoretic interpretation. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 802–815, 2016.

22 Timothy S. Freeman and Frank Pfenning. Refinement types for ML. In *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI), Toronto, Ontario, Canada, June 26-28, 1991*, pages 268–277, 1991.

23 J.R. Hindley. *Basic Simple Type Theory*, volume 42 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1997.

24 S. Hirokawa. Infiniteness of proof (α) is polynomial-space complete. *Theoretical Computer Science*, 206(1-2):331–339, 1998.

25 Jean-Louis Krivine. *Lambda-calculus, types and models*. Ellis Horwood series in computers and their applications. Masson, 1993.

26 Toshihiko Kurata and Masako Takahashi. Decidable properties of intersection type systems. In *Typed Lambda Calculi and Applications, Second International Conference on Typed Lambda Calculi and Applications, TLCA '95, Edinburgh, UK, April 10-12, 1995, Proceedings*, pages 297–311, 1995.

27 D. Kusmierek. The inhabitation problem for rank two intersection types. In *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, pages 240–254, 2007.

28 D. Leivant. Polymorphic type inference. In *Proceedings of Principles of Programming Languages (POPL'83)*, pages 88–98, New York, NY, USA, 1983. ACM Press.

29 Ralph Loader. The undecidability of lambda-definability.

30 Christian Mossin. Exact flow analysis. *Mathematical Structures in Computer Science*, 13(1):125–156, 2003.

31 Jens Palsberg and Christina Pavlopoulou. From polyvariant flow information to intersection and union types. *J. Funct. Program.*, 11(3):263–317, 2001.

32 G. Plotkin. Lambda definability and logical relations, Technical Report Memorandum SAI-RM-4, School of Artificial Intelligence, University of Edingburgh, (1973).

33 G. Pottinger. A type assignment for the strongly normalizable lambda-terms. In J. Hindley and J. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, 1980.

34 Steven J. Ramsay. Exact intersection type abstractions for safety checking of recursion schemes. In *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming, Kent, Canterbury, United Kingdom, September 8-10, 2014*, pages 175–186, 2014.

35 Jakob Rehof and Pawel Urzyczyn. The complexity of inhabitation with explicit intersection. In *Logic and Program Semantics - Essays Dedicated to Dexter Kozen on the Occasion of His 60th Birthday*, pages 256–270, 2012.

36 John C. Reynolds. *Design of the Programming Language FORSYTHE*, page 173–233. Birkhauser Boston Inc., USA, 1997.

37 Sylvain Salvati. Recognizability in the simply typed lambda-calculus. In *Logic, Language, Information and Computation, 16th International Workshop, WoLLIC 2009, Tokyo, Japan, June 21-24, 2009. Proceedings*, pages 48–60, 2009.

38 Sylvain Salvati, Giulio Manzonetto, Mai Gehrke, and Henk Barendregt. Loader and urzyczyn are logically related. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 364–376, 2012.

39 R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9:67–72, 1979.

40 M. Takahashi, Y. Akama, and S. Hirokawa. Normal proofs and their grammar. *Information and Computation*, 125(2):144–153, 1996.

41 P. Urzyczyn. Inhabitation in typed lambda-calculi (a syntactic approach). In *TLCA'97*, volume 1210 of *LNCS*, pages 373–389. Springer, 1997.

42 P. Urzyczyn. The emptiness problem for intersection types. *Journal of Symbolic Logic*, 64(3):1195–1215, 1999.

43 P. Urzyczyn. Inhabitation of low-rank intersection types. In *TLCA'09*, volume 5608 of *LNCS*, pages 356–370. Springer, 2009.

44 Steffen van Bakel. Strict intersection types for the lambda calculus. *ACM Comput. Surv.*, 43(3):20:1–20:49, 2011.

A Appendix

Proof of Lemma 3

Lemma 3. We have $\Gamma \models M : \theta$ if and only if there is a derivation of $\Gamma \vdash M : \theta$, such that for every formula $\Gamma' \vdash N : \tau$ in that derivation, either

- τ is an intersection and $\Gamma' \vdash N : \tau_1 \cap \cdots \cap \tau_n$ $(n \geq 2)$ was derived from $\Gamma' \vdash N : \tau_1, \ldots, \Gamma' \vdash N : \tau_n$ by one application of rule $(I\cap)$;
- or $\Gamma' \vdash N : \tau$ was derived using one of the rules (var), $(\mathrm{I}{\rightarrow})$, or$(\mathrm{E}{\rightarrow})$.

Proof. For the *if*-part of the proof note that the resulting formula of an application of rule (var) (resp. $(\mathrm{E}{\rightarrow})$) in a $\vdash$-derivation can be obtained by one derivation step with rule (var) (resp. $(\mathrm{E}{\rightarrow})$), followed by zero or one application of rule $(\mathrm{E}\cap)$ in a $\models$-derivation. On the other hand, rules $(\mathrm{I}{\rightarrow})$ and $(\mathrm{I}\cap)$ are identical in both systems.

For the *only if*-part of the proof we proceed by induction on the length of the $\models$-derivation. Suppose that $\Gamma' \models N : \tau_1 \cap \cdots \cap \tau_n$ was obtained by rule (var) because $(x : \tau_1 \cap \cdots \cap \tau_n) \in \Gamma'$. If $n = 1$, then $\Gamma' \vdash N : \tau_1$ can also be obtained by rule (var). Otherwise, $\Gamma' \vdash N : \tau_1 \cap \cdots \cap \tau_n$ can be obtained by n applications of rule (var) followed by one application of $(\mathrm{I}\cap)$. Since rules $(\mathrm{I}{\rightarrow})$ and $(\mathrm{I}\cap)$ are identical in both systems, it remains to consider a formula of the form $\Gamma' \models xN_1 \cdots N_s N_{s+1} : \rho_1 \cap \cdots \cap \rho_n$ obtained from $\Gamma' \models xN_1 \cdots N_s : \sigma \rightarrow \rho_1 \cap \cdots \cap \rho_n$ and $\Gamma' \models N_{s+1} : \sigma$ by rule $(\mathrm{E}{\rightarrow})$. The corresponding formula $\Gamma' \vdash xN_1 \cdots N_s N_{s+1} : \rho_1 \cap \cdots \cap \rho_n$ can be obtained by considering n copies of the derivations for $\Gamma' \vdash xN_1 \cdots N_s : \sigma \rightarrow \rho_1 \cap \cdots \cap \rho_n$ and for $\Gamma' \vdash N_{s+1} : \sigma$. To each pair rule $(\mathrm{E}{\rightarrow})$ is applied, leading to derivations of $\Gamma' \vdash xN_1 \cdots N_s N_{s+1} : \rho_1, \ldots, \Gamma' \vdash xN_1 \cdots N_s N_{s+1} : \rho_n$. Finally, $\Gamma' \vdash xN_1 \cdots N_s N_{s+1} : \rho_1 \cap \cdots \cap \rho_n$ follows in one step using $(\mathrm{I}\cap)$ for $n > 1$. ◀

In order to prove correctness of our method we first need to establish the precise correspondence that exists between occurrences of subtypes and of variables and subterms in inhabitants.

Variables, Subterms and Occurrences of Subtypes

Consider a term M, a type τ, and a derivation of $\vdash M : \tau$. In the following we assign occurrences of subtypes of τ to variables and terms in formulas $\Gamma \vdash N : \sigma$ in that derivation. This assignment will be used to establish the correctness of the pregrammars, that are defined in the next section. Every $x \in \mathsf{dom}(\Gamma)$ is assigned a <u>n</u>egative <u>s</u>ub<u>p</u>remise $\mathsf{nsp}(x)$ of τ. Term N is assigned a <u>p</u>ositive <u>s</u>ub<u>t</u>ype $\mathsf{pst}(N)$ of τ. Additionally, if $\Gamma \vdash N : \sigma$ was derived by rule (var) or $(\mathrm{E}{\rightarrow})$, then N is also assigned a <u>n</u>egative <u>s</u>ub<u>t</u>ype $\mathsf{nst}(N)$ of τ. The assignment is such that $\mathsf{pst}(N)$ and $\mathsf{nst}(N)$ are occurrences of σ and for $x \in \mathsf{dom}(\Gamma)$ we have $\mathsf{nsp}(x) = \Gamma(x)$.

▶ **Definition 24** ($\mathsf{pst}, \mathsf{nst}, \mathsf{nsp}$). *Consider a term M, a type $\tau \in \mathcal{T}_\cap$, and a derivation of $\vdash M : \tau$ with derivation tree Π. The assignment of nsp, nst and pst to occurrences of variables and terms in the formulas, that appear in Π, is bottom-up, starting with $\vdash M : \tau$.*

- *For the root $\vdash M : \tau$ of Π, let $\mathsf{pst}(M) = \tau$*[3]*.*
- *Consider $\Gamma \vdash x : \tau_i$ obtained by (var), where $\tau_1 \cap \cdots \cap \tau_n = \Gamma(x)$, $n \geq 1$ and $i \in [1..n]$, with $\mathsf{nsp}(x) = \tau_1 \cap \cdots \cap \tau_n$ and $\mathsf{pst}(x) = \tau_i$. We assign $\mathsf{nst}(x)$ to x on the right side of $\vdash$, choosing the negative occurrence of τ_i in $\mathsf{nsp}(x)$.*

[3] M is necessarily of the form $\lambda x.N$, thus $\mathsf{nst}(M)$ is not defined.

- *Next, consider a formula $\Gamma \vdash xN_1 \cdots N_k : \tau_i$ derived by ($I\!\to$), with $k > 0$. Let Π' be the subtree of Π that derives that formula, $\mathsf{pst}(xN_1 \cdots N_k) = \tau_i$ and suppose that $\mathsf{nsp}(y)$ is defined for all $y \in \mathsf{dom}(\Gamma)$. Consider the declaration $x : \rho_0 \in \Gamma$. Then, there are subtypes $\sigma_1, \ldots, \sigma_k, \rho_1, \ldots, \rho_k$, such that $\sigma_j \to \rho_j \preceq_\cap \rho_{j-1}$ for $j \in [1..k]$ and $\tau_i \preceq_\cap \rho_k$. Furthermore, formula $\Gamma \vdash x : \sigma_1 \to \rho_1$ is first combined with a derived formula $\Gamma \vdash N_1 : \sigma_1$ (by some subtree Π_1 of Π'). The resulting formula $\Gamma \vdash xN_1 : \sigma_2 \to \rho_2$ is then combined with a derived formula $\Gamma \vdash N_2 : \sigma_2$ (by some subtree Π_2 of Π'), etc. Contexts of formulas in Π' always contain Γ and the negative subpremises to variables in Γ in these formulas will be those assigned to variables in the root of Π', which is $\Gamma \vdash xN_1 \cdots N_k : \tau_i$. We now successively assign, operating top down, negative subtypes to terms $x, xN_1, \ldots, xN_1 \cdots N_k$, as well as positive subtypes to terms $N_1, \ldots, N_k$, in the formulas in this part of the tree. For x on the right side of $\Gamma \vdash x : \sigma_1 \to \rho_1$ let $\mathsf{nst}(x)$ be the occurrence of $\sigma_1 \to \rho_1$ in $\mathsf{nsp}(x) = \rho_0$. Now, suppose that formula $\Gamma \vdash xN_1 \cdots N_{j-1} : \sigma_j \to \rho_j$ is combined with $\Gamma \vdash N_j : \sigma_j$, deriving $\Gamma \vdash xN_1 \cdots N_j : \rho'_j$, where $\rho'_j \preceq_\cap \rho_j$ and $j \in [1..k]$. Consider $\mathsf{nst}(xN_1 \cdots N_{j-1}) = \sigma_j \to \rho_j$ (which is already assigned). Then, let $\mathsf{pst}(N_j)$ be the occurrence of σ_j in $\mathsf{nst}(xN_1 \cdots N_{j-1})$ and let $\mathsf{nst}(xN_1 \cdots N_j)$ be the negative occurrence of the component ρ'_j in $\sigma_j \to \rho_j$ (note that $\rho'_j \preceq_\cap \rho_j$).*
- *Consider $\Gamma \vdash \lambda x.N : \sigma \to \rho$ derived from $\Gamma \cup \{x : \sigma\} \vdash N : \rho$ by ($I\!\to$) and $\mathsf{pst}(\lambda x.N) = \sigma \to \rho$. Then, for $\Gamma \cup \{x : \sigma\} \vdash N : \rho$ let $\mathsf{pst}(N)$ be the occurrence of ρ in $\mathsf{pst}(\lambda x.N)$. If $x \in \mathsf{dom}(\Gamma)$, then $\mathsf{nsp}(x)$ is already defined. Otherwise, let $\mathsf{nsp}(x)$ be the negative occurrence of σ in $\mathsf{pst}(\lambda x.N)$.*
- *Now, consider $\Gamma \vdash N : \tau_1 \cap \cdots \cap \tau_n$ derived from $\Gamma \vdash N : \tau_1, \ldots, \Gamma \vdash N : \tau_n$, $(n > 1)$ by rule ($I\cap$). Then, $\mathsf{nsp}(x)$ is already defined for all $x \in \mathsf{dom}(\Gamma)$, as well as $\mathsf{pst}(N) = \tau_1 \cap \cdots \cap \tau_n$. To the occurrence of N in $\Gamma \vdash N : \tau_i$ we assign the positive occurrence of τ_i in $\mathsf{pst}(N)$.*

Besides of the operations on tuples of updating and replication we also need to define an operation of contraction.

▶ **Definition 25** (Contraction)**.** Contracting $k \geq 2$ positions starting at position i in $\vec{m} = (m_1, \ldots, m_t)$ *is denoted by $\vec{m}[i, k \downarrow]$ for $i + k \leq t$ and defined by*

$$\vec{m}[i, k \downarrow] = (m_1, \ldots, m_i, m_{i+k}, \ldots, m_t).$$

Given an object E, possibly containing placeholders, let $E[i, k \downarrow]$ denote the result of applying operation $[i, k \downarrow]$ to all placeholders in E. Then, $E[i, k][i, k \downarrow] = E$.

Proof of Theorem 10

Theorem 10 shows correctness of $\hookrightarrow$, where $\vec{k}|_i$ denotes the projection of the ith coordinate of a tuple $\vec{k}$.

Theorem 10. $\mathsf{Nhabs}(\tau) = \{\, M \mid (M : \mathsf{n}(\tau)) \hookrightarrow^* \epsilon \,\}$.

Proof.
($\subseteq$) Consider a derivation tree of $\vdash M : \tau$ and an occurrence of a formula Ψ of the form $\Gamma \vdash N : \rho$ in derivation tree Π, where $\Gamma = \{x_1 : \sigma_1, \ldots, x_s : \sigma_s\}$. Let $n_i^- = \mathsf{n}(\mathsf{nsp}(x_i)) \in \mathsf{N}(\tau)$ for $i \in [1..s]$, and $m^+ = \mathsf{n}(\mathsf{pst}(N))$, according to Definition 24. Then, $\mathsf{t}(n_i) = \sigma_i$ for $i \in [1..s]$ and $\mathsf{t}(m) = \rho$. We show by induction that $(N[\Gamma] : m) \hookrightarrow^* \epsilon$, using $N[\Gamma]$ as an abbreviation for $N[n_1/x_1, \cdots, n_s/x_s]$.

Suppose that Ψ was obtained by rule (var), i.e. $N = x_i$, $\sigma_i = \sigma_i^1 \cap \cdots \cap \sigma_i^l$ and $\rho = \sigma_i^j$ for some $j \in [1..l]$. For $n_i = \mathsf{n}(\sigma_i)$ and $n_i^j = \mathsf{n}(\sigma_i^j)$, we have $m^+ \equiv_{\mathsf{occT}} n_i^j$ and $n_i^j \preceq_\cap n_i$, and consequently $m := n_i \in \mathsf{pre}(\tau)$. Thus, $(N[\Gamma] : m) = (n_i : m) \hookrightarrow \epsilon$.

Next, consider a formula Ψ of the form $\Gamma \vdash x_i N_1 \cdots N_k : \rho$ derived by (I$\rightarrow$), with $k > 0$. The derivation of Ψ results from successively applying rule (I$\rightarrow$) to $\Gamma \vdash x_i : \delta_1 \rightarrow \rho_1$ and $\Gamma \vdash N_1 : \delta_1$, to $\Gamma \vdash xN_1 : \delta_2 \rightarrow \rho_2$ and $\Gamma \vdash N_2 : \delta_2$, ..., to $\Gamma \vdash xN_1 \cdots N_{k-1} : \delta_k \rightarrow \rho_k$ and $\Gamma \vdash N_k : \delta_k$. Then, $\delta_1 \rightarrow \rho_1 \preceq_{\cap} \sigma_i$, $\delta_j \rightarrow \rho_j \preceq_{\cap} \rho_{j-1}$ for $j \in [2..k]$, and $\rho \preceq_{\cap} \rho_k$. Consider the identifiers of the subtypes assigned to terms in these formulas according to Definition 24. Let $c_j = \mathsf{n}(\mathsf{nst}(x_i N_1 \dots N_j))$ for $j \in [0..k]$, and $q_j = \mathsf{n}(\mathsf{pst}(N_j))$ for $j \in [1..k]$. Furthermore, let p_j be the identifier of the occurrence of ρ_j in $\mathsf{t}(c_{j-1}) = \delta_j \rightarrow \rho_j$ for $j \in [1..k]$. Then, $m^+ \equiv_{\mathsf{occT}} c_k \preceq_{\cap} p_k \rightarrowtail^{q_k} c_{k-1} \preceq_{\cap} \dots \preceq_{\cap} p_2 \rightarrowtail^{q_2} c_1 \preceq_{\cap} p_1 \rightarrowtail^{q_1} c_0 \preceq_{\cap} n_i$. We conclude that $m := n_i\ q_1 \cdots q_k \in \mathsf{pre}(\tau)$. Thus, $(N[\Gamma] : m) = (n_i(N_1[\Gamma]) \cdots (N_k[\Gamma]) : m) \hookrightarrow$ $(N_1[\Gamma] : q_1), \dots, (N_k[\Gamma] : q_k)$
$\hookrightarrow^* \epsilon$.

Let Ψ be $\Gamma \vdash \lambda x.N' : \sigma \rightarrow \delta$ obtained by rule (I$\rightarrow$) from $\Gamma' \vdash N' : \delta$, where $\Gamma' = \Gamma \cup \{x : \sigma\}$. For Ψ we have $m^+ = \mathsf{n}(\mathsf{pst}(\lambda x.N'))$. Consider for $\Gamma' \vdash N' : \delta$ the identifiers $k^- = \mathsf{n}(\mathsf{nsp}(x))$ and $n^+ = \mathsf{n}(\mathsf{pst}(N))$. Then, $n \rightarrowtail^k m$, and consequently $m := \lambda k.n \in \mathsf{pre}(\tau)$. Thus, $((\lambda x.N')[\Gamma] : m) \hookrightarrow (N'[\Gamma][k/x] : n) = (N'[\Gamma'], n) \hookrightarrow^* \epsilon$.

Finally, consider $\Gamma \vdash N : \tau_1 \cap \cdots \cap \tau_k$ derived from $\Gamma \vdash N : \tau_1, \dots, \Gamma \vdash N : \tau_k$ by rule by (I$\cap$), for some $k > 1$. Let $m^+ = \mathsf{n}(\mathsf{pst}(N))$ and $n_i = \mathsf{n}(\mathsf{pst}(N))$ in formulas $\Gamma \vdash N : \tau_i$, for $i \in [1..k]$. Then, $m := (n_1, \dots, n_k) \in \mathsf{pre}(\tau)$. Thus, $(N[\Gamma] : m) \hookrightarrow (N[\Gamma][1,k] : (n_1, \dots, n_k))$. By induction $(N[\Gamma] : n_i) \hookrightarrow^* \epsilon$, for $i \in [1..k]$. These rewriting sequences are, but for applications of rule 1 which creates replications in corresponding places, determined by the structure of $N[\Gamma]$ and can be combined to a rewriting sequence from $(N[\Gamma][1,k] : (n_1, \dots, n_k))$ to ϵ.

($\supseteq$) For the other inclusion consider a term M, such that $(M : \mathsf{n}(\tau)) \hookrightarrow^* \epsilon$. Let $(E : \vec{m})$ be any pair appearing in the corresponding rewriting sequence, where E is an extended term and $\vec{m} = (m_1, \dots, m_k)$. Furthermore, consider $\vec{P} = \{\vec{p_1}, \dots, \vec{p_l}\}$, the set of placeholders that occur in E, and let us interpret each tuple in $\vec{P}$ as the name of a term variable. We will show, by induction on the length of $(E, \vec{m}) \hookrightarrow^* \epsilon$, that for all $i \in [1..k]$ we have $\Gamma_i \vdash E : \mathsf{t}(\vec{m}|_i)$, where the projection $|_i$ is defined by $(s_1, \dots, s_k)|_i = s_i$ and $\Gamma_i = \{\vec{p_1} : \mathsf{t}(\vec{p_1}|_i), \dots, \vec{p_l} : \mathsf{t}(\vec{p_l}|_i)\}$. In particular, it follows that $\vdash M : \tau$.

Suppose that $(E : \vec{m}) \hookrightarrow (E[j,t] : \vec{m'})$, with $\vec{m'} =$ $\vec{m}[(s_1, \dots, s_t)/j] = (m_1, \dots, m_{j-1}, s_1, \dots, s_t, m_{j+1}, \dots, m_k)$, because $m_j := (s_1, \dots, s_t) \in \mathsf{pre}(\tau)$, for some $j \in [1..k]$. For this last pair the set of placeholders is now $\vec{P'} =$ $= \{\vec{p_1}[j,t], \dots, \vec{p_l}[j,t]\}$. If $j \neq i$, let i' be the new position of m_i in $\vec{m'}$, which is i if $i < j$, and equal to $i + t - 1$ if $i > j$. Then, $\Gamma_{i'} = \{\vec{p_1}[j,t] : \mathsf{t}(\vec{p_1}[j,t]|_{i'}), \dots, \vec{p_l}[j,t] : \mathsf{t}(\vec{p_l}[j,t]|_{i'})\} = \{\vec{p_1}[j,t] : \mathsf{t}(\vec{p_1}|_i), \dots, \vec{p_l}[j,t] : \mathsf{t}(\vec{p_l}|_i)\}$. By the induction hypothesis, $\Gamma_{i'} \vdash E[j,t] : \mathsf{t}(\vec{m'}|_{i'})$. This, means that $\Gamma_i \vdash E : \mathsf{t}(\vec{m}|_i)$, because $\vec{m}|_i = \vec{m'}|_{i'}$ and the only change in both formulas is the name of variables (from $\vec{p_h}$ to $\vec{p_h}[j,t]$). If $i = j$, then we have $\Gamma_i \vdash E : \mathsf{t}(\vec{m}|_i)$ if $\Gamma'_i \vdash E[i,t] : \mathsf{t}(s_r)$ for all $r \in [1..t]$, where $\Gamma'_i = \{\vec{p_1}[i,t] : \mathsf{t}(\vec{p_1}|_i), \dots, \vec{p_l}[i,t] : \mathsf{t}(\vec{p_l}|_i)\}$. It follows from the induction hypothesis (note that the position of s_r in $\vec{p}[i,t]$ is $i + r - 1$) that for $\Gamma^r_i = \{\vec{p_1}[i,t] : \mathsf{t}(\vec{p_1}[i,t]|_{i+r-1}), \dots, \vec{p_l}[i,t] : \mathsf{t}(\vec{p_l}[i,t]|_{i+r-1})\}$ we have $\Gamma^r_i \vdash E[i,t] : \mathsf{t}(s_r)$ for all $r \in [1..t]$. But, $\Gamma'_i = \Gamma^r_i$ and the result holds.

Now, suppose that $(\lambda x.E : \vec{m}) \hookrightarrow (N[\vec{v}/x] : \vec{n})$ because $m_j := \lambda v_j.n_j \in \mathsf{pre}(\tau)$, for all $j \in [1..k]$, where $\vec{v} = (v_1, \dots, v_k)$ and $\vec{n} = (n_1, \dots, n_k)$. In particular, $m_i := \lambda v_i.n_i \in \mathsf{pre}(\tau)$ and $\mathsf{t}(m_i) = \mathsf{t}(v_i) \rightarrow \mathsf{t}(n_i)$. By the induction hypothesis $\Gamma_i \cup \{\vec{v} : \mathsf{t}(v_i)\} \vdash E[\vec{v}/x] : \mathsf{t}(n_i)$. Thus, $\Gamma_i \vdash \lambda\vec{v}.E[\vec{v}/x] : \mathsf{t}(\vec{m}|_i)$, but $\lambda\vec{v}.E[\vec{v}/x] \equiv_\alpha \lambda x.E$.

Finally, consider $(\vec{v}\ E_1 \cdots E_s : \vec{m}) \hookrightarrow (E_1 : \vec{n_1}), \dots, (E_s : \vec{n_s})$ with $s \geq 0$, because $\vec{v} = (v_1, \dots, v_k)$, and $m_j := v_j\ n^j_1 \cdots n^j_s \in \mathsf{pre}(\tau)$, for all $j \in [1..k]$. In particular, we have $\vec{m}|_i = m_i := v_i\ n^i_1 \cdots n^i_s \in \mathsf{pre}(\tau)$. It follows from Definition 5 that there is a sequence of identifiers such that

$$m^+_i \equiv_{\mathsf{occT}} c_s \preceq_{\cap} q_s \rightarrowtail c_{s-1} \preceq_{\cap} \dots \preceq_{\cap} q_1 \rightarrowtail c_0 \preceq_{\cap} v^-_i,$$

and $\mathsf{lab}(q_t, c_{t-1}) = n^i_t$ for $t \in [1..s]$. Then, $\mathsf{t}(m_i) = \mathsf{t}(c_s) \preceq_\cap \mathsf{t}(q_s)$, $\mathsf{t}(c_{s-1}) = (\mathsf{t}(n^i_s) \to \mathsf{t}(q_s)), \dots, \mathsf{t}(c_0) = (\mathsf{t}(n^i_1) \to \mathsf{t}(q_1)) \preceq_\cap \mathsf{t}(v_i)$. We have $\vec{v} : \mathsf{t}(v_i) \in \Gamma_i$. Thus, $\Gamma_i \vdash \vec{v} : \mathsf{t}(n^i_1) \to \mathsf{t}(q_1)$ by (var). By the induction hypothesis, $\Gamma_{E_1} \vdash E_1 : \mathsf{t}(n^i_1)$, where Γ_{E_1} is the restriction of Γ_i to the free variables (placeholders) in E_1. Thus we also have $\Gamma_i \vdash E_1 : \mathsf{t}(n^i_1)$ and by rule (E$\to$) and $(\mathsf{t}(n^i_2) \to \mathsf{t}(q_2)) \preceq_\cap \mathsf{t}(q_1)$ it follows that $\Gamma_i \vdash \vec{v}\ E_1 : \mathsf{t}(n^i_2) \to \mathsf{t}(q_2)$, etc. Repeating this process we conclude that $\Gamma_i \vdash \vec{v}\ E_1 \cdots E_s : \mathsf{t}(c_s) = \mathsf{t}(m_i) = \mathsf{t}(\vec{m}|_i)$. ◀

Proof of Theorem 13

▶ **Definition 26.** *For a particular rewriting sequence of* $(\emptyset : \mathsf{n}(\tau)) \rightsquigarrow^* \epsilon$, *where in each step the combination of rewriting rules applied is given, we define a function* pair *that computes for each* $(\vec{V} : \vec{m})$ *in that rewriting sequence a tuple* $(\vec{\Gamma}, M) = \mathsf{pair}(\vec{V} : \vec{m})$, *where* M *is an extended term and* $\vec{\Gamma}$ *a set of placeholders with the arity of* $\vec{m}$. *The function* pair *is recursively defined as follows.*

1. *Let* $\vec{n} = (n_1, \dots, n_k)$ *and suppose that* $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V}[i,k] : \vec{m}[\vec{n}/i])$ *because* $m_i := \vec{n} \in \mathsf{pre}(\tau)$, *for some* $i \in [1..t]$ *and* $k \geq 2$. *If* $\mathsf{pair}((\vec{V}[i,k] : \vec{m}[\vec{n}/i]) = (\vec{\Gamma}, M)$, *then* $\mathsf{pair}(\vec{V} : \vec{m}) = (\vec{\Gamma}[i, k\downarrow], M[i, k\downarrow])$.
2. *If* $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} \cup \{\vec{k}\} : \vec{n})$ *because* $m_i := \lambda k_i.n_i \in \mathsf{pre}(\tau)$, *for all* $i \in [1..t]$, $\vec{k} = (k_1, \dots, k_t)$ *and* $\vec{n} = (n_1, \dots, n_t)$, *then* $\mathsf{pair}(\vec{V} : \vec{m}) = (\vec{\Gamma} \setminus \{\vec{k}\}, \lambda\vec{k}.N)$, *where* $(\vec{\Gamma}, N) = \mathsf{pair}(\vec{V} \cup \{\vec{k}\} : \vec{n})$.
3. *If* $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} : \vec{n_1}), \dots, (\vec{V} : \vec{n_s})$, *because* $m_i := k_i\ n^i_1 \cdots n^i_s \in \mathsf{pre}(\tau)$, *for all* $i \in [1..t]$, $\vec{k} = (k_1, \dots, k_t) \in \vec{V}$, *and* $\vec{n_j} = (n^1_j, \dots, n^t_j)$, *then* $\mathsf{pair}(\vec{V} : \vec{m}) = (\{\vec{k}\} \cup \vec{\Gamma_1} \cup \cdots \cup \vec{\Gamma_s}\ , \vec{k}\ N_1 \cdots N_s)$, *where* $(\vec{\Gamma_j}, N_j) = \mathsf{pair}(\vec{V} : \vec{n_j})$, *for* $1 \leq j \leq s$ $(s \geq 0)$.

The correctness of function pair is stated in the following lemma.

▶ **Lemma 27.** *Suppose that* $(\vec{V} : \vec{m}) \rightsquigarrow^* \epsilon$ *for* $\vec{m} = (m_1, \dots, m_t)$, $(t \geq 1)$, *and for some particular rewriting sequence* $(\vec{\Gamma}, M) = \mathsf{pair}(\vec{V} : \vec{m})$. *Let* $\vec{\Gamma}|_i$ *denote the context* $\{\ \vec{p}|_i : \mathsf{t}(\vec{p}|_i) \mid \vec{p} \in \vec{\Gamma}\ \}$. *Furthermore, let* $M|_i$ *be the result obtained by replacing every placeholder* $\vec{v}$ *in* M *by* $\vec{v}|_i$. *Then,* $\vec{\Gamma}|_j \vdash M|_j : \mathsf{t}(\vec{m}|_j)$, *for all* $j \in [1..t]$.

Proof. By induction on the length of the rewriting sequence. Let $\vec{n} = (n_1, \dots, n_k)$ and suppose that $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V}[i,k] : \vec{m}[\vec{n}/i])$ because $m_i := \vec{n} \in \mathsf{pre}(\tau)$, for some $i \in [1..t]$ and $k \geq 2$. Furthermore, let $\mathsf{pair}(\vec{V} : \vec{m}) = (\vec{\Gamma}[i, k\downarrow], M[i, k\downarrow])$, for $\mathsf{pair}(\vec{V}[i,k] : \vec{m}[\vec{n}/i]) = (\vec{\Gamma}, M)$. First, consider $j \neq i$ $(j \in [1..t])$ and let j' be the position of m_j in $\vec{m}[\vec{n}/i] = (m_1, \dots, m_{i-1}, n_1, \dots, n_k, m_{i+1}, \dots, m_t)$. By induction, we have $\vec{\Gamma}|_{j'} \vdash M|_{j'} : \mathsf{t}(\vec{m}[\vec{n}/i]|_{j'})$. But $(\vec{\Gamma}|_{j'}, M|_{j'}) = (\vec{\Gamma}[i, k\downarrow]|_j, M[i, k\downarrow]|_j)$ and $\mathsf{t}(\vec{m}[\vec{n}/i]|_{j'}) = \mathsf{t}(\vec{m}|_j)$. For $j = i$ note that $\mathsf{t}(m_i) = \mathsf{t}(n_1) \cap \cdots \cap \mathsf{t}(n_k)$. It follows from the induction hypothesis that $\vec{\Gamma}|_{i+r-1} \vdash M|_{i+r-1} : \mathsf{t}(\vec{m}[\vec{n}/i]|_{i+r-1})$ for all $r \in [1..k]$. But, $(\vec{\Gamma}|_{i+r-1}, M|_{i+r-1}) = (\vec{\Gamma}[i, k\downarrow]|_i, M[i, k\downarrow]|_i)$ and $\mathsf{t}(\vec{m}[\vec{n}/i]|_{i+r-1}) = \mathsf{t}(n_r)$. Thus, by rule ($\cap$I) we have $\vec{\Gamma}[i, k\downarrow]|_i \vdash M[i, k\downarrow] : \mathsf{t}(m_i)$.

Now suppose that $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} \cup \{\vec{k}\} : \vec{n})$, where $\vec{k} = (k_1, \dots, k_t)$ and $\vec{n} = (n_1, \dots, n_t)$ and such that $m_i := \lambda k_i.n_i \in \mathsf{pre}(\tau)$, for all $i \in [1..t]$. For $j \in [1..t]$ we have $\mathsf{t}(\vec{m}|_j) = \mathsf{t}(\vec{k}|_j) \to \mathsf{t}(\vec{n}|_j)$. Let $\mathsf{pair}(\vec{V} \cup \{\vec{k}\} : \vec{n}) = (\vec{\Gamma}, N)$ and consider $\mathsf{pair}(\vec{V} : \vec{m}) = (\vec{\Gamma} \setminus \{\vec{k}\}, \lambda\vec{k}.N)$. By the induction hypothesis $\vec{\Gamma}|_j \vdash N|_j : \mathsf{t}(\vec{n}|_j)$. Since $\vec{\Gamma}|_j \cup \{\vec{k}|_j : \mathsf{t}(\vec{k}|_j)\}$ is consistent by definition, it follows that $\vec{\Gamma}|_j \backslash \{\vec{k}|_j : \mathsf{t}(\vec{k}|_j)\} \vdash \lambda\vec{k}|_j.(N|_j) : \mathsf{t}(\vec{k}|_j) \to \mathsf{t}(\vec{n}|_j)$. But $\vec{\Gamma}|_j \backslash \{\vec{k}|_j : \mathsf{t}(\vec{k}|_j)\} = (\vec{\Gamma} \backslash \{\vec{k}\})|_j$ and $\lambda\vec{k}|_j.(N|_j) = (\lambda\vec{k}.N)|_j$.

Finally suppose that $(\vec{V} : \vec{m}) \rightsquigarrow (\vec{V} : \vec{n_1}), \dots, (\vec{V} : \vec{n_s})$, because $m_i := k_i\ n^i_1 \cdots n^i_s \in \mathsf{pre}(\tau)$, for all $i \in [1..t]$, $\vec{k} = (k_1, \dots, k_t) \in \vec{V}$, and $\vec{n_j} = (n^1_j, \dots, n^t_j)$, (Â§$1 \leq j \leq s$). Let $(\vec{\Gamma_h}, N_h) = \mathsf{pair}(\vec{V} : \vec{n_h})$, for $1 \leq h \leq s$ $(s \geq 0)$. Consider $\mathsf{pair}(\vec{V} : \vec{m}) = (\vec{\Gamma}, \vec{k}\ N_1 \cdots N_s)$, where $\vec{\Gamma} =$

$\{\vec{k}\}\cup\vec{\Gamma_1}\cup\cdots\cup\vec{\Gamma_s}$. By definition $\vec{\Gamma}|_j$ is consistent. It follows from the induction hypothesis that $\vec{\Gamma_h}|_j \vdash N_h|_j : \mathsf{t}(\vec{n_h}|_j)$. On the other hand, $\vec{\Gamma}|_j \vdash \vec{k}|_j : \mathsf{t}(\vec{k}|_j)$. Now, it remains to apply the same argument as in the proof of Theorem 10 to conclude that $\vec{\Gamma}|_j \vdash (\vec{k}\ N_1 \cdots N_s)|_j : \mathsf{t}(\vec{m}|_j)$. ◀

Theorem 13. $\mathsf{Nhabs}(\tau) \neq \emptyset$ if and only if $(\emptyset : \mathsf{n}(\tau)) \rightsquigarrow^* \epsilon$.

Proof. The 'if' part follows from Lemma 27. For the 'only if' part consider a term P such that $\vdash P : \tau$. By Theorem 10 there exists a $\hookrightarrow$-rewriting sequence, such that $(P : \mathsf{n}(\tau)) \hookrightarrow^* \epsilon$. We show, by induction on its length, that for each pair $(M : \vec{m})$ in that sequence we have $(\mathsf{FV}(M) : \vec{m}) \rightsquigarrow^* \epsilon$, where M is an extended term with placeholders figuring as names for term variables. In this part of the proof we also use the fact that $(\vec{V}, \vec{p}) \rightsquigarrow^* \epsilon$ implies $(\vec{V'}, \vec{p}) \rightsquigarrow^* \epsilon$, whenever $\vec{V} \subseteq \vec{V'}$. If $(M : \vec{m}) \hookrightarrow (M[i,k] : \vec{m}[\vec{n}/i])$, then $(\mathsf{FV}(M) : \vec{m}) \rightsquigarrow (\mathsf{FV}(M)[i,k] : \vec{m}[\vec{n}/i]) = (\mathsf{FV}(M[i,k]) : \vec{m}[\vec{n}/i])$ and the result follows from the induction hypothesis. If $(\lambda x.N : \vec{m}) \hookrightarrow (N[\vec{k}/x] : \vec{n})$, then $(\mathsf{FV}(\lambda x.N) : \vec{m}) \rightsquigarrow (\mathsf{FV}(\lambda x.N) \cup \{\vec{k}\} : \vec{n})$. But $\mathsf{FV}(N[\vec{k}/x]) \subseteq \mathsf{FV}(\lambda x.N) \cup \{\vec{k}\}$ and the result follows from the induction hypothesis. Finally, if $(\vec{k}\ N_1 \cdots N_s : \vec{m}) \hookrightarrow (N_1 : \vec{n_1}), \ldots, (N_s : \vec{n_s})$, then $(\mathsf{FV}(\vec{k}\ N_1 \cdots N_s) : \vec{m}) \rightsquigarrow (\mathsf{FV}(\vec{k}\ N_1 \cdots N_s) : \vec{n_1}), \ldots, (\mathsf{FV}(\vec{k}\ N_1 \cdots N_s) : \vec{n_s})$. Again $\mathsf{FV}(N_j) \subseteq \mathsf{FV}(\vec{k}\ N_1 \cdots N_s)$ and the result holds by induction. ◀

Proof of Proposition 16

Proposition 16. Let τ be a strict intersection type. Then, there exists M such that M can be typed with τ at bounded multiset dimension $\leq p$ (denoted by $\Vdash_p M : \tau$) if and only if $\tau \in \mathsf{INH}_p$.

Proof. We consider the decision procedure $\mathcal{J}$ given in [16], Section 6.1. The procedure transforms multisets of constraints of the form $\mathcal{C} = \langle \Gamma_1 \vdash ? : \gamma_1, \ldots, \Gamma_n \vdash ? : \gamma_n \rangle$, where all Γ_i's have the same domain and each γ_i is strict and not an intersection. Such a multiset is referred to as a configuration of the decision procedure. Consider a configuration $\mathcal{C}$ as above with $\mathsf{dom}(\Gamma_i) = \{x_1, \ldots, x_k\}$. Regarding some particular order in the multiset (for instance $1, \ldots, n$) we associate the pair $\mathcal{C}^{\nabla} = (\vec{V} : \vec{m})$, where $\vec{m} = (\mathsf{n}(\gamma_1), \ldots, \mathsf{n}(\gamma_n))$ and $\vec{V} = \{\ (\mathsf{n}(\Gamma_1(x_i)), \ldots, \mathsf{n}(\Gamma_n(x_i))) \mid x_i \in [1..k]\ \}$. In the other direction, let $(\vec{V} : \vec{m})^{\triangle} = (\{\vec{v}_1, \ldots, \vec{v}_k\} : (m_1, \ldots, m_n))^{\triangle} = \langle \Gamma_1 \vdash ? : \mathsf{t}(m_1), \ldots, \Gamma_n \vdash ? : \mathsf{t}(m_n) \rangle$, where $\Gamma_i(x_j) = \mathsf{t}(\vec{v}_j|_i)$, for $i \in [1..n]$ and $j \in [1..k]$. We have that $\mathsf{dim}(\mathcal{C}^{\nabla})$ equals the number of constraints in multiset $\mathcal{C}$, and the same is true for a pair $(\vec{V}, \vec{m})$ and configuration $(\vec{V}, \vec{m})^{\triangle}$.

Transformation step 1 of procedure $\mathcal{J}$ corresponds to an application of rule 2 by algorithm $\mathcal{I}$. Since in strict types intersections are not allowed on the right side of $\to$, rule 1 never applies right after rule 2. On the other hand, transformation step 2 in $\mathcal{J}$ corresponds to an application of rule 3, followed subsequently by all possible applications of rule 1 (which is incompatible with the remaining two rules) by algorithm $\mathcal{I}$.

Now, using these conversions from configurations[4] to pairs and back, it remains to show by induction on the height of trees that for every accepting computation tree $T^{\mathcal{J}}_{\langle \vdash ? : \tau_1, \ldots, \vdash ? : \tau_n \rangle}$ determined by a run of $\mathcal{J}$ (cf. [16]) there is an accepting computation tree Π for $\tau = \tau_1 \cap \cdots \cap \tau_n$ by $\mathcal{I}$, and vice-versa. The height of Π is typically $O(|\tau|) \cdot h$, where h is the height of $T^{\mathcal{J}}_{\langle \vdash ? : \tau \rangle}$, due to the fact that each step 2 of $\mathcal{J}$ corresponds to an application of rule 3 followed by all possible ($\leq |\tau|$) applications of rule 1 by $\mathcal{I}$. ◀

[4] For conversion $\cdot^{\nabla}$ one has to consider in each step a convenient order in $\mathcal{C}$.

Learning Automata and Transducers: A Categorical Approach

Thomas Colcombet
IRIF, CNRS, Paris, France
https://www.irif.fr/~colcombe/
thomas.colcombet@irif.fr

Daniela Petrişan
IRIF, Université de Paris, France
https://www.irif.fr/~petrisan/
daniela.petrisan@irif.fr

Riccardo Stabile
Università degli Studi di Milano, Dipartimento di Matematica, Italy
riccardo.stabile@yahoo.com

Abstract

In this paper, we present a categorical approach to learning automata over words, in the sense of the $\mathtt{L}^*$-algorithm of Angluin. This yields a new generic $\mathtt{L}^*$-like algorithm which can be instantiated for learning deterministic automata, automata weighted over fields, as well as subsequential transducers. The generic nature of our algorithm is obtained by adopting an approach in which automata are simply functors from a particular category representing words to a "computation category". We establish that the sufficient properties for yielding the existence of minimal automata (that were disclosed in a previous paper), in combination with some additional hypotheses relative to termination, ensure the correctness of our generic algorithm.

2012 ACM Subject Classification Theory of computation → Algebraic language theory; Theory of computation → Transducers

Keywords and phrases Automata, transducer, learning, category

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.15

Related Version A full version of the paper is available at [14], https://arxiv.org/abs/2010.13675.

Funding *Thomas Colcombet*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624) and the DeLTA ANR project (ANR-16-CE40-0007).
Daniela Petrişan: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624) and the DeLTA ANR project (ANR-16-CE40-0007).
Riccardo Stabile: Supported by the European Commission under the Erasmus+ programme for a five-month study period at Université de Paris.

1 Introduction

Learning automata is a classical subject at the intersection of machine learning and automata theory. It has found a wide range of applications spanning from adaptive model checking, compositional verification to learning network invariants or interface specifications for Java classes. We refer the reader to [18] and the references therein for a survey of such applications.

The most famous learning algorithm for automata is certainly the $\mathtt{L}^*$*-algorithm* of Angluin [1]. Its goal is to learn a regular language of words L. For this, the algorithm interacts with a *teacher* (an oracle) who knows L by asking two kinds of queries:

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 15; pp. 15:1–15:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Membership query it can ask whether a given word belongs to L, or

Equivalence query it can provide a *hypothesis automaton* and ask the teacher whether this automaton recognizes L or not. If the answer is no, the teacher is bound to provide a *counter-example word*, witnessing the non-equivalence.

The algorithm stops when the teacher agrees that the hypothesis automaton recognizes the language L. A key property of the $\mathtt{L}^*$-algorithm is that it terminates in time polynomial in the size of the alphabet, of the minimal deterministic automaton for L, and of the longest counter-example. Furthermore, all candidate automata appearing during its execution (and hence in particular the final one) are deterministic, complete, and minimal.

The $\mathtt{L}^*$-algorithm

Let us illustrate the behaviour of this algorithm when it tries to learn the language $\{a\}$ over the alphabet $\Sigma = \{a\}$. At each step, the algorithm maintains two sets of words Q, T, starting with $Q = \{\varepsilon\}$, $T = \{\varepsilon\}$. One can understand Q as a set of words which identify states of the hypothesis automaton under construction. The set T is used in order to discover if words need to be distinguished by the automaton: two words $u, v \in \Sigma^*$ are *T-equivalent* if for all $t \in T$, $ut \in L$ if and only if $vt \in L$.

At the beginning, the algorithm attempts to construct an automaton with as sole state $\varepsilon \in Q$, which has to be initial. In particular, the target of the transition labelled a issued from ε has to be determined. Such a transition should go to a state in Q which has to be T-equivalent to $\varepsilon a = a$. It fails since there are no such states in Q (we say that the pair Q, T fails to have the *closedness property*). The algorithm corrects it by adding the word a to Q. We reach $Q = \{\varepsilon, a\}$, $T = \{\varepsilon\}$. The algorithm now tries to construct an automaton with states $Q = \{\varepsilon, a\}$: this time, it is possible to construct an a-labelled transition from $\varepsilon \in Q$ to $a \in Q$. What should now be the a-labelled transition issued from a? It should be some state $q \in Q$ which is T-equivalent to aa. Luckily, there is one, namely ε. Hence, we succeed in constructing the left hypothesis automaton in Figure 1. The algorithm now queries for equivalence of the language of this automaton with the language. This fails since $a(aa)^* \neq L = \{a\}$, and hence the teacher answers in return a counter-example word, say aaa. The algorithm then adds (for reasons that are not detailed here) the prefix aa of aaa to Q, yielding $Q = \{\varepsilon, a, aa\}$, $T = \{\varepsilon\}$. Here, ε and aa are T-equivalent, but constructing an a-labeled transition from ε would yield a, while constructing one from aa would yield aaa, which is T-equivalent to ε. Hence, ε and aa cannot be merged as a same state (we say that Q, T fails to have the *consistency property*). The algorithm compensates it by adding a to T, thus yielding $Q = \{\varepsilon, a, aa\}$ and $T = \{\varepsilon, a\}$. Now, Q, T are both closed and consistent, and the right hypothesis automaton in Figure 1 is constructed. It recognizes $\{a\}$, and thus the teacher agrees and the algorithm terminates. It has constructed the minimal deterministic and complete automaton for the language $L = \{a\}$.

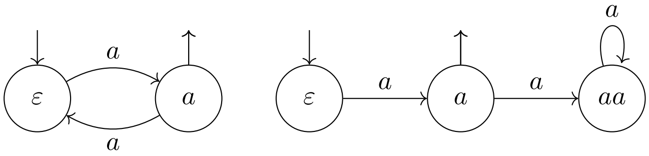


Figure 1 Two successive hypothesis automata.

This example witnesses the different steps involved in the algorithm: (a) if (Q, T) is not closed, a word is added to Q, (b) if (Q, T) is not consistent, a word is added to T, and (c) when (Q, T) is both closed and consistent, it is possible to construct a hypothesis automaton and perform an equivalence query: if this automaton happens to not accept the expected language, the teacher provides a counter-example word from which words to add to Q are constructed. The algorithm functions by performing the operation until the teacher agrees.

The correctness of the algorithm bears many resemblances with the question of minimizing deterministic automata. This can be witnessed in the fact that the L*-algorithm automatically constructs minimal deterministic and complete automata. It can also be witnessed in the fact that the T-equivalences induce along the run finer and finer partitions of the words that converge eventually to the Myhill-Nerode equivalence, another notion highly connected to minimization questions.

The L*-algorithm turns out to be extremely robust, and has been extended to various other forms of automata (weighted automata over fields [5, 6], nominal automata [19], omega automata [3], non-deterministic automata [9], alternating automata [2], symbolic automata [15], subsequential transducers [24,25], transducers of trees [7,8]). Although with a focus on concrete implementations, Bollig et al. [10] emphasize that "the need for a unifying framework collecting various types of learning techniques is, thus, beyond all questions."

Contributions

The aim of this paper is to present such a unifying framework for learning word automata using the toolkit of category theory. Concretely, we provide an abstract categorical version of Angluin's L*-algorithm, called FunL* (Algorithm 1), we prove its correctness and termination (Theorem 26), and we give three running instantiations for it, namely in the case of deterministic automata, field weighted automata and subsequential transducers.

To this end, we reuse the framework developed in [13] which models automata as functors from an input category $\mathcal{I}$ (describing the structure of the computation) to an output category $\mathcal{C}$. For example, to model word automata, the input category is a fixed three-object category $\mathcal{I}_{A^*}$, that we will recall in Section 2. By varying the category $\mathcal{C}$, this definition captures several forms of automata, and in particular the ones mentioned above. In [13], we present sufficient conditions on $\mathcal{C}$ that guarantee the existence of minimal automata. These conditions are quite mild: $\mathcal{C}$ should have certain products and coproducts, on one hand, and a factorization system, on the other. Apart from these three conditions on the output category, Theorem 26 – which states that our new algorithm computes the minimal automaton for the language to be learned – requires only one additional assumption which ensures termination, namely a 'finiteness' hypothesis (using the notion of noetherianity).

In order to describe our generic FunL* algorithm we provide abstract versions of the steps of the L*-algorithm described above. These are obtained as follows:

- We describe the pair of sets of words (Q, T) using a four-object category $\mathcal{I}_{Q,T}$, introduced in Definition 15. This category is a modification of $\mathcal{I}_{A^*}$, which allows us to obtain a partial view of the language, namely only its values on words of the form qt and qat with $q \in Q, t \in T$ and a a letter in the alphabet.
- Computing the approximations of the Myhill-Nerode equivalence (that is, the T-equivalence relations) roughly corresponds in our generic setting to performing a minimization-like computation. This is achieved using off-the-shelf results from [13] by changing the input category from $\mathcal{I}_{A^*}$ to $\mathcal{I}_{Q,T}$. We obtain a form of minimal "biautomaton" featuring an ε-transition between its two state objects.
- The pair (Q, T) being closed and consistent amounts to the ε-transition of the above biautomaton being an isomorphism between the two state objects. We then say that the pair (Q, T) is $\mathcal{L}$-automatable. Under this assumption, it is meaningful to collapse the two state objects, defining in this way the hypothesis automaton (represented now as a functor $\mathcal{I}_{A^*} \to \mathcal{C}$).

What is interesting about our FunL*-algorithm – compared to previous approaches – is that it highlights the strong relationship between learning and minimizing automata. Each elementary step of the algorithm involves performing a minimization-like computation and leverages the modularity of our previous work [13], this time by varying the input category. In contrast to other category theoretic approaches to learning, FunL* does not rely neither on algebras nor on coalgebras. Instead, we exploit the symmetry of the word automata model. This is reflected by the self-duality of the input category $\mathcal{I}_{A^*}$, which is underpinning the well known duality between observability and reachability.

Finally, a prominent instantiation of the FunL*-algorithm is Vilar's learning algorithm of subsequential transducers [24]. Our notion of $\mathcal{L}$-automatable pairs (Q, T) perfectly instantiates to the conditions considered by Vilar to construct a hypothesis transducer. A coalgebraic modelisation of subsequential transducers was provided in [16], but, to the best of our knowledge, this example is not featured in the category theoretic learning literature.

Related works

We briefly review the (co)algebraic approaches to automata learning that have been proposed in recent years. The paper [17] was the first to recast key ingredients of Angluin's algorithm in a coalgebraic setting. This line of work was continued with the CALF framework of van Heerdt et.al [22], which models automata as triples consisting of an algebra for a functor, an initial map and an output map. In [22, Section 5] a connection between minimization and learning is mentioned and formalized for DFAs. More precisely, the main theorem proving the correctness of the learning algorithm [22, Theorem 16] can be used to show the correctness of the minimization algorithm for DFA, with reachability and observability playing a crucial role. The same authors proposed in [23] a learning algorithm for automata with side-effects. These are extensions of DFAs to automata interpreted in a category of Eilenberg-Moore algebras for a Set monad T – used to represent a certain side effect. For example, the finite powerset monad corresponds to non-determinism and the ensuing automata model serves to represent non-deterministic automata. In order to prove the termination of the learning algorithm, the monad T above is assumed to preserve finite sets. Hence the monad that we use in the present work to model subsequential transducers does not fit in the scope of [23]. Another small difference is that we work within the Kleisli category.

Another category theoretic learning algorithm was proposed in [4] and provides a coalgebraic and duality theoretic foundation for learning bisimilarity quotients of state-based transition systems. The core idea is to use logical formulas as tests, taking stock of dual adjunctions between states and logical theories, formalized as algebra-coalgebra dualities.

The recent paper [21] gives a learning algorithm for automata whose transitions can be encoded both as algebras for a functor F on a category $\mathcal{C}$, and as coalgebras for the right adjoint of F (assumed to exist). The approximations of the Myhill-Nerode equivalence present in the learning algorithm are computed using factorizations of morphisms from approximations of an initial algebra (obtained using an initial chain) to approximations of a final coalgebra (obtained using a final co-chain). Some of the ingredients of this category theoretic algorithm are similar to ours, e.g. the heavy use of factorization systems or the notion of "finite" object in a category, however, there are more assumptions on the underlying category and on the preservation properties of the adjoint functors considered in the (co)-algebraic definition of the automata, see [21, Assumption 3.5] and [21, Assumption 4.1].

Structure of the paper

In Section 2, we present necessary material from [13], which includes in particular the categorical modeling of automata, its instantiation for deterministic finite automata, for field weighted automata and for subsequential transducers, and how to minimize them. This material is key in our description of the algorithm in Section 3. For simplicity, we describe first a slightly simplified version of the algorithm. The optimized version is the presented in [14, Appendix C]. Section 4 concludes.

2 Minimization

In this section, we recall the categorical approach to automata minimization from [12, 13].

2.1 Languages and automata as functors

We first recall the notion of automata as functors. We consider an arbitrary small category $\mathcal{I}$, called the *input category*, and one of its full subcategories $\mathcal{O}$, denoting by i the inclusion functor: $\mathcal{O} \xhookrightarrow{i} \mathcal{I}$. Intuitively, $\mathcal{I}$ represents the the inner computations performed by an automaton, and in particular its internal behaviour, while $\mathcal{O}$ represents the observable behaviour of the automaton and is used to define the language it accepts.

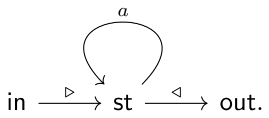
We consider another category $\mathcal{C}$, called the *output category*, which models the output computed by the automaton (e.g., a boolean value, probabilities, words over an alphabet).

▶ **Definition 1.** *A* $\mathcal{C}$-automaton *(or simply an automaton)* $\mathcal{A}$ *is a functor from* $\mathcal{I}$ *to* $\mathcal{C}$*. A* $\mathcal{C}$-language *(or simply a language)* $\mathcal{L}$ *is a functor from* $\mathcal{O}$ *to* $\mathcal{C}$*. A* $\mathcal{C}$*-automaton* $\mathcal{A}$ accepts *a* $\mathcal{C}$*-language* $\mathcal{L}$ *if* $\mathcal{A} \circ i = \mathcal{L}$*.*

We denote by $\mathsf{Auto}(\mathcal{L})$ *the subcategory of the functor category* $[\mathcal{I}, \mathcal{C}]$*:*

- *whose objects are all* $\mathcal{C}$*-automata* $\mathcal{A}$ *accepting* $\mathcal{L}$*;*
- *whose arrows are* $\mathcal{C}$-automata morphisms*, meaning natural transformations* $\alpha \colon \mathcal{A}_1 \Rightarrow \mathcal{A}_2$ *such that* $\alpha \circ id_i = id_{\mathcal{L}}$*.*

In this paper, we will instantiate the input category $\mathcal{I}$ in two ways. The first one, $\mathcal{I}_{A^*}$, is used in [12] to model different forms of *word automata*; we describe it in this section and use it for modeling the three running instantiations. In Section 3, we will consider another input category, $\mathcal{I}_{Q,T}$, which we use in the process of constructing our hypothesis automata.



We define now the input category $\mathcal{I}_{A^*}$ used for describing word automata. Here A is a finite alphabet, fixed for the rest of the paper, and A^* the set of words over it. The input category $\mathcal{I}_{A^*}$ is the category freely generated by the graph on the right, where a ranges over A: That is, $\mathcal{I}_{A^*}$ is the three-object category with arrows spanned by $\triangleright$, $\triangleleft$ and a for all $a \in A$, so that the composition $\mathsf{st} \xrightarrow{w} \mathsf{st} \xrightarrow{w'} \mathsf{st}$ is given by the concatenation ww'. So, for example, the morphisms in $\mathcal{I}_{A^*}$ from in to st are of the form $\triangleright w$ with $w \in A^*$, while the morphisms on the object st are of the form w with $w \in A^*$.

Let $\mathcal{O}_{A^*}$ denote the full subcategory of $\mathcal{I}_{A^*}$ on the objects in and out. Its morphisms are of the form $\mathsf{in} \xrightarrow{\triangleright w \triangleleft} \mathsf{out}$ for $w \in A^*$.

Hereafter, by a language we mean a functor from $\mathcal{O}_{A^*}$ to $\mathcal{C}$ and by an automaton we mean a functor from $\mathcal{I}_{A^*}$ to $\mathcal{C}$. If $\mathcal{L}(\mathsf{in}) = X$ and $\mathcal{L}(\mathsf{out}) = Y$, a language $\mathcal{L}$ will be referred to as a $(\mathcal{C}, X, Y)$-*language*; if $\mathcal{A}(\mathsf{in}) = X$ and $\mathcal{A}(\mathsf{out}) = Y$, an automaton $\mathcal{A}$ will be called a $(\mathcal{C}, X, Y)$-*automaton*. We provide three *running instantiations* of the output category $\mathcal{C}$ and of the objects X and Y, in order to model deterministic automata, field weighted automata and subsequential transducers.

▶ **Example 2** (Deterministic automata)**.** A deterministic and complete automaton is a $(\mathsf{Set}, 1, 2)$-automaton. Indeed, we can see a functor $\mathcal{A}\colon \mathcal{I}_{A^*} \to \mathsf{Set}$ with $\mathcal{A}(\mathsf{in}) = 1$ and $\mathcal{A}(\mathsf{out}) = 2$ as a deterministic automaton by interpreting

- $\mathcal{A}(\mathsf{st})$ as its set of states,
- $\mathcal{A}(\triangleright)\colon 1 \to \mathcal{A}(\mathsf{st})$ as choosing the initial state,
- $\mathcal{A}(a)\colon \mathcal{A}(\mathsf{st}) \to \mathcal{A}(\mathsf{st})$ as the transition map for the letter $a \in A$,
- $\mathcal{A}(\triangleleft)\colon \mathcal{A}(\mathsf{st}) \to 2$ as the characteristic map of the subset of accepting states.

▶ **Example 3** (Weighted automata over a field)**.** Let $\mathbb{K}$ be a field and let Vec denote the corresponding category of $\mathbb{K}$-vector spaces and linear transformations. A *weighted automaton over the field* $\mathbb{K}$ (in the sense of [20]) is a $(\mathsf{Vec}, \mathbb{K}, \mathbb{K})$-automaton. Indeed, a functor $\mathcal{A}\colon \mathcal{I}_{A^*} \to \mathsf{Vec}$ with $\mathcal{A}(\mathsf{in}) = \mathbb{K}$ and $\mathcal{A}(\mathsf{out}) = \mathbb{K}$ is seen as a weighted automaton over $\mathbb{K}$ by interpreting

- $\mathcal{A}(\mathsf{st})$ as the vector space spanned by its states,
- $\mathcal{A}(\triangleright)\colon \mathbb{K} \to \mathcal{A}(\mathsf{st})$ as the linear transformation mapping the unit of $\mathbb{K}$ to the initial vector,
- $\mathcal{A}(a)\colon \mathcal{A}(\mathsf{st}) \to \mathcal{A}(\mathsf{st})$ as the linear transformation transition for the letter $a \in A$,
- $\mathcal{A}(\triangleleft)\colon \mathcal{A}(\mathsf{st}) \to \mathbb{K}$ as the output linear transformation.

▶ **Example 4** (Subsequential transducers)**.** The aim is to represent what we call *transductions* in this paper, which are partial maps from A^* to B^*, where B is some fixed output alphabet. Roughly, a subsequential transducer [11] is a deterministic automaton which, at each step, while reading an input letter from A, either has no transition or has a unique transition which changes deterministically the state and outputs a word from B^*. In this paper, we define *subsequential transducers* as $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-automata [13], for a definition of $\mathsf{Kl}(\mathcal{T})$ that we give now.

The **output category** $\mathsf{Kl}(\mathcal{T})$**.** Let $\mathcal{T}$ be the monad defined by $\mathcal{T}X = B^* \times X + 1$ and let $\mathsf{Kl}(\mathcal{T})$ denote the Kleisli category for $\mathcal{T}$. Concretely, the objects of $\mathsf{Kl}(\mathcal{T})$ are sets, while its morphisms, denoted by negated arrows, are of the form $f\colon X \nrightarrow Y$ for a function $f\colon X \to \mathcal{T}Y$, that is, a partial function from X to $B^* \times Y$. We write $\perp$ for the element of the singleton 1 and think of it as the *undefined* element. Given $f\colon X \nrightarrow Y$ and $g\colon Y \nrightarrow Z$, their composite $g \circ f\colon X \nrightarrow Z$ is defined on $x \in X$ by (uv, z), when $f(x) = (u, y) \in B^* \times Y$ and $g(y) = (v, z) \in B^* \times Z$ (with uv denoting the concatenation of u and v in B^*) and $f(x) = \perp$ in all other cases. Note that with this definition, a transduction can be identified in an obvious manner with a map from A^* to arrows of the form $1 \nrightarrow 1$.

We now recall that $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-automata are equivalent to *subsequential transducers* in the sense of Choffrut's definition [11]. Indeed, we can see a functor $\mathcal{A}\colon \mathcal{I}_{A^*} \to \mathsf{Kl}(\mathcal{T})$ with $\mathcal{A}(\mathsf{in}) = 1$ and $\mathcal{A}(\mathsf{out}) = 1$ as a subsequential transducer by interpreting

- $\mathcal{A}(\mathsf{st})$ as the set of states,
- $\mathcal{A}(\triangleright)\colon 1 \nrightarrow \mathcal{A}(\mathsf{st})$ as either choosing an initial state together with an initial output in B^* or having an undefined initial state,
- $\mathcal{A}(a)\colon \mathcal{A}(\mathsf{st}) \nrightarrow \mathcal{A}(\mathsf{st})$ as the transition map for the letter a which associates to a given state either undefined or a pair consisting of an output word in B^* and a successor state,
- $\mathcal{A}(\triangleleft)\colon \mathcal{A}(\mathsf{st}) \nrightarrow 1$ as the final map which associates to a state either its output in B^* or undefined when it is non-accepting.

2.2 Minimization of automata

Now we describe what it means to be minimal in a category (Definition 5) together with an abstract result of existence of such an object (Lemma 6). We then provide sufficient material for our three running instantiations to be covered.

Let $\mathcal{K}$ be a category endowed with a factorization system $(\mathcal{E}, \mathcal{M})$. We write $\twoheadrightarrow$ for arrows belonging to $\mathcal{E}$ and we will call them $\mathcal{E}$*-quotients*; we write $\rightarrowtail$ for arrows belonging to $\mathcal{M}$ and we will call them $\mathcal{M}$*-subobjects.*

▶ **Definition 5.** *Consider two objects X, Y of $\mathcal{K}$. We say that X $(\mathcal{E}, \mathcal{M})$-*divides *$Y$ whenever X is an $\mathcal{E}$-quotient of an $\mathcal{M}$-subobject of Y, that is, we have a span of the form:*

$$X \twoheadleftarrow \cdot \rightarrowtail Y\,.$$

*An object Z in $\mathcal{K}$ is $(\mathcal{E}, \mathcal{M})$-*minimal *if it $(\mathcal{E}, \mathcal{M})$-divides all the objects in $\mathcal{K}$.*

As shown in the following lemma, having an initial and a final object turns out to be a sufficient condition for the minimal object to exist and be unique up to isomorphism.

▶ **Lemma 6.** *Let $\mathcal{K}$ be a category endowed with an initial object I, a final object F and a factorization system $(\mathcal{E}, \mathcal{M})$. Let* Min *be the factorization of the unique arrow from I to F:*

$$I \twoheadrightarrow \mathsf{Min} \rightarrowtail F.$$

Then Min *is $(\mathcal{E}, \mathcal{M})$-minimal.*

We apply this lemma when $\mathcal{K}$ is instantiated with a category of automata $\mathsf{Auto}(\mathcal{L})$.

▶ **Corollary 7.** *If the category $\mathsf{Auto}(\mathcal{L})$ has an initial automaton $\mathcal{A}_{init}(\mathcal{L})$, a final automaton $\mathcal{A}_{final}(\mathcal{L})$ and a factorization system, then the minimal automaton $\mathsf{Min}(\mathcal{L})$ for the language $\mathcal{L}$ is obtained via the following factorization: $\mathcal{A}_{init}(\mathcal{L}) \twoheadrightarrow \mathsf{Min}(\mathcal{L}) \rightarrowtail \mathcal{A}_{final}(\mathcal{L})$.*

Notice that this notion of minimization is parametric in the factorization system. In all our examples, we obtain a suitable factorization system on $\mathsf{Auto}(\mathcal{L})$ from one on $\mathcal{C}$, as follows.

▶ **Lemma 8.** *If a category $\mathcal{C}$ has a factorization system $(\mathcal{E}, \mathcal{M})$, then the category $\mathsf{Auto}(\mathcal{L})$ has a factorization system $(\mathcal{E}_{\mathsf{Auto}(\mathcal{L})}, \mathcal{M}_{\mathsf{Auto}(\mathcal{L})})$, where $\mathcal{E}_{\mathsf{Auto}(\mathcal{L})}$ consists of all natural transformations with components in $\mathcal{E}$ and $\mathcal{M}_{\mathsf{Auto}(\mathcal{L})}$ consists of all natural transformations with components in $\mathcal{M}$.*

▶ **Example 9** (factorization systems)**.** There exists a factorization system in our three running examples. For Set, this is the well known factorization system $(\mathrm{Surjections}, \mathrm{Injections})$. Similarly in Vec, (Surjective linear maps, Injective linear maps) is a factorization system.

In the case of $\mathsf{Kl}(\mathcal{T})$, the factorization system does not follow from general arguments. We define now the factorization system $(\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}, \mathcal{M}_{\mathsf{Kl}(\mathcal{T})})$ for $\mathsf{Kl}(\mathcal{T})$. Given a morphism $f \colon X \nrightarrow Y$ in $\mathsf{Kl}(\mathcal{T})$, we write $\pi_1(f) \colon X \to B^* + \{\bot\}$ and $\pi_2(f) \colon X \to Y + \{\bot\}$ for the projections: if $f(x) = \bot$ then $\pi_1(x) = \pi_2(x) = \bot$, otherwise $f(x) = (\pi_1(f)(x), \pi_2(f)(x))$.

The class $\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}$ consists of all the morphisms of the form $e \colon X \nrightarrow Y$ such that $\pi_2(e)$ is surjective (i.e. for every $y \in Y$ there exists $x \in X$ so that $\pi_2(e)(x) = y$) and the class $\mathcal{M}_{\mathsf{Kl}(\mathcal{T})}$ consists of all the morphisms of the form $m \colon X \nrightarrow Y$ such that $\pi_2(m)$ is injective and $\pi_1(m)$ is the constant function mapping every $x \in X$ to ε.

By [13, Lemma 4.8], $(\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}, \mathcal{M}_{\mathsf{Kl}(\mathcal{T})})$ is a factorization system.

We specialize the result of Corollary 7 to the case of word automata $\mathcal{I}_{A^*} \to \mathcal{C}$. Due to the special shape of the category $\mathcal{I}_{A^*}$, we can compute the initial and the final automata, provided the output category satisfies some mild assumptions, recalled in Lemmas 10 and 12.

▶ **Lemma 10.** *Fix a language $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathcal{C}$. If the category $\mathcal{C}$ has countable copowers of $\mathcal{L}(\mathsf{in})$, the* initial automaton $\mathcal{A}_{init}(\mathcal{L})$ *exists and is given by the following data:*

- $\mathcal{A}_{init}(\mathcal{L})(\mathsf{st}) = \coprod_{A^*} \mathcal{L}(\mathsf{in})$*;*
- $\mathcal{A}_{init}(\mathcal{L})(\triangleright)\colon \mathcal{L}(\mathsf{in}) \to \coprod_{A^*} \mathcal{L}(\mathsf{in})$ *is given by the coproduct injection corresponding to ε, for this reason we will denote this map by ε;*
- $\mathcal{A}_{init}(\mathcal{L})(a)\colon \coprod_{A^*} \mathcal{L}(\mathsf{in}) \to \coprod_{A^*} \mathcal{L}(\mathsf{in})$ *is given on the w-component $\mathcal{L}(\mathsf{in})$ by the coproduct injection corresponding to wa;*
- $\mathcal{A}_{init}(\mathcal{L})(\triangleleft)\colon \coprod_{A^*} \mathcal{L}(\mathsf{in}) \to \mathcal{L}(\mathsf{out})$ *is the coproduct of the morphisms $\mathcal{L}(\triangleright w \triangleleft)\colon \mathcal{L}(\mathsf{in}) \to \mathcal{L}(\mathsf{out})$ with $w \in A^*$, that is, it computes the value of the language on a given word, for this reason we will also denote this map by $\mathcal{L}?$.*

▶ **Example 11.** Since the categories Set, Vec and $\mathsf{Kl}(\mathcal{T})$ have all copowers, the initial automaton for a given language can be easily computed in these cases as an instance of the above lemma. We recall the details for Set and $\mathsf{Kl}(\mathcal{T})$.

$$\begin{tikzcd} 1 \arrow[r, "\varepsilon"] & A^* \arrow[loop above, "w\mapsto wa"] \arrow[r, "\mathcal{L}?"] & 2 \end{tikzcd}$$

$$\begin{tikzcd} 1 \arrow[r, "(\varepsilon{,}\varepsilon)", "/" marking] & A^* \arrow[loop above, "w\mapsto(\varepsilon{,}wa)", "/" marking] \arrow[r, "\mathcal{L}?", "/" marking] & 1 \end{tikzcd}$$

- Given a language $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathsf{Set}$, the initial deterministic automaton accepting $\mathcal{L}$ is described above in the left diagram. Its state space is the set of all words, with ε being the initial one. A word is accepted if and only if it belongs to the language.
- For a language $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathsf{Kl}(\mathcal{T})$, the initial subsequential transducer accepting $\mathcal{L}$ is as depicted in the right diagram. Its state space is A^*, the initial state is $\varepsilon \in A^*$ with initial output $\varepsilon \in B^*$. For an input letter $a \in A$, the corresponding transition maps w to wa and produces output $\varepsilon \in B^*$. Finally, the map $\mathcal{L}?$, which is in fact a function from A^* to $B^* + 1$, associates to a word w the value of the language at w, that is, the value computed by $\mathcal{L}(\triangleright w \triangleleft)$.

▶ **Lemma 12.** *Fix a language $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathcal{C}$. If the category $\mathcal{C}$ has countable powers of $\mathcal{L}(\mathsf{out})$, the* final automaton $\mathcal{A}_{final}(\mathcal{L})$ *exists and is given by the following data:*

- $\mathcal{A}_{final}(\mathcal{L})(\mathsf{st}) = \prod_{A^*} \mathcal{L}(\mathsf{out})$*;*
- $\mathcal{A}_{final}(\mathcal{L})(\triangleright)\colon \mathcal{L}(\mathsf{in}) \to \prod_{A^*} \mathcal{L}(\mathsf{out})$ *is the product of the morphisms $\mathcal{L}(\triangleright w \triangleleft)\colon \mathcal{L}(\mathsf{in}) \to \mathcal{L}(\mathsf{out})$ with $w \in A^*$, for this reason we will also denote this map by $\mathcal{L}$;*
- $\mathcal{A}_{final}(\mathcal{L})(a)\colon \prod_{A^*} \mathcal{L}(\mathsf{out}) \to \prod_{A^*} \mathcal{L}(\mathsf{out})$ *is the product over $w \in A^*$ of the aw-projections $\prod_{A^*} \mathcal{L}(\mathsf{out}) \to \mathcal{L}(\mathsf{out})$;*
- $\mathcal{A}_{final}(\mathcal{L})(\triangleleft)\colon \prod_{A^*} \mathcal{L}(\mathsf{out}) \to \mathcal{L}(\mathsf{out})$ *is given by the ε-projection, for this reason we will also denote this map by $\varepsilon?$.*

▶ **Example 13** (the final automata in Set, Vec and $\mathsf{Kl}(\mathcal{T})$)**.** Since the categories Set and Vec have all products, the final automaton for a given language can be computed using Lemma 12. We illustrate this for Set and $\mathsf{Kl}(\mathcal{T})$.

$$\begin{tikzcd} 1 \arrow[r, "\mathcal{L}"] & 2^{A^*} \arrow[loop above, "K \mapsto a^{-1}K"] \arrow[r, "K \mapsto K(\varepsilon)"] & 2 \end{tikzcd} \qquad \begin{tikzcd} 1 \arrow[r, "/" marking, "(\mathsf{lcp}(\mathcal{L}){,}\mathsf{red}(\mathcal{L}))"] & \mathsf{Irr}(A^*, B^*) \arrow[loop above, "/" marking, "K \mapsto (\mathsf{lcp}(K){,}\mathsf{red}(K))"] \arrow[r, "/" marking, "K \mapsto K(\varepsilon)"] & 1 \end{tikzcd}$$

- Given a language $\mathcal{L} \colon \mathcal{O}_{A^*} \to \mathsf{Set}$, the final deterministic automaton accepting $\mathcal{L}$ is described above in the left diagram. Its state space is the set of all languages over the alphabet A. The initial state is the language $\mathcal{L}$ itself. A language is an accepting state if and only if it contains ε. Given a language K, while reading letter a, the automaton goes to the residual language $a^{-1}K = \{u \in K \mid au \in K\}$.
- Somewhat suprisingly, $\mathsf{Kl}(\mathcal{T})$-automata also fit in the scope of Lemma 12, as we can prove that the object $\mathsf{Irr}(A^*, B^*)$ (which we will define next) is the power of A^*-many copies of 1 in $\mathsf{Kl}(\mathcal{T})$. Define first, given a transduction K, $\mathsf{lcp}(K)$ to be undefined if K is nowhere defined, and the longest common prefix of the words in $\{K(u) \mid u \in A^*\}$ otherwise. A transduction K is *irreducible* if $\mathsf{lcp}(K) = \varepsilon$. We denote by $\mathsf{Irr}(A^*, B^*)$ the set of irreducible transductions. For all K not nowhere defined, we put $\mathsf{red}(K)$ to be the only irreducible transduction such that $K(u) = \mathsf{lcp}(K)\mathsf{red}(K)(u)$, i.e. the transduction in which the prefix $\mathsf{lcp}(K)$ has been stripped away from all outputs. For K nowhere defined, let $\mathsf{red}(K)$ be also nowhere defined.

  We can describe now the final automaton for a transduction $\mathcal{L}$ as an automaton that has irreducible transductions as states. The initial map is the constant map equal to $(\mathsf{lcp}(\mathcal{L}), \mathsf{red}(\mathcal{L}))$ (or undefined if $\mathcal{L}$ is nowhere defined). When reading the letter a from state K, the automaton jumps to $\mathsf{red}(K(a-))$ in which $K(a-)$ is such that $K(a-)(u) = K(au)$ (or undefined if $K(a-)$ is nowhere defined). The final map sends an irreducible transduction to $K(\varepsilon)$.

$$\begin{tikzcd} & \coprod_{A^*} \mathcal{L}(\mathsf{in}) \arrow[dr, bend left, "\mathcal{L}?"] \arrow[d, two heads, "e_{min}"] & \\ \mathcal{L}(\mathsf{in}) \arrow[ur, bend left, "\varepsilon"] \arrow[dr, bend right, "\mathcal{L}"'] \arrow[r] & \mathsf{Min}(\mathcal{L})(\mathsf{st}) \arrow[r] \arrow[d, tail, "m_{min}"] & \mathcal{L}(\mathsf{out}) \\ & \prod_{A^*} \mathcal{L}(\mathsf{out}) \arrow[ur, bend right, "\varepsilon?"'] & \end{tikzcd}$$

Combining Lemmas 8, 10 and 12 with Corollary 7, we obtain the following result.

▶ **Theorem 14.** *Let $\mathcal{C}$ be a category with a factorization system $(\mathcal{E}, \mathcal{M})$ and let $\mathcal{L} \colon \mathcal{O}_{A^*} \to \mathcal{C}$ be a language. Suppose $\mathcal{C}$ has all countable copowers of $\mathcal{L}(\mathsf{in})$ and all countable powers of $\mathcal{L}(\mathsf{out})$. The minimal $\mathcal{C}$-automaton $\mathsf{Min}(\mathcal{L})$ accepting $\mathcal{L}$ is obtained via the factorization in the commuting diagram to the right.*

3 The basic FunL* algorithm

In this section, we provide our generic FunL*-algorithm for learning word automata. Just as in Angluin's algorithm, there are a teacher and a learner. Throughout this section we fix the alphabet A, the output category $\mathcal{C}$ and its factorization systems $(\mathcal{E}, \mathcal{M})$, all known to both teacher and learner. The *teacher* knows a language $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathcal{C}$, hereafter called the *target language*. The learner wants to find this language, the output of the algorithm being the minimal automaton $\mathsf{Min}(\mathcal{L})$ accepting $\mathcal{L}$. The learner can ask two kinds of queries, which can be thought as high-level generalizations of Angluin's original ones in the special case of deterministic automata (see [1]).

- *Evaluation queries*: given a certain word w, what is $\mathcal{L}(\triangleright w \triangleleft)$?
- *Equivalence queries*: does a certain automaton accept the target language? If it does not, what is a counterexample for it not doing that?

Let $\mathcal{A}$ be an automaton which is incorrect, that is, such that $\mathcal{A} \circ i \neq \mathcal{L}$, $\mathcal{L}$ being the target language; a word w is said to be a *counterexample* if $\mathcal{A} \circ i(\triangleright w \triangleleft) \neq \mathcal{L}(\triangleright w \triangleleft)$. In other words, a counterexample witnesses the incorrectness of a certain automaton proposed by the learner.

In order to formulate the generic algorithm, we still need to generalize the notions of table and hypothesis automaton from Angluin's original algorithm. We do this in Section 3.1. We provide the generic algorithm and prove its correctness and termination in Section 3.2.

3.1 Hypothesis automata

Just as in Angluin's L^*-algorithm, the learner keeps in memory a pair (Q, T) of subsets of A^* such that Q is prefix-closed, i.e. it contains the prefixes of all its elements, while T is suffix-closed, the same for the suffixes; in particular, $\varepsilon \in Q \cap T$. Using the evaluation queries, the learner produces an approximation of $\mathsf{Min}(\mathcal{L})$, explicitly a hypothesis automaton, to be introduced in Definition 22.

It turns out that the category $\mathsf{Auto}(\mathcal{L})$ does not suffice to capture the whole learning process. At a given stage of the algorithm, the learner has access, via evaluation queries, only to a part of $\mathcal{L}$: specifically, he knows the values of $\mathcal{L}(\triangleright qt \triangleleft)$ and $\mathcal{L}(\triangleright qat \triangleleft)$, where $q \in Q$, $t \in T$ and $a \in A$. This leads us to consider a restriction of the language $\mathcal{L}$ to a subcategory of $\mathcal{O}_{A^*}$ whose arrows are of the form $\triangleright qt \triangleleft$ or $\triangleright qat \triangleleft$ as above. To produce a hypothesis automaton consistent with this partial view of $\mathcal{L}$, we would also need to adapt the input category. A first attempt would be to discard some of the arrows of $\mathcal{I}_{A^*}$ from in to st, respectively from st to out. Explicitly, we would like to keep only the arrows of the form $\triangleright q\colon \mathsf{in} \to \mathsf{st}$ for the state words $q \in Q$ and, respectively, $t \triangleleft\colon \mathsf{st} \to \mathsf{out}$ for the test words $t \in T$. However, this is not feasible: we would also need the transition maps $a\colon \mathsf{st} \to \mathsf{st}$, and via composition we would generate, for example, all arrows $\triangleright w\colon \mathsf{in} \to \mathsf{st}$. The solution is to "dissociate" the state object st in $\mathcal{I}_{A^*}$ and consider a four-state input category.

▶ **Definition 15.** *The input category $\mathcal{I}_{Q,T}$ is the free category generated by the graph*

$$\begin{tikzcd}
\mathsf{in} \arrow[r, "\triangleright q"] & \mathsf{st}_1 \arrow[r, shift left, "a"] \arrow[r, shift right, "\varepsilon"'] & \mathsf{st}_2 \arrow[r, "t\triangleleft"] & \mathsf{out}
\end{tikzcd}$$

for all $q \in Q$, $a \in A$, $t \in T$ and with ε a fixed symbol (informally representing the empty word) such that the following coherence diagrams *commute for all $a \in A$, for all $q \in Q$ such that $qa \in Q$, and for all $t \in T$ such that $at \in T$:*

$$\begin{tikzcd}
 & \mathsf{st}_1 \arrow[dr, "a"] & & & & \mathsf{st}_2 \arrow[dr, "t\triangleleft"] & \\
\mathsf{in} \arrow[ur, "\triangleright q"] \arrow[dr, "\triangleright qa"'] & & \mathsf{st}_2; & & \mathsf{st}_1 \arrow[ur, "a"] \arrow[dr, "\varepsilon"'] & & \mathsf{out}. \\
 & \mathsf{st}_1 \arrow[ur, "\varepsilon"'] & & & & \mathsf{st}_2 \arrow[ur, "at\triangleleft"'] &
\end{tikzcd}$$

Furthermore, let $\mathcal{O}_{Q,T}$ denote the full subcategory of $\mathcal{I}_{Q,T}$ on the objects in and out.

The two coherence diagrams in the definition of $\mathcal{I}_{Q,T}$, as well as the prefix-closure of Q and the suffix-closure of T, ensure that we have a functor

$$i^* \colon \mathcal{I}_{Q,T} \to \mathcal{I}_{A^*}$$

which merges st_1 and st_2 sending both of them to st, maps $\varepsilon \colon \mathsf{st}_1 \to \mathsf{st}_2$ to the identity on st and maps all the other morphisms of $\mathcal{I}_{Q,T}$ to the homonymous ones in $\mathcal{I}_{A^*}$.

▶ **Lemma 16.** *The functor $i^* \colon \mathcal{I}_{Q,T} \to \mathcal{I}_{A^*}$ is well defined and, furthermore, $\mathcal{O}_{Q,T}$ is a subcategory of $\mathcal{O}_{A^*}$. That is, we have the following commuting diagram:*

$$\begin{tikzcd}
\mathcal{O}_{Q,T} \arrow[r, hook] \arrow[d, hook] & \mathcal{O}_{A^*} \arrow[d, hook] \\
\mathcal{I}_{Q,T} \arrow[r, "i^*"] & \mathcal{I}_{A^*}.
\end{tikzcd}$$

The partial knowledge of the language $\mathcal{L}$ the learner has access to at this given stage of the algorithm is captured by the restriction $\mathcal{L}_{Q,T}$ of $\mathcal{L}$ to $\mathcal{O}_{Q,T}$:

$$\mathcal{L}_{Q,T} \colon \mathcal{O}_{Q,T} \hookrightarrow \mathcal{O}_{A^*} \xrightarrow{\mathcal{L}} \mathcal{C}.$$

Hence, to a pair (Q,T) we can associate the category $\mathsf{Auto}(\mathcal{L}_{Q,T})$ obtained by instantiating in Definition 1 the input category $\mathcal{I}$ with $\mathcal{I}_{Q,T}$ and its observable behaviour subcategory with $\mathcal{O}_{Q,T} \hookrightarrow \mathcal{I}_{Q,T}$.

▶ **Definition 17.** *We call a functor $\mathcal{B}$ in $\mathsf{Auto}(\mathcal{L}_{Q,T})$ a* (Q,T)-biautomaton $\mathcal{B}$ *or a* $\mathcal{C}_{Q,T}$-biautomaton, *if we want to underline the dependence on $\mathcal{C}$. We say that $\mathcal{B}$ is* consistent *with the $\mathcal{C}$-language $\mathcal{L}$.*

In the $\mathtt{L}^*$-algorithm, the learner constructs a table associated to each pair of subsets (Q,T). This is done essentially by computing the quotient of the state words in Q by an approximation $\sim_T$ of the Myhill-Nerode equivalence for a language L given by: $w \sim_T v$ iff for all $t \in T$ we have $wt \in L \Leftrightarrow vt \in L$. This leads us to consider as a generalization of the notion of *table* the *minimal biautomaton* $\mathsf{Min}(\mathcal{L}_{Q,T})$ in the category $\mathsf{Auto}(\mathcal{L}_{Q,T})$. In order to compute it, we use Corollary 7. To this end, we first exhibit explicitly the initial and the final objects of $\mathsf{Auto}(\mathcal{L}_{Q,T})$, assuming that the output category $\mathcal{C}$ has got certain products and coproducts.

We will use the following notation. Given two subsets R and S of A^*, let RS denote the set $\{xy \colon x \in R, y \in S\}$.

▶ **Lemma 18.** *Assume $\mathcal{C}$ has all countable copowers of $\mathcal{L}(\mathsf{in})$. The initial $\mathcal{C}_{Q,T}$-biautomaton is the functor $\mathcal{A}_{init}(\mathcal{L}_{Q,T}) \colon \mathcal{I}_{Q,T} \to \mathcal{C}$ described in the next diagram*

$$\begin{tikzcd}
\mathcal{L}(\mathsf{in}) \arrow[r, "\triangleright q_{init}"] & \coprod\limits_{Q} \mathcal{L}(\mathsf{in}) \arrow[r, shift left, "a_{init}"] \arrow[r, shift right, "\varepsilon_{init}"'] & \coprod\limits_{Q \cup QA} \mathcal{L}(\mathsf{in}) \arrow[r, "t \triangleleft_{init}"] & \mathcal{L}(\mathsf{out}),
\end{tikzcd}$$

where, explicitly:

- $\mathcal{A}_{init}(\mathcal{L}_{Q,T})(\mathsf{st}_1) = \coprod\limits_{Q} \mathcal{L}(\mathsf{in})$ *and* $\mathcal{A}_{init}(\mathcal{L}_{Q,T})(\mathsf{st}_2) = \coprod\limits_{Q \cup QA} \mathcal{L}(\mathsf{in})$*;*
- $\triangleright q_{init} := \mathcal{A}_{init}(\mathcal{L}_{Q,T})(\triangleright q)$ *is the coproduct injection j_q of $\mathcal{L}(\mathsf{in})$ into* $\coprod\limits_{Q} \mathcal{L}(\mathsf{in})$*;*
- $\varepsilon_{init} := \mathcal{A}_{init}(\mathcal{L}_{Q,T})(\varepsilon)$ *is the canonical inclusion between the two coproducts;*

- $a_{init} := \mathcal{A}_{init}(\mathcal{L}_{Q,T})(a)$ *is obtained via the universal property as the coproduct over* $q \in Q$ *of the canonical injections* $j_{qa} \colon \mathcal{L}(\mathsf{in}) \to \coprod_{Q \cup QA} \mathcal{L}(\mathsf{in})$*;*
- $t\triangleleft_{init} := \mathcal{A}_{init}(\mathcal{L}_{Q,T})(t\triangleleft)$ *is obtained via the universal property as the coproduct over* $w \in Q \cup QA$ *of the morphims* $\mathcal{L}(\triangleright wt\triangleleft) \colon \mathcal{L}(\mathsf{in}) \to \mathcal{L}(\mathsf{out})$*.*

Dually, we can describe the final $\mathcal{C}_{Q,T}$-biautomaton as follows.

▶ **Lemma 19.** *Assume* $\mathcal{C}$ *has all countable powers of* $\mathcal{L}(\mathsf{out})$*. The final* $\mathcal{C}_{Q,T}$*-biautomaton is the functor* $\mathcal{A}_{final}(\mathcal{L}_{Q,T}) \colon \mathcal{I}_{Q,T} \to \mathcal{C}$ *described in the next diagram*

$$\begin{tikzcd}
\mathcal{L}(\mathsf{in}) \arrow[r, "\triangleright q_{final}"] & \prod\limits_{T \cup AT} \mathcal{L}(\mathsf{out}) \arrow[r, shift left, "a_{final}"] \arrow[r, shift right, "\varepsilon_{final}"'] & \prod\limits_{T} \mathcal{L}(\mathsf{out}) \arrow[r, "t\triangleleft_{final}"] & \mathcal{L}(\mathsf{out}),
\end{tikzcd}$$

where, explicitly:

- $\mathcal{A}_{final}(\mathcal{L}_{Q,T})(\mathsf{st}_1) = \prod\limits_{T \cup AT} \mathcal{L}(\mathsf{out})$ *and* $\mathcal{A}_{init}(\mathcal{L}_{Q,T})(\mathsf{st}_2) = \prod\limits_{T} \mathcal{L}(\mathsf{out})$*;*
- $\triangleright q_{final} := \mathcal{A}_{final}(\mathcal{L}_{Q,T})(\triangleright q)$ *is obtained via the universal property as the product over* $w \in T \cup AT$ *of the morphisms* $\mathcal{L}(\triangleright qw\triangleleft) \colon \mathcal{L}(\mathsf{in}) \to \mathcal{L}(\mathsf{out})$*;*
- $\varepsilon_{final} := \mathcal{A}_{final}(\mathcal{L}_{Q,T})(\varepsilon)$ *is the canonical restriction between the two products;*
- $a_{final} := \mathcal{A}_{final}(\mathcal{L}_{Q,T})(a)$ *is obtained via the universal property of* $\prod\limits_{T} \mathcal{L}(\mathsf{out})$ *as the product over* $t \in T$ *of the canonical projections* $\pi_{at} \colon \prod\limits_{T \cup AT} \mathcal{L}(\mathsf{out}) \to \mathcal{L}(\mathsf{out})$*;*
- $t\triangleleft_{final} := \mathcal{A}_{final}(\mathcal{L}_{Q,T})(t\triangleleft)$ *is the projection* $\pi_t \colon \prod\limits_{T} \mathcal{L}(\mathsf{out}) \to \mathcal{L}(\mathsf{out})$*.*

Combining Corollary 7 with Lemmas 8, 18 and 19, we obtain the minimal biautomaton $\mathsf{Min}(\mathcal{L}_{Q,T})$ in $\mathsf{Auto}(\mathcal{L}_{Q,T})$.

▶ **Theorem 20.** *Assume that* $\mathcal{C}$ *is equipped with a factorization system* $(\mathcal{E}, \mathcal{M})$ *and has countable copowers of* $\mathcal{L}(\mathsf{in})$ *and countable powers of* $\mathcal{L}(\mathsf{out})$*. Then the minimal* $\mathcal{C}_{Q,T}$*-biautomaton* $\mathsf{Min}(\mathcal{L}_{Q,T})$ *is obtained as the unique up to isomorphism factorization of the unique morphism from* $\mathcal{A}_{init}(\mathcal{L}_{Q,T})$ *to* $\mathcal{A}_{final}(\mathcal{L}_{Q,T})$*.*

$$\begin{tikzcd}
 & \coprod\limits_{Q} \mathcal{L}(\mathsf{in}) \arrow[r, shift left, "a_{init}"] \arrow[r, shift right, "\varepsilon_{init}"'] \arrow[d, two heads, "e^1_{min}"] & \coprod\limits_{Q \cup QA} \mathcal{L}(\mathsf{in}) \arrow[d, two heads, "e^2_{min}"] \arrow[rd, bend left, "t\triangleleft_{init}"] & \\
\mathcal{L}(\mathsf{in}) \arrow[ru, bend left, "\triangleright q_{init}"] \arrow[r, "\triangleright q_{min}"] \arrow[rd, bend right, "\triangleright q_{final}"'] & \mathsf{Min}(\mathcal{L}_{Q,T})(\mathsf{st}_1) \arrow[r, shift left, "a_{min}"] \arrow[r, shift right, "\varepsilon_{min}"'] \arrow[d, tail, "m^1_{min}"] & \mathsf{Min}(\mathcal{L}_{Q,T})(\mathsf{st}_2) \arrow[r, "t\triangleleft_{min}"] \arrow[d, tail, "m^2_{min}"] & \mathcal{L}(\mathsf{out}) \\
 & \prod\limits_{T \cup AT} \mathcal{L}(\mathsf{out}) \arrow[r, shift left, "a_{final}"] \arrow[r, shift right, "\varepsilon_{final}"'] & \prod\limits_{T} \mathcal{L}(\mathsf{out}) \arrow[ru, bend right, "t\triangleleft_{final}"'] &
\end{tikzcd}$$

Notice that the arrows $\triangleright q_{min}$, a_{min}, ε_{min} and $t\triangleleft_{min}$ are obtained using the diagonal fill-in property of the factorization system. Let us now see how this theorem instantiates in the case of deterministic automata.

▶ **Example 21.** Assume the target language $\mathcal{L}$ is a $(\mathsf{Set}, 1, 2)$-language, so the learner wants to learn the minimal deterministic automaton accepting $\mathcal{L}$. For a given couple (Q, T), the minimal biautomaton is obtained as the following factorization.

$$\begin{tikzcd}
 & Q \arrow[r, shift left, "a_{init}"] \arrow[r, shift right, "\varepsilon_{init}"'] \arrow[d, two heads, "e^1_{min}"] & Q \cup QA \arrow[rd, bend left, "t\triangleleft_{init}"] \arrow[d, two heads, "e^2_{min}"] & \\
1 \arrow[ru, bend left, "\triangleright q_{init}"] \arrow[r, "\triangleright q_{min}"] \arrow[rd, bend right, "\triangleright q_{final}"'] & Q/{\sim_{T \cup AT}} \arrow[r, shift left, "a_{min}"] \arrow[r, shift right, "\varepsilon_{min}"'] \arrow[d, tail, "m^1_{min}"] & (Q \cup QA)/{\sim_T} \arrow[r, "t\triangleleft_{min}"] \arrow[d, tail, "m^2_{min}"] & 2 \\
 & 2^{T \cup AT} \arrow[r, shift left, "a_{final}"] \arrow[r, shift right, "\varepsilon_{final}"'] & 2^T \arrow[ru, bend right, "t\triangleleft_{final}"'] &
\end{tikzcd}$$

Hence, the first set of states of the minimal biautomaton is the set Q quotiented by the $T \cup AT$-approximation $\sim_{T \cup AT}$ of the Myhill-Nerode equivalence. The second set of states is the quotient of $Q \cup QA$ by $\sim_T$. These kinds of quotients are also needed in the classical Angluin's algorithm, when building the table corresponding to the couple (Q, T).

Let us now understand when the map ε_{min} is an isomorphism, that is, in this case, a bijection. We can verify that ε_{min} being a surjection is equivalent to the table in $\mathtt{L}^*$-algorithm being closed, that is, for all $q \in Q$ and $a \in A$ there exists $q' \in Q$ such that $q' \sim_T qa$. On the other hand, ε_{min} being an injection is equivalent to the consistency of the table from Angluin's $\mathtt{L}^*$-algorithm. It means that if q and q' are such that $q \sim_T q'$ then $q \sim_{T \cup AT} q'$.

If the table from Angluin's algorithm is generalized via the minimal biautomaton $\mathsf{Min}(\mathcal{L}_{Q,T})$, the above example suggests that the conditions that make possible the generation of a hypothesis automaton from a table can be stated at this abstract level by requiring the morphism ε_{min} be an isomorphism. In this way, we can identify $\mathsf{Min}(\mathcal{L})(\mathsf{st}_1)$ and $\mathsf{Min}(\mathcal{L})(\mathsf{st}_2)$ to obtain the state space of the hypothesis automaton.

▶ **Definition 22.** *If the map ε_{min} is an isomorphism, we say that (Q, T) is $\mathcal{L}$-*automatable. *The* hypothesis automaton $\mathcal{H}(Q, T)$ *associated to a $\mathcal{L}$-automatable couple (Q, T) is the $\mathcal{C}$-automaton with state space $\mathsf{Min}(\mathcal{L}_{Q,T})(\mathsf{st}_1)$ described on the generator arrows of $\mathcal{I}_{A^*}$ by*

$$\begin{tikzcd}
\mathcal{L}(\mathsf{in}) \arrow[r, "\triangleright\varepsilon_{min}"] & \mathsf{Min}(\mathcal{L}_{Q,T})(\mathsf{st}_1) \arrow[loop above, "{\varepsilon_{min}}^{-1} \circ a_{min}"] \arrow[r, "\varepsilon\triangleleft_{min} \circ \varepsilon_{min}"] & \mathcal{L}(\mathsf{out}).
\end{tikzcd}$$

The uniqueness up to isomorphism of the hypothesis automaton $\mathcal{H}(Q, T)$ is an easy consequence of the uniqueness up to isomorphism of the minimal biautomaton in $\mathsf{Auto}(\mathcal{L}_{Q,T})$.

It is important to remark that, when passing from a biautomaton to an automaton, the consistency with the language is preserved, in the sense of the lemma below.

▶ **Lemma 23.** *Let (Q, T) be an $\mathcal{L}$-automatable couple and let $\mathcal{H}(Q, T)$ be its associated hypothesis automaton. Then the next diagram commutes:*

$$\begin{tikzcd}
\mathcal{O}_{Q,T} \arrow[r, hook] \arrow[rr, bend left, "\mathcal{L}_{Q,T}"] & \mathcal{I}_{A^*} \arrow[r, "\mathcal{H}(Q,T)"'] & \mathcal{C}.
\end{tikzcd}$$

3.2 The learning algorithm

We now have all the necessary ingredients to state the $\mathtt{FunL}^*$-algorithm. Our algorithm takes as input a target language $\mathcal{L}$ and outputs its minimal automaton, provided some mild assumptions listed in Theorem 26 are satisfied. We start by instantianting the couple (Q, T) by $(\varepsilon, \varepsilon)$. As long as this couple is not $\mathcal{L}$-automatable, further words are added to the subsets Q and T to force ε_{min} to become an isomorphism. Once this is achieved, we obtain a hypothesis automaton. If this automaton does not recognize the target language, then the provided counterexample and its prefixes are added to Q, in order to let the learner progress in learning.

While the role played by equivalence queries is self-evident, notice that evaluation queries are necessary in order to build up the category $\mathsf{Auto}(\mathcal{L}_{Q,T})$ and analyse its minimal automaton.

Algorithm 1 The basic FunL*learning algorithm.

```
input  : minimally adequate teacher of the target language $\mathcal{L}$
output : $\mathsf{Min}(\mathcal{L})$
$Q := T := \{\varepsilon\}$
repeat
    while $(Q,T)$ is not $\mathcal{L}$-automatable do
        if $\varepsilon_{min} \notin \mathcal{E}$ then
            add $QA$ to $Q$
        end
        if $\varepsilon_{min} \notin \mathcal{M}$ then
            add $AT$ to $T$
        end
    end
    ask an equivalence query for the hypothesis automaton $\mathcal{H}(Q,T)$
    if the answer is no then
        add the provided counterexample and all its prefixes to $Q$
    end
until the answer is yes;
return $\mathcal{H}(Q,T)$
```

In order for this generic algorithm to work, we need several mild assumptions on the output category $\mathcal{C}$ and on the target language. First, in order to compute the hypothesis automaton we need the existence of the minimal automaton in the category $\mathsf{Auto}(\mathcal{L}_{Q,T})$. For this reason, we will assume the hypothesis of Theorem 20, pertaining to the existence of certain powers, certain copowers and a factorization system. Furthermore, in order to ensure the termination of our algorithm, a noetherianity condition is required on the language $\mathcal{L}$, akin to the regularity of the language in the L*-algorithm. This notion, also used in [21], can be understood as a finiteness assumption as shown in Example 25.

▶ **Definition 24.** *An object X of $\mathcal{C}$ is called $(\mathcal{E}, \mathcal{M})$-*noetherian *when the following conditions hold.*

- *There does not exist an infinite co-chain of $\mathcal{E}$-quotients of X as in the left commutative diagram below and such that the arrows $e_1, e_2 \ldots \in \mathcal{E}$ are not isomorphisms.*
- *There does not exist an infinite chain of $\mathcal{M}$-subobjects of X as in the right commutative diagram below and such that the arrows $m_1, m_2 \ldots \in \mathcal{M}$ are not isomorphisms.*

$$\begin{tikzcd} & & X \arrow[ddll, bend right, two heads] \arrow[ddl, bend right, two heads] \\ & & \cdots \\ \cdot & \cdot \arrow[l, two heads, "e_1"'] & \cdots \arrow[l, two heads, "e_2"'] \end{tikzcd} \qquad \begin{tikzcd} & & X \\ & & \cdots \\ \cdot \arrow[r, tail, "m_1"] \arrow[uurr, bend left, tail] & \cdot \arrow[r, tail, "m_2"] \arrow[uur, bend left, tail] & \cdots \end{tikzcd}$$

▶ **Example 25.** Let us see now what noetherianity means for the factorization systems of our running instantiations (Example 9). It is easy to see that in Set, an object X is $(\text{Surjections}, \text{Injections})$-noetherian if and only if it is finite in the usual sense. Similarly, an object of Vec is (Surjective linear maps, Injective linear maps)-noetherian if and only if it is a finite dimension vector space. With a bit more thoughts, one can establish that an object X of $\mathsf{Kl}(\mathcal{T})$ is $(\mathcal{E}_{\mathsf{Kl}(\mathcal{T})}, \mathcal{M}_{\mathsf{Kl}(\mathcal{T})})$-noetherian if and only if it is finite.

In order to guarantee the termination of our algorithm, we require the $(\mathcal{E}, \mathcal{M})$-noetherianity of the state space of the minimal automaton of the target language. This is a natural condition, generalizing the regularity of the target language in the $\mathtt{L}^*$-algorithm. If $(\mathcal{E}, \mathcal{M})$-noetherian objects are closed under $\mathcal{E}$-quotients and $\mathcal{M}$-subobjects – as it is the case in all our examples – we could also replace this hypothesis by assuming the existence of an automaton with $(\mathcal{E}, \mathcal{M})$-noetherian state space which accepts the target language.

▶ **Theorem 26.** *We consider a target language $\mathcal{L}\colon \mathcal{O}_{A^*} \to \mathcal{C}$ such that:*

- *the output category $\mathcal{C}$ is endowed with a factorization system $(\mathcal{E}, \mathcal{M})$;*
- *$\mathcal{C}$ has all copowers of $\mathcal{L}(\mathsf{in})$ and all powers of $\mathcal{L}(\mathsf{out})$;*
- *the state space $\mathsf{Min}(\mathcal{L})(\mathsf{st})$ of the minimal automaton for $\mathcal{L}$ is $(\mathcal{E}, \mathcal{M})$-noetherian.*

Then the $\mathtt{FunL}^*$-algorithm terminates, eventually producing the minimal automaton $\mathsf{Min}(\mathcal{L})$ accepting the target language.

The proof of this theorem relies on a careful analysis of the factorizations

$$\begin{tikzcd} \coprod_Q \mathcal{L}(\mathsf{in}) \arrow[r, two heads] & \Im_{Q,T} \arrow[r, tail] & \prod_T \mathcal{L}(\mathsf{out}) \end{tikzcd}$$

of the canonical maps $\coprod_Q \mathcal{L}(\mathsf{in}) \to \prod_T \mathcal{L}(\mathsf{out})$ obtained by taking the coproduct over $q \in Q$ of the product over $t \in T$ of $\mathcal{L}(\triangleright qt \triangleleft)$. We can prove that the state spaces of the biautomata featured while running the algorithm are precisely of the form $\Im_{Q,T}$, while the state space of the minimal automaton accepting $\mathcal{L}$ is $\Im_{A^*,A^*}$.

We prove that the **while** loop terminates in [14, Proposition 35]. In [14, Lemma 36] we show that only finitely many counterexamples can be added, hence the algorithm terminates. Finally, the fact that the outcome automaton is minimal is shown in [14, Lemma 38].

Next, we see how the $\mathtt{FunL}^*$-algorithm instantiates to the case of subsequential transducers. We need to understand what it means for ε_{min} to be an isomorphism.

▶ **Example 27** (Learning algorithm for subsequential transducers)**.** Assume the target language $\mathcal{L}$ is a $(\mathsf{Kl}(\mathcal{T}), 1, 1)$-language, so the learner wants to learn the minimal subsequential transducer accepting $\mathcal{L}$. We need to extend the notions of lcp and red to a generic partial map g whose domain is $T \subseteq A^*$ and whose codomain is B^* as follows: $\mathsf{lcp}(g)$ is undefined if g is nowhere defined, and denotes the longest common prefix of the words in $\{g(u) \mid u \in T\}$ otherwise; analogously, $\mathsf{red}(g)\colon T \to B^* \cup \{\bot\}$ is nowhere defined if g is nowhere defined, and is the only partial map such that $g(u) = \mathsf{lcp}(g)\mathsf{red}(g)(u)$ otherwise. Thinking of the language to learn as a transduction $f\colon A^* \to B^* + \{\bot\}$, let's define the following equivalence relation for all $q_1, q_2 \in Q$: $q_1 \sim_T q_2$ if and only if $\mathsf{red}(f(q_1-)|_T)(t) = \mathsf{red}(f(q_2-)|_T)(t)$ for all $t \in T$, $f(q-)|_T$ being the restriction of $f(q-)$ to T. For a couple (Q, T), ε_{min} in $\mathsf{Auto}(\mathcal{L}_{Q,T})$ turns out to be the map $Q/{\sim_{T \cup AT}} \twoheadrightarrow (Q \cup QA)/{\sim_T}, [q] \mapsto (\mathsf{lcp}(f(q-)|_{T\cup AT})^{-1}\mathsf{lcp}(f(q-)|_T), [q])$, the first set of states being Q quotiented by $\sim_{T\cup AT}$, the second set of states being the quotient of $Q \cup QA$ by $\sim_T$. Let's understand the word a class $[q]$ is mapped to: with $\mathsf{lcp}(f(q-)|_T)$, we mean the lcp of the function $f(q-)$ restricted to the domain T, that is, the longest common prefix of the subset $\{f(qt)|t \in T\}$; with $\mathsf{lcp}(f(q-)|_{T\cup AT})^{-1}\mathsf{lcp}(f(q-)|_T)$, we mean the word $\mathsf{lcp}(f(q-)|_T)$ from which $\mathsf{lcp}(f(q-)|_{T\cup AT})$ (one of its prefixes, as it is the longest common prefix of a bigger set of words) has been stripped away; when one of the lcps is undefined, $[q]$ is mapped to undefined.

ε_{min} is an isomorphism if and only if $\pi_2(\varepsilon_{min})$ is a bijection and $\pi_1(\varepsilon_{min})$ is the constant function mapping every $x \in X$ to ε. We can verify that $\pi_2(\varepsilon_{min})$ being a surjection is equivalent to the condition that for all $q \in Q$ and $a \in A$ there exists $q' \in Q$ such that $q' \sim_T qa$, whereas $\pi_2(\varepsilon_{min})$ being an injection is equivalent to the condition that if q and q' are such that $q \sim_T q'$, then $q \sim_{T \cup AT} q'$. Finally, the condition on $\pi_1(\varepsilon_{min})$ is true if and only if $\mathsf{lcp}(f(q-)|_{T \cup AT}) = \mathsf{lcp}(f(q-)|_T)$. Remarkably, these three naturally arising conditions turn out to be equivalent to the ones required in Vilar's learning algorithm for subsequential transducers (see [24]).

Every time the while cycle runs, our algorithm adds either all words QA to Q or all words AT to T: this is not strictly necessary. We show next that it is sufficient to add just one properly chosen single word $qa \in QA$ or $at \in AT$, preserving the correctness of the algorithm. The canonical inclusion $\coprod_Q \mathcal{L}(\mathsf{in}) \to \coprod_{Q \cup \{qa\}} \mathcal{L}(\mathsf{in})$ induces a canonical morphism between the factorizations $\mathfrak{F}_{Q,T} \rightarrowtail \mathfrak{F}_{Q \cup \{qa\},T}$. Similarly, the canonical restriction $\prod_{T \cup \{at\}} \mathcal{L}(\mathsf{out}) \to \prod_T \mathcal{L}(\mathsf{out})$ induces a canonical morphism between the factorizations $\mathfrak{F}_{Q,T} \twoheadleftarrow \mathfrak{F}_{Q,T \cup \{at\}}$, which will be featured in the optimized algorithm.

▶ **Theorem 28.** *Algorithm 1 can be optimized by replacing lines 5 and 8 respectively by:*

- add to Q a $qa \in QA$ s.t. $\mathfrak{F}_{Q,T} \rightarrowtail \mathfrak{F}_{Q \cup \{qa\},T}$ is not an isomorphism*;*
- add to T an $at \in AT$ s.t. $\mathfrak{F}_{Q,T} \twoheadleftarrow \mathfrak{F}_{Q,T \cup \{at\}}$ is not an isomorphism.

4 Conclusion and future work

In this paper, we described the abstract algorithm $\mathtt{FunL}^*$, a categorical version of Angluin's $\mathtt{L}^*$-algorithm for learning word automata. The focus was on providing a minimalistic category theoretic framework for learning, with as few assumptions as possible, emphasizing along the way the deep connection between learning and minimization.

So far, $\mathtt{FunL}^*$ does not cover instances of the $\mathtt{L}^*$-like algorithms such as nominal automata, or automata/transducers over trees. A natural continuation is to develop these generalizations. Another aspect to understand abstractly is the complexity of this algorithm in terms of number of evaluation and equivalence queries.

References

1 Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput., 75, 87-106*, 1987.

2 Dana Angluin, Sarah Eisenstat, and Dana Fisman. Learning regular languages via alternating automata. In *IJCAI*, pages 3308–3314. AAAI Press, 2015.

3 Dana Angluin and Dana Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016. doi:10.1016/j.tcs.2016.07.031.

4 Simone Barlocco, Clemens Kupke, and Jurriaan Rot. Coalgebra learning via duality. In *FoSSaCS*, volume 11425 of *Lecture Notes in Computer Science*, pages 62–79. Springer, 2019.

5 Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. In *CIAC*, volume 778 of *Lecture Notes in Computer Science*, pages 54–62. Springer, 1994.

6 Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM J. Comput.*, 25(6):1268–1280, 1996.

7 Adrien Boiret, Aurélien Lemay, and Joachim Niehren. Learning rational functions. In *Developments in Language Theory*, volume 7410 of *Lecture Notes in Computer Science*, pages 273–283. Springer, 2012.

8 Adrien Boiret, Aurélien Lemay, and Joachim Niehren. Learning top-down tree transducers with regular domain inspection. In *ICGI*, volume 57 of *JMLR Workshop and Conference Proceedings*, pages 54–65. JMLR.org, 2016.

9 Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In *IJCAI*, pages 1004–1009, 2009. URL: http://ijcai.org/Proceedings/09/Papers/170.pdf.

10 Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker, Daniel Neider, and David R. Piegdon. libalf: The automata learning framework. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 360–364, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

11 Christian Choffrut. A generalization of Ginsburg and Rose's characterization of G-S-M mappings. In *ICALP*, volume 71 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 1979.

12 Thomas Colcombet and Daniela Petrişan. Automata minimization: a functorial approach. In *7th Conference on Algebra and Coalgebra in Computer Science, CALCO 2017, June 12-16, 2017, Ljubljana, Slovenia*, pages 8:1–8:16, 2017. doi:10.4230/LIPIcs.CALCO.2017.8.

13 Thomas Colcombet and Daniela Petrişan. Automata minimization: a functorial approach. *Logical Methods in Computer Science*, Volume 16, Issue 1, 2020. URL: https://lmcs.episciences.org/6213.

14 Thomas Colcombet, Daniela Petrişan, and Riccardo Stabile. Learning automata and transducers: a categorical approach. *CoRR*, 2020. arXiv:2010.13675.

15 Samuel Drews and Loris D'Antoni. Learning symbolic automata. In *TACAS (1)*, volume 10205 of *Lecture Notes in Computer Science*, pages 173–189, 2017.

16 Helle Hvid Hansen. Subsequential transducers: a coalgebraic perspective. *Inf. Comput.*, 208(12):1368–1397, 2010.

17 Bart Jacobs and Alexandra Silva. Automata learning: A categorical perspective. In *Horizons of the Mind*, volume 8464 of *Lecture Notes in Computer Science*, pages 384–406. Springer, 2014.

18 Martin Leucker. Learning meets verification. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Formal Methods for Components and Objects*, pages 127–151, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

19 Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michal Szynwelski. Learning nominal automata. In *POPL*, pages 613–625. ACM, 2017. URL: http://dl.acm.org/citation.cfm?id=3009879.

20 M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.

21 Henning Urbat and Lutz Schröder. Automata learning: An algebraic approach. In *LICS*, pages 900–914. ACM, 2020.

22 Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. CALF: categorical automata learning framework. In *CSL*, volume 82 of *LIPIcs*, pages 29:1–29:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

23 Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. Learning automata with side-effects. In *CMCS*, 2020.

24 Juan Miguel Vilar. Query learning of subsequential transducers. In *Grammatical Inference: Learning Syntax from Sentences, 3rd International Colloquium, ICGI-96, Montpellier, France, September 25-27, 1996, Proceedings*, pages 72–83, 1996. doi:10.1007/BFb0033343.

25 Juan Miguel Vilar. Improve the learning of subsequential transducers by using alignments and dictionaries. In *Grammatical Inference: Algorithms and Applications, 5th International Colloquium, ICGI 2000, Lisbon, Portugal, September 11-13, 2000, Proceedings*, pages 298–311, 2000. doi:10.1007/978-3-540-45257-7_24.

Game Comonads & Generalised Quantifiers

Adam Ó Conghaile
Department of Computer Science and Technology, University of Cambridge, UK
ac891@cl.cam.ac.uk

Anuj Dawar
Department of Computer Science and Technology, University of Cambridge, UK
anuj.dawar@cl.cam.ac.uk

Abstract

Game comonads, introduced by Abramsky, Dawar and Wang and developed by Abramsky and Shah, give an interesting categorical semantics to some Spoiler-Duplicator games that are common in finite model theory. In particular they expose connections between one-sided and two-sided games, and parameters such as treewidth and treedepth and corresponding notions of decomposition. In the present paper, we expand the realm of game comonads to logics with generalised quantifiers. In particular, we introduce a comonad graded by two parameter $n \leq k$ such that isomorphisms in the resulting Kleisli category are exactly Duplicator winning strategies in Hella's n-bijection game with k pebbles. We define a one-sided version of this game which allows us to provide a categorical semantics for a number of logics with generalised quantifiers. We also give a novel notion of tree decomposition that emerges from the construction.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Abstraction

Keywords and phrases Logic, Finite Model Theory, Game Comonads, Generalised Quantifiers

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.16

Related Version A full version of the paper is available at `https://www.arxiv.org/abs/2006.16039`.

Funding Research funded in part by EPSRC grant EP/T007257/1.

1 Introduction

Model-comparison games, such as Ehrenfeucht-Fraïssé games and pebble games, play a central role in finite model theory. Recent work by Abramsky et al. [3, 4] provides a category-theoretic view of such games which yields new insights. In particular, the *pebbling comonad* $\mathbb{P}_k$ introduced in [3] reveals an interesting relationship between one-sided and two-sided pebble games. The morphisms in the Kleisli category associated with $\mathbb{P}_k$ correspond exactly to winning strategies in the existential positive k-pebble game. This game was introduced by Kolaitis and Vardi [18] to study the expressive power of `Datalog`. A winning strategy for Duplicator in the game played on structures $\mathcal{A}$ and $\mathcal{B}$ implies that all formulas of existential positive k-variable logic true in $\mathcal{A}$ are also true in $\mathcal{B}$. The game has found widespread application in the study of database query languages as well as constraint satisfaction problems. Indeed, the widely used k-local consistency algorithms for solving constraint satisfaction can be understood as computing the approximation to homomorphism given by such strategies [19]. At the same time, isomorphisms in the Kleisli category associated with $\mathbb{P}_k$ correspond to winning strategies in the k-pebble *bijection* game. This game, introduced by Hella [16], characterises equivalence in the k-variable logic with counting. This gives a family of equivalence relations (parameterised by k) which has been widely studied as approximations of graph isomorphism. It is often called the Weisfeiler-Leman family of equivalences and has a number of characterisations in logic, algebra and combinatorics (see the discussion in [13]).

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 16; pp. 16:1–16:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The bijection game introduced by Hella is actually the initial level of a hierarchy of games that he defined to characterise equivalence in logics with generalised (i.e. Lindström) quantifiers. For each $n, k \in \mathbb{N}$ we have a k-pebble n-bijection game that characterises equivalence with respect to an infinitary k-variable logic with quantifiers of arity at most n. In the present paper, we introduce a graded comonad associated with this game. Our comonad $\mathbb{G}_{n,k}$ is obtained as a quotient of the comonad $\mathbb{P}_k$ and we are able to show that isomorphisms in the associated Kleisli category correspond to winning strategies for Duplicator in the k-pebble n-bijection game. The morphisms then correspond to a new one-way game we define, which we call the k-pebble n-function game. We are able to show that this relates to a natural logic: a k-variable positive infinitary logic with n-ary *homomorphism-closed quantifiers*.

This leads us to a systematic eight-way classification of model-comparison games based on what kinds of functions Duplicator is permitted (arbitrary functions, injections, surjections or bijections) and what the partial maps in game positions are required to preserve: just atomic information or also negated atoms. We show that each of these variations correspond to preservation of formulas in a natural fragment of bounded-variable infinitary logic with n-ary Lindström quantifiers. Moreover, winning strategies in these games also correspond to natural restrictions of the morphisms in the Kleisli category of $\mathbb{G}_{n,k}$ that are well-motivated from the category-theoretic point of view.

Another key insight provided by the work of Abramsky et al. is that coalgebras in the pebbling comonad $\mathbb{P}_k$ correspond exactly to tree decompositions of width k. Similarly, the coalgebras in the Ehrenfeucht-Fraïssé comonad introduced by Abramksy and Shah characterise the *treedepth* of structures. This motivates us to look at coalgebras in $\mathbb{G}_{n,k}$ and we show that the yield a new and natural notion of generalised tree decomposition.

In what follows, after a review of the necessary background in Section 2, we introduce the various games and logics in Section 3 and establish the relationships between them. Section 4 contains the definition of the comonad $\mathbb{G}_{n,k}$ and shows that interesting classes of morphisms in the associated Kleisli category correspond to winning strategies in the games. Section 5 defines a new class of extended tree decompositions and traversals and relates them to the coalgebras of the comonad $\mathbb{G}_{n,k}$. Proofs are omitted due to lack of space and may be found in the appendix.

2 Background

In this section we introduce notation that we use throughout the paper and give a brief overview of background we assume.

For a positive integer n, we write $[n]$ for the set $\{1, \ldots, n\}$.

A tree T is a set with a partial order $\leq$ such that for all $t \in T$, the set $\{x \mid x \leq t\}$ is linearly ordered by $\leq$ and such that there is an element $r \in T$ called *the root* such that $r \leq t$ for all $t \in T$. If $t < t'$ in T and there is no x with $t < x < t'$, we call t' a *child* of t and t the *parent* of t'.

For X a set, we write X^* for the set of *lists* over elements of X and X^+ for the set of non-empty lists. We write the list with elements $x_1, \ldots x_m$ in that order as $[x_1, \ldots x_m]$. For two lists $s_1, s_2 \in X^*$ we write $s_1 \cdot s_2$ for the list formed by concatenating s_1 and s_2. For $x \in X$ and $s \in X^*$ we write $x; s$ for the list $[x] \cdot s$ and $s; x$ for the list $s \cdot [x]$. We occasionally underline the fact that $s_1 \cdot s_2, x; s$, and $s; x$ are lists by writing them enclosed in square brackets, as $[s_1 \cdot s_2]$, $[x; s]$, and $[s; x]$.

2.1 Logics

We work with finite relational signatures and assume a fixed signature σ. Unless stated otherwise, the structures we consider are finite σ-structures. We write $\mathcal{A}, \mathcal{B}, \mathcal{C}$ etc. to denote such structures, and the corresponding roman letters A, B, C etc. to denote their universes.

We assume a standard syntax and semantics for first-order logic (as in [20]), which we denote **FO**. We write $\mathcal{L}_\infty$ for the infinitary logic that is obtained from **FO** by allowing conjunctions and disjunctions over arbitrary sets of formulas. We write $\exists^+\mathcal{L}_\infty$ and $\exists^+\mathbf{FO}$ for the restriction of $\mathcal{L}_\infty$ and **FO** to existential positive formulas, i.e. those without negations or universal quantifiers. We use natural number superscripts to denote restrictions of the logic to a fixed number of variables. We write $\mathcal{C}$ to denote the extension of $\mathcal{L}_\infty$ with counting quantifiers. We are mainly interested in the k-variable fragments of this logic $\mathcal{C}^k$.

2.2 Generalised quantifiers

We use the term *generalised quantifer* in the sense of Lindström [21]. These have been extensively studied in finite model theory (see [16, 8, 5]). In what follows, we give a brief account of the basic variant that is of interest to us here. For more on Lindström quantifiers, consult [11, Chap. 12]. We only consider quantifiers without relativisation, vectorisation or taking quotients in the interpretation.

Any isomorphism-closed class of structures K over a signature τ gives rise to a *generalised quantifier* Q_K. For a logic L, we write $L(Q_K)$ for its extension with the quantifier Q_K. We define the *arity* of the quantifier Q_K to be the maximum arity of any relation in τ. For Q_K with arity m, the formula $Q_K x_1, \dots x_m.(\psi_R(x_1^R, \dots x_l^R, \mathbf{z}^R))_{R\in\tau}$ where $x_i^R \in \mathbf{x}$ and $\mathbf{z}^R \subset \mathbf{z}$ is true on $\mathcal{A}, \mathbf{a}$ if the τ-structure $\langle A, (\psi_R(\cdot, \mathbf{a}^R))_{R\in\tau}\rangle$ is in K. We write $\mathcal{L}^k_\infty(\mathcal{Q}_n)$ for the extension of $\mathcal{L}^k_\infty$ with *all* quantifiers of arity n. This is only of interest when $n \le k$. Kolaitis and Väänänen [17] showed that $\mathcal{L}^k_\infty(\mathcal{Q}_1)$ is equivalent to $\mathcal{C}^k$. However, allowing quantifiers of higher arity gives logics of considerably more expressive power. In particular, if σ is a signature with all relations of arity at most n, then *any* property of σ-structures is expressible in $\mathcal{L}^n_\infty(\mathcal{Q}_n)$. Thus, all properties of graphs, for instance, are expressible in $\mathcal{L}^2_\infty(\mathcal{Q}_2)$.

2.3 Games

For a pair of structures $\mathcal{A}$ and $\mathcal{B}$ and a logic L, we write $\mathcal{A} \Rrightarrow_L \mathcal{B}$ to denote that every sentence of L that is true in $\mathcal{A}$ is also true in $\mathcal{B}$. When the logic is closed under negation, as is the case with **FO** and $\mathcal{L}_\infty$, for instance, $\mathcal{A} \Rrightarrow_L \mathcal{B}$ implies $\mathcal{B} \Rrightarrow_L \mathcal{A}$. In this case, we have an equivalence relation between structures and we write $\mathcal{A} \equiv_L \mathcal{B}$. When $\mathcal{A}$ and $\mathcal{B}$ are finite structures, $\mathcal{A} \Rrightarrow_{\mathbf{FO}} \mathcal{B}$ implies $\mathcal{A} \Rrightarrow_{\mathcal{L}_\infty} \mathcal{B}$, and the same holds for the k-variable fragments of these logics (see [10]).

The relations $\Rrightarrow_L$ are often characterised in terms of games which we generically call *Spoiler-Duplicator* games. For instance, the existential-positive k-pebble game introduced by Kolaitis and Vardi [18], which we denote $\exists\mathbf{Peb}_k$, characterises the relation $\Rrightarrow_{\exists^+\mathcal{L}^k_\infty}$. In this game, Spoiler and Duplicator each has a collection of k pebbles indexed $1, \dots, k$. In each round Spoiler places one of its pebbles on an element of $\mathcal{A}$ and Duplicator responds by placing its corresponding pebble (i.e. the one of the same index) on an element of $\mathcal{B}$. Note the game can go on for more than k rounds and pebbles can be repositioned throughout. If the partial map taking the element of $\mathcal{A}$ on which Spoiler's pebble i sits to the element of $\mathcal{B}$ on which Duplicator's pebble i is, fails to be a partial homomorphism, then Spoiler has won the game. Duplicator wins by playing forever without losing. We get a game characterising

$\equiv_{\mathcal{L}^k_\infty}$ if *(i)* Spoiler is allowed to choose, at each move, on which of the two structures it places a pebble and Duplicator is required to respond in the other structure; and *(ii)* Duplicator is required to ensure that the pebbled positions form a partial *isomorphism.*

Hella [16] introduced a *bijection game* which characterises the equivalence $\equiv_{\mathcal{C}^k}$. We write $\mathbf{Bij}_k(\mathcal{A}, \mathcal{B})$ for the bijection game played on $\mathcal{A}$ and $\mathcal{B}$. At each move, Spoiler chooses an index $i \in [k]$ and Duplicator is required to respond with a bijection $f : A \to B$. Spoiler then chooses an element $a \in A$ and pebbles indexed i are placed on a and $f(a)$. If the partial map defined by the pebbled positions is not a partial isomorphism, then Spoiler has won. Duplicator wins by playing forever without losing.

In Hella's original work, the bijection games appear as a special case of the *n-bijective k-pebble game*, which we denote $\mathbf{Bij}^k_n(\mathcal{A}, \mathcal{B})$ when played on structures $\mathcal{A}$ and $\mathcal{B}$. This characterises the equivalence relation $\equiv_{\mathcal{L}^k_\infty(\mathcal{Q}_n)}$. Once again, we have a set of k pebbles associated with each of the structures $\mathcal{A}$ and $\mathcal{B}$ and indexed by $[k]$. At each move, Duplicator is required to give a bijection $f : A \to B$ and Spoiler chooses a set of up to n pebble indices $p_1, \dots, p_n \in [k]$ and moves the corresponding indices to elements $a_1, \dots, a_n \in A$ and $f(a_1), \dots, f(a_n)$ in B. If the partial map defined by the pebbled positions is not a partial isomorphism, then Spoiler has won. Duplicator wins by playing forever without losing. Note, in particular, that for Duplicator to have a winning strategy it is necessary that the reducts of $\mathcal{A}$ and $\mathcal{B}$ to relations of arity at most n are isomorphic. For example, on graphs Spoiler wins any game on non-isomorphic graphs with $n, k \geq 2$.

2.4 Comonads

We assume that the reader is familiar with basic definitions from category theory, in particular the notions of category, functor and natural transformation. For a finite signature σ, we are interested in the category $\mathcal{R}(\sigma)$ of relational structures over σ. The objects of the category are such structures and the maps are homomorphisms between structures.

Comonads on a category $\mathcal{C}$ are triples (T, ϵ, δ) where T is an endofunctor on $\mathcal{C}$ and ϵ and δ are natural transformations of type $T \to 1$ and $TT \to T$ respectively, satisfying certain comonad laws. The Kleisli category $\mathcal{K}(T)$ is the category with the same objects as $\mathcal{C}$ where $\mathcal{K}(T)$-morphisms are $\mathcal{C}$-morphisms of type $TA \to B$. Composition is defined with the help of δ. A T-coalgebra on an object $A \in \mathcal{C}$ is a $\mathcal{C}$-map $\alpha : A \to TA$ such that $\epsilon_A \circ \alpha = 1_A$.

Abramsky et al. [3] describe the construction of a comonad $\mathbb{P}_k$, graded by k, on the category $\mathcal{R}(\sigma)$ which exposes an interesting relationship between the games $\exists\mathbf{Peb}_k(\mathcal{A}, \mathcal{B})$ and $\mathbf{Bij}_k(\mathcal{A}, \mathcal{B})$. Specifically, this construction shows that Duplicator winning strategies in the latter are exactly the isomorphisms in a category in which the morphisms are winning strategies in the former.

For any $\mathcal{A}$, $\mathbb{P}_k\mathcal{A}$ is an infinite structure (even when $\mathcal{A}$ is finite) with universe $(A \times [k])^+$. The counit $\epsilon_\mathcal{A}$ takes a sequence $[(a_1, p_1), \dots, (a_m, p_m)]$ to a_m, i.e. the first component of the last element of the sequence. The comultiplication $\delta_\mathcal{A}$ takes a sequence $[(a_1, p_1), \dots, (a_m, p_m)]$ to the sequence $[(s_1, p_1), \dots, (s_m, p_m)]$ where $s_i = [(a_1, p_1), \dots, (a_i, p_i)]$. The relations are defined so that $(s_1, \dots, s_r) \in R^{\mathbb{P}_k\mathcal{A}}$ if, and only if, the s_i are all comparable in the prefix order of sequences, $R^\mathcal{A}(\epsilon_\mathcal{A}(s_1), \dots, \epsilon_\mathcal{A}(s_r))$ and whenever s_i is a prefix of s_j and ends with the pair (a, p), there is no prefix of s_j properly extending s_i which ends with (a', p) for any $a' \in A$.

It is convenient to consider structures over a signature $\sigma \cup \{I\}$ where I is a new binary relation symbol. An I-structure is a structure over this signature which interprets I as the identity relation. Note that even when $\mathcal{A}$ is an I-structure, $\mathbb{P}_k\mathcal{A}$ is not one. The key results from [3] relating the comonad with pebble games can now be stated as establishing a precise translation between *(i)* $\mathcal{K}(\mathbb{P}_k)$-morphisms $\mathcal{A} \longrightarrow \mathcal{B}$ for I-structures $\mathcal{A}$ and $\mathcal{B}$; and *(ii)* winning strategies for Duplicator in $\exists\mathbf{Peb}_k(\mathcal{A},\mathcal{B})$; and similarly a precise translation between *(i)* isomorphisms in $\mathcal{K}(\mathbb{P}_k)$ between $\mathcal{A}$ and $\mathcal{B}$ for I-structures $\mathcal{A}$ and $\mathcal{B}$; and *(ii)* winning strategies for Duplicator in $\mathbf{Bij}_k(\mathcal{A},\mathcal{B})$.

A key result from the construction of the comonad $\mathbb{P}_k$ is the relationship between the coalgebras of this comonad and tree decompositions. In particular, it is shown that a structure $\mathcal{A}$ has a $\mathbb{P}_k$-coalgebra if, and only if, the treewidth of $\mathcal{A}$ is at most $k-1$. This relationship between coalgebras and tree decompositions is established through a definition of a *tree traversal* which we review in Section 5 below.

3 Games and Logic with Generalised Quantifers

Hella's n-bijective k-pebble game, $\mathbf{Bij}^k_n$ is a model-comparison game which captures equivalence of structures over the logic $\mathcal{L}^k_\infty(\mathcal{Q}_n)$, i.e. k-variable infinitary logic where the allowed quantifiers are all generalised quantifiers with arity $\leq n$. This game generalises the bijection game $\mathbf{Bij}_k$ which captures equivalence over $\mathcal{C}^k$, k-variable infinitary logic with counting quantifiers (which is equivalent to $\mathcal{L}^k_\infty(\mathcal{Q}_1)$ as shown by Kolaitis and Väänänen[17]). In this section, we introduce a family of games which relax the rules of $\mathbf{Bij}^k_n$ and show their correspondence to different fragments of $\mathcal{L}^k_\infty(\mathcal{Q}_n)$. In particular, we introduce a "one-way" version of $\mathbf{Bij}^k_n$ which is crucial to our construction of a generalised version of the $\mathbb{P}_k$ comonad.

3.1 Relaxing $\mathbf{Bij}^k_n$

Recall that each round of $\mathbf{Bij}^k_n(\mathcal{A},\mathcal{B})$ involves Duplicator selecting a bijection $f : A \to B$ and ends with a test of whether for the pebbled positions $(a_i, b_i)_{i\in[k]}$ it is the case that, for any $\{i_1, \dots i_r\} \subset [k]$, $(a_{i_1}, \dots a_{i_r}) \in R^{\mathcal{A}} \iff (b_{i_1}, \dots b_{i_r}) \in R^{\mathcal{A}}$ where Duplicator loses if the test is failed. For the rest of the round, Spoiler rearranges up to n pebbles on $\mathcal{A}$ with the corresponding pebbles on $\mathcal{B}$ moved according to f.

So, to create from $\mathbf{Bij}^k_n$ a "one-way" game from $\mathcal{A}$ to $\mathcal{B}$ we need to relax the condition that f be a bijection and the $\iff$ in the final test. The following definition captures the most basic such relaxation:

▶ **Definition 1.** *For two relational structures $\mathcal{A}$, $\mathcal{B}$, the positive k-pebble n-function game, $+$**Fun**$^k_n(\mathcal{A},\mathcal{B})$ is played by Spoiler and Duplicator. Prior to the jth round the position consists of partial maps $\pi^a_{j-1} : [k] \rightharpoonup A$ and $\pi^n_{j-1} : [k] \rightharpoonup B$. In Round j*

- *Duplicator provides a function $h_j : A \to B$ such that for each $i \in [k]$, $h_j(\pi^a_{j-1}(i)) = \pi^b_{j-1}(i)$.*
- *Spoiler picks up to n distinct* pebbles, *i.e. elements $p_1, \dots p_m \in [k] (m \leq n)$ and m elements $x_1, \dots x_m \in A$.*
- *The updated position is given by $\pi^a_j(p_l) = x_l$ and $\pi^b_j(p_l) = h_j(x_l)$ for $l \in [m]$; and $\pi^a_j(i) = \pi^a_{j-1}(i)$ and $\pi^b_j(i) = \pi^b_{j-1}(i)$ for $i \notin \{p_1, \dots, p_m\}$.*
- *Spoiler has won the game if there is some $R \in \sigma$ and $(i_1, \dots i_r) \in [k]^r$ with $(\pi^a_j(i_1), \dots, \pi^a_j(i_r)) \in R^{\mathcal{A}}$ but $(\pi^b_j(i_1), \dots, \pi^b_j(i_r)) \notin R^{\mathcal{B}}$.*

Duplicator wins by preventing Spoiler from winning.

As this game is to serve as the appropriate one-way game for $\mathbf{Bij}^k_n$, it is worth asking why this game is a reasonable generalisation of $\exists\mathbf{Peb}_k$ (the one-way game for $\mathbf{Bij}_k$). The answer comes in recalling Abramsky et al.'s presentation of a (deterministic) strategy for Duplicator in $\exists\mathbf{Peb}_k(\mathcal{A},\mathcal{B})$ as a collection of *branch maps* $\phi_{s,i} : A \to B$ for each $s \in (A \times [k])^*$, a history of Spoiler moves and $i \in [k]$ a pebble index. These branch maps tell us how Duplicator would respond to Spoiler moving pebble i to any element in A given the moves s that Spoiler has played in preceding rounds. The functions h_j in Definition 1 serve as just such branch maps.

In addition to this game, we now define some other relaxations of $\mathbf{Bij}^k_n$ which are important. In particular we define the following *positive* games by retaining that the pebbled position need only preserve positive atoms at the end of each round but varying the condition on f.

▶ **Definition 2.** *For two relational structures $\mathcal{A}$, $\mathcal{B}$, the positive k-pebble n-injection (resp. surjection, bijection) game, $+\boldsymbol{Inj}^k_n(\mathcal{A},\mathcal{B})$ (resp. $+\boldsymbol{Surj}^k_n(\mathcal{A},\mathcal{B})$, $+\boldsymbol{Bij}^k_n(\mathcal{A},\mathcal{B})$) is played by Spoiler and Duplicator. Prior to the jth round the position consists of partial maps $\pi^a_{j-1} : [k] \rightharpoonup A$ and $\pi^n_{j-1} : [k] \rightharpoonup B$. In Round j*

- *Duplicator provides an injection (resp. a surjection, bijection) $h_j : A \to B$ such that for each $i \in [k]$, $h_j(\pi^a_{j-1}(i)) = \pi^b_{j-1}(i)$.*
- *Spoiler picks up to n distinct* pebbles, *i.e. elements $p_1, \dots p_m \in [k](m \leq n)$ and m elements $x_1, \dots x_m \in A$.*
- *The updated position is given by $\pi^a_j(p_l) = x_l$ and $\pi^b_j(p_l) = h_j(x_l)$ for $l \in [m]$; and $\pi^a_j(i) = \pi^a_{j-1}(i)$ and $\pi^b_j(i) = \pi^b_{j-1}(i)$ for $i \notin \{p_1, \dots, p_m\}$.*
- *Spoiler has won the game if there is some $R \in \sigma$ and $(i_1, \dots i_r) \in [k]^r$ with $(\pi^a_j(i_1), \dots, \pi^a_j(i_r)) \in R^{\mathcal{A}}$ but $(\pi^b_j(i_1), \dots, \pi^b_j(i_r)) \notin R^{\mathcal{B}}$.*

Duplicator wins by preventing Spoiler from winning.

Strengthening the test condition in each round so that Spoiler wins if there is some $R \in \sigma$ and $(i_1, \dots i_r) \in [k]^r$ with $(\pi^a_j(i_1), \dots, \pi^a_j(i_r)) \in R^{\mathcal{A}}$ if, and only if, $(\pi^b_j(i_1), \dots, \pi^b_j(i_r)) \notin R^{\mathcal{B}}$, we get the definitions for the games $\mathbf{Fun}^k_n$, $\mathbf{Inj}^k_n$, $\mathbf{Surj}^k_n$ and $\mathbf{Bij}^k_n$ where the latter is precisely the n-bijective k-pebble game of Hella. We recap the poset of the games we've just defined ordered by strengthening of the rules/restrictions on Duplicator in the leftmost Hasse diagram in Figure 1. Here a game $\mathcal{G}$ is above $\mathcal{G}'$ if a Duplicator winning strategy in $\mathcal{G}$ is also one in $\mathcal{G}'$.

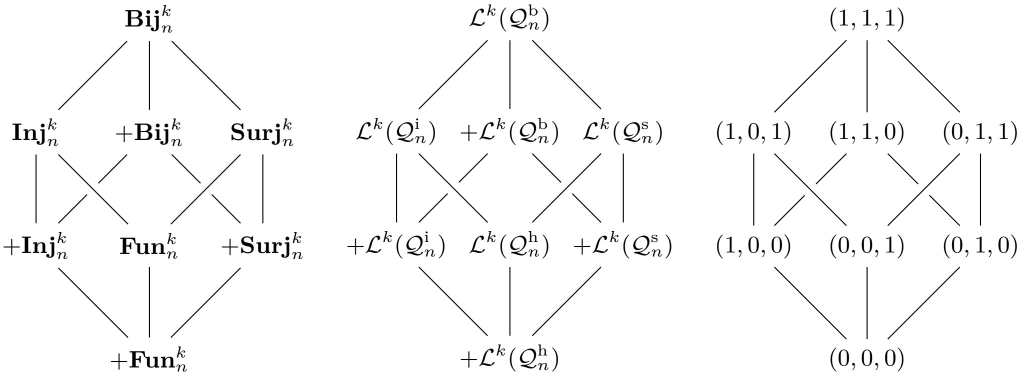

$\mathbf{Bij}^k_n$
$\mathbf{Inj}^k_n$
$+\mathbf{Bij}^k_n$
$\mathbf{Surj}^k_n$
$+\mathbf{Inj}^k_n$
$\mathbf{Fun}^k_n$
$+\mathbf{Surj}^k_n$
$+\mathbf{Fun}^k_n$
$\mathcal{L}^k(\mathcal{Q}^{\mathrm{b}}_n)$
$\mathcal{L}^k(\mathcal{Q}^{\mathrm{i}}_n)$
$+\mathcal{L}^k(\mathcal{Q}^{\mathrm{b}}_n)$
$\mathcal{L}^k(\mathcal{Q}^{\mathrm{s}}_n)$
$+\mathcal{L}^k(\mathcal{Q}^{\mathrm{i}}_n)$
$\mathcal{L}^k(\mathcal{Q}^{\mathrm{h}}_n)$
$+\mathcal{L}^k(\mathcal{Q}^{\mathrm{s}}_n)$
$+\mathcal{L}^k(\mathcal{Q}^{\mathrm{h}}_n)$
$(1,1,1)$
$(1,0,1)$
$(1,1,0)$
$(0,1,1)$
$(1,0,0)$
$(0,0,1)$
$(0,1,0)$
$(0,0,0)$

Figure 1 Hasse Diagrams of Games and Logics with Labels For Reference.

3.2 Logics with generalised quantifiers

In Section 2, we introduce for each $n, k \in \mathbb{N}$ the logics, $\mathcal{L}^k_\infty(\mathcal{Q}_n)$ as the infinitary logic extended with all generalised quantifiers of arity n. For $n = 1$ this logic leads somewhat of a double life. Kolaitis and Väänänen [17] show that this logic is equivalent to $\mathcal{C}^k$, the infinitary logic with counting quantifiers and at most k variables.

In this section we explore fragments of $\mathcal{L}^k_\infty(\mathcal{Q}_n)$ defined by restricted classes of generalised quantifiers, which we introduce next.

▶ **Definition 3.** *A class of σ-structures K is homomorphism-closed if for all homomorphisms $f : \mathcal{A} \to \mathcal{B}$ we have that $\mathcal{A} \in K \implies \mathcal{B} \in K$. Similarly, we say K is injection-closed (resp. surjection-closed, bijection-closed) if for all injective homomorphisms (resp. surjective, bijective homomorphisms) $f : \mathcal{A} \to \mathcal{B}$, we have $\mathcal{A} \in K \implies \mathcal{B} \in K$.*

We write $\mathcal{Q}^h_n$ for the class of all generalised quantifiers Q_K of arity n where K is homomorphism-closed. Similarly, we write $\mathcal{Q}^i_n$, $\mathcal{Q}^s_n$ and $\mathcal{Q}^b_n$ for the collections of n-ary quantifiers based on injection-closed, surjection-closed and bijection-closed classes.

In order to define logics which incorporate these restricted classes of quantifiers, we first define a base logic without quantifiers or negation.

▶ **Definition 4.** *We denote by $+\mathcal{L}^k$ the class of positive infinitary k-variable quantifier-free formulas. That means the k-variable fragment of the class of formulas $+\mathcal{L}[\sigma]$ (for any signature σ), given by the grammar*

$$\phi ::= R(x_1, \dots x_m) \mid \bigwedge_{\mathcal{I}} \phi \mid \bigvee_{\mathcal{J}} \phi$$

for $R \in \sigma$. We use $\mathcal{L}^k$ to denote a similar class of formulas but with negation permitted on atoms.

This basic set of formulas can be extended into a logic by adding some set of quantifiers as described here:

▶ **Definition 5.** *For $\mathcal{Q}$ some collection of generalised quantifiers, we denote by $+\mathcal{L}^k(\mathcal{Q})$ the smallest extension of $+\mathcal{L}^k$ to include the all quantifiers $Q_K \in \mathcal{Q}$, closed under quantification and ordinary boolean operations (excluding negation). $\mathcal{L}^k(\mathcal{Q})$ is the same logic but with negation on atoms. Note that $\exists^+\mathcal{L}^k_\infty \equiv +\mathcal{L}^k(\exists)$ and, as we can always push negation down to the level of atoms in $\mathcal{L}^k_\infty$, $\mathcal{L}^k_\infty \equiv \mathcal{L}^k(\exists, \forall)$.*

With this definition we are ready to introduce our logics. These are $\mathcal{L}^k(\mathcal{Q}^{\mathrm{h}}_n)$, $\mathcal{L}^k(\mathcal{Q}^{\mathrm{i}}_n)$, $\mathcal{L}^k(\mathcal{Q}^{\mathrm{s}}_n)$ and $\mathcal{L}^k(\mathcal{Q}^{\mathrm{b}}_n)$ and their positive counterparts $+\mathcal{L}^k(\mathcal{Q}^{\mathrm{h}}_n)$, $+\mathcal{L}^k(\mathcal{Q}^{\mathrm{i}}_n)$, $+\mathcal{L}^k(\mathcal{Q}^{\mathrm{s}}_n)$ and $+\mathcal{L}^k(\mathcal{Q}^{\mathrm{b}}_n)$. The obvious inclusion relationships between these logics are given by the middle Hasse diagram in Figure 1. As we shall see, these logics are governed exactly by the games pictured in the leftmost diagram in Figure 1.

Here we highlight two results relating this family of logics with more familiar infinitary logics. These results classify the two extreme logics in the diagram from Figure 1, namely $\mathcal{L}^k(\mathcal{Q}^{\mathrm{b}}_n)$ and $+\mathcal{L}^k(\mathcal{Q}^{\mathrm{h}}_n)$.

▶ **Lemma 6.** *For all $n, k \in \mathbb{N}$, $\mathcal{L}^k(\mathcal{Q}^b_n) \equiv \mathcal{L}^k_\infty(\mathcal{Q}_n)$.*

▶ **Lemma 7.** $+\mathcal{L}^k(\mathcal{Q}^h_1) \equiv +\mathcal{L}^k(\exists)$

3.3 Games and logics correspond

So far we have introduced a series of games and logics which are all variations on Hella's n-bijection k-pebble game, $\mathbf{Bij}^k_n$, and the corresponding logic $\mathcal{L}^k_\infty(\mathcal{Q}_n)$. Here we show that these games and logics match up in the way as one would expect looking at the respective refinement posets in Figures 1.

In order to present the proof of this in a uniform fashion, we label the corners of these cubes by three parameters $x_i, x_s, x_n \in \{0, 1\}$, standing for *injection*, *surjection* and *negated atoms* respectively. This can be seen in Figure 1.

Now we define the aliases of each of the games which modify $\mathbf{Fun}_n^k$ as follows, with the games defined lining up with the games defined in Section 3.1.

▶ **Definition 8.** *For two σ-structures $\mathcal{A}$ and $\mathcal{B}$, the game (x_i, x_s, x_n)-$\boldsymbol{Fun}_n^k(\mathcal{A}, \mathcal{B})$ is played by Spoiler and Duplicator in the same fashion as the game $\boldsymbol{Fun}_n^k(\mathcal{A}, \mathcal{B})$ with the following additional rules:*

1. *When Duplicator provides a function $f : A \to B$ at the beginning of a round, f is required to be injective if $x_i = 1$ and surjective if $x_s = 1$.*
2. *If $x_n = 1$, Spoiler wins at move j if the partial map taking $\pi_j^a(i)$ to $\pi_j^b(i)$ fails to preserve negated atoms as well as atoms.*

Similarly, we define parameterised aliases for the logics introduced in Section 3.2. To lighten our notational burden, we use $\mathcal{H}^{n,k}$ to denote the logic $+\mathcal{L}^k(\mathcal{Q}_n^{\mathrm{h}})$ throughout this section.

▶ **Definition 9.** *We define $\mathcal{H}_{\mathbf{x}}^{n,k}$ to be the logic $\mathcal{H}^{n,k}$ extended by*

1. *all n-ary generalised quantifiers closed by all homomorphisms which are injective, if $x_i = 1$; and surjective, if $x_s = 1$.*
2. *if $x_n = 1$, negation on atoms.*

For example, $\mathcal{H}_{001}^{n,k}$ extends $\mathcal{H}^{n,k}$ with negation on atoms but contains no additional quantifiers as all n-ary quantifiers closed under homomorphisms are already in $\mathcal{H}^{n,k}$. On the other hand, $\mathcal{H}_{110}^{n,k}$ does not allow negation on atoms but allows all quantifiers that are closed under bijective homomorphisms.

Now to prove the desired correspondence between $\mathbf{x}$-$\mathbf{Fun}_n^k$ and $\mathcal{H}_{\mathbf{x}}^{n,k}$, we adapt a proof from Hella[16] to work for this parameterised set of games. For this we need the language of *back-and-forth* systems which Hella uses as an explicit representation of a Duplicator winning strategy. We provide the appropriate generalised definition here:

▶ **Definition 10.** *Let $Part_{x_n}^k(\mathcal{A}, \mathcal{B})$ be the set of all partial functions $A \rightharpoonup B$ which preserve atoms (i.e. are partial homomorphisms) and, if $x_n = 1$ additionally preserve negated atoms.*

A set $\mathcal{S} \subset \boldsymbol{Part}_{x_n}^k(\mathcal{A}, \mathcal{B})$ is a back-and-forth system for the game (x_i, x_s, x_n)-$\boldsymbol{Fun}_n^k(\mathcal{A}, \mathcal{B})$ if it satisfies the following properties:

- ***Closure under subfunction**: If $f \in \mathcal{S}$ then $g \in \mathcal{S}$ for any $g \subset f$*
- ***(x_i, x_s)-forth property** For any f in $\mathcal{S}$ s.t. $|f| < k$, there exists a function $\phi_f : A \to B$, which is injective if $x_i = 1$ and surjective if $x_s = 1$ s.t. for every $C \subset dom(f), D \subset A$ with $|D| \leq n$ we have $(f \upharpoonright C) \cup (\phi_f \upharpoonright D) \in \mathcal{S}$.*

As this definition is essentially an unravelling of a Duplicator winning strategy for the game (x_i, x_s, x_n)-$\mathbf{Fun}_n^k(\mathcal{A}, \mathcal{B})$ we see that

▶ **Lemma 11.** *There is a back-and-forth system $\mathcal{S}$ containing the empty partial homomorphism $\emptyset$ if, and only if, Duplicator has a winning strategy for the game (x_i, x_s, x_n)-$\boldsymbol{Fun}_n^k(\mathcal{A}, \mathcal{B})$*

Following Hella, we define the *canonical* back-and-forth system for a game as follows:

▶ **Definition 12.** *The canonical back-and-forth system for (x_i, x_s, x_n)-$\boldsymbol{Fun}_n^k(\mathcal{A}, \mathcal{B})$ is denoted $I_{\mathbf{x}}^{n,k}(\mathcal{A}, \mathcal{B})$ and is given by the intersection $\bigcap_m I_{\mathbf{x}}^{n,k,m}(\mathcal{A}, \mathcal{B})$, whose conjuncts are defined inductively by setting $I_{\mathbf{x}}^{n,k,0}(\mathcal{A}, \mathcal{B}) := \boldsymbol{Part}_{x_n}^k(\mathcal{A}, \mathcal{B})$ and letting $I_{\mathbf{x}}^{n,k,m+1}(\mathcal{A}, \mathcal{B})$ be the set of $\rho \in I_{\mathbf{x}}^{n,k,m}(\mathcal{A}, \mathcal{B})$ such that ρ satisfies the (x_i, x_s)-forth condition with respect to the set $I_{\mathbf{x}}^{n,k,m}(\mathcal{A}, \mathcal{B})$*

It is not difficult to see that for any back-and-forth system $\mathcal{S}$ for $\mathbf{x}$-$\mathbf{Fun}^k_n(\mathcal{A},\mathcal{B})$ we have $\mathcal{S} \subset I^{n,k}_{\mathbf{x}}(\mathcal{A},\mathcal{B})$. This means that there is a winning strategy for Duplicator in the game $\mathbf{x}$-$\mathbf{Fun}^k_n(\mathcal{A},\mathcal{B})$ if, and only if, $I^{n,k}_{\mathbf{x}}(\mathcal{A},\mathcal{B})$ is not empty.

To complete the vocabulary needed to emulate Hella's proof in this setting we introduce the following generalisations of Hella's definitions.

▶ **Definition 13.** *Denote by $J^{n,k}_{\mathbf{x}}(\mathcal{A},\mathcal{B})$ the set of all $\rho \in Part^k_{x_n}(\mathcal{A},\mathcal{B})$ which preserve the validity of all $\mathcal{H}^{n,k}_{\mathbf{x}}$ formulas in elements of $dom(\rho)$. Let $\exists^+\mathbf{FO}^{n,k}_{\mathbf{x}}$ denote the fragment of $\mathcal{H}^{n,k}_{\mathbf{x}}$ with only finitary conjunctions and disjunctions. Denote by $K^{n,k}_{\mathbf{x}}(\mathcal{A},\mathcal{B})$ the set of all $\rho \in Part^k_{x_n}(\mathcal{A},\mathcal{B})$ which preserve the validity of all $\exists^+\mathbf{FO}^{n,k}_{\mathbf{x}}$ formulas in elements of $dom(\rho)$.*

An adaptation of Hella's argument yields the following Lemma:

▶ **Lemma 14.** *For $\mathcal{A},\mathcal{B}$ finite relational structures, $I^{n,k}_{\mathbf{x}}(\mathcal{A},\mathcal{B}) = J^{n,k}_{\mathbf{x}}(\mathcal{A},\mathcal{B}) = K^{n,k}_{\mathbf{x}}(\mathcal{A},\mathcal{B})$*

We conclude this section by showing the desired correspondence for the whole family of games and logics we have introduced.

▶ **Theorem 15.** *For $\mathbf{x} \in \{0,1\}^3$ and all $n,k \in \mathbb{N}$ the following are equivalent:*

- *Duplicator has a winning strategy for $\mathbf{x}$-$\mathbf{Fun}^k_n(\mathcal{A},\mathcal{B})$*
- $\mathcal{A} \Rrightarrow_{\mathcal{H}^{n,k}_{\mathbf{x}}} \mathcal{B}$
- $\mathcal{A} \Rrightarrow_{\exists^+\mathbf{FO}^{n,k}_{\mathbf{x}}} \mathcal{B}$

The case of $n = 1$ for this correspondence is particularly interesting as we can show that unary injection-closed and surjection-closed quantifiers are generated by all counting quantifiers and the quantifiers $\{\exists, \forall\}$ respectively.

4 The Comonad and Kleisli Category

In this section, we show how to construct a game comonad $\mathbb{G}_{n,k}$ which captures the strategies of $+\mathbf{Fun}^k_n$ in the same way that $\mathbb{P}_k$ captures the strategies of $\exists\mathbf{Peb}_k$. We do this using a new technique for constructing new game comonads from old based on strategy translation. We then show that different types of morphism in the Kleisli category of this new comonad correspond to Duplicator strategies for the games introduced in Section 3.

4.1 Translating between games

The pebbling comonad is obtained by defining a structure $\mathbb{P}_k\mathcal{A}$ for each $\mathcal{A}$ whose universe consists of (non-empty) lists in $(A \times [k])^*$ which we think of as sequences of moves by Spoiler in a game $\mathbf{Peb}_k(\mathcal{A},\mathcal{B})$, with $\mathcal{B}$ unspecified. With this in mind, we call a sequence in $(A \times [k])^*$ a *k-history* (allowing the empty sequence). In contrast, a move in the game $+\mathbf{Fun}^k_n(\mathcal{A},\mathcal{B})$ involves Spoiler moving up to n pebbles and therefore a history of Spoiler moves is a sequence in $((A \times [k])^{\leq n})^*$. We call such a sequence an *n,k-history.* With this set-up, (deterministic) strategies are given by functions $((A \times [k])^* \times [k]) \to (A \to B)$ for $\mathbf{Peb}_k(\mathcal{A},\mathcal{B})$ and $((A \times [k])^{\leq n})^* \to (A \to B)$ for $+\mathbf{Fun}^k_n(\mathcal{A},\mathcal{B})$.

A winning strategy for Duplicator in $+\mathbf{Fun}^k_n(\mathcal{A},\mathcal{B})$ can always be translated into one in $\mathbf{Peb}_k(\mathcal{A},\mathcal{B})$. We aim now to establish conditions when a translation can be made in the reverse direction. For this, it is useful to establish some machinery.

There is a natural *flattening* operation that takes n,k-histories to k-histories. We denote the operation by F, so $F([s_1, s_2, \ldots, s_m]) = s_1 \cdot s_2 \cdots s_m$, where $s_1, \ldots s_m \in (A \times [k])^{\leq n}$. Of course, the function F is not injective and has no inverse. It is worth, however, considering functions G from k-histories to n, k-histories that are inverse to F in the sense that $F(G(t)) = t$. One obvious such function takes a k-history $s_1, \ldots, s_m$ to the n, k-history $[[s_1], \ldots, [s_m]]$, i.e. the sequence of one-element sequences. This is, in some sense, minimal in that imposes the minimal amount of structure on $G(t)$. We are interested in a *maximal* such function. For this, recall that the sequences in $(A \times [k])^{\leq n}$ that form the elements of an n, k-history have length at most n and do not have a repeated index from $[k]$. We aim to break a k-history t into maximal such blocks. This leads us to the following definition.

▶ **Definition 16.** *A list $s \in (A \times [k])^*$ is called* basic *if it contains fewer than or equal to n pairs and the pebble indices are all distinct.*

The n-structure function $S_n : (A \times [k])^* \to ((A \times [k])^{\leq n})^*$ is defined recursively as follows:

- *$S_n(s) = [s]$ if s is basic*
- *otherwise, $S_n(s) = [a]; S_n(t)$ where $s = a \cdot t$ such that a is the largest basic prefix of s.*

It is immediate from the definition that $F(S_n(t)) = t$. It is useful to characterise the range of the function S_n, which we do through the following definition.

▶ **Definition 17.** *An n, k-history t is* structured *if whenever s and s' are successive elements of t, then either s has length exactly n or s' begins with a pair (a, p) such that p occurs in s.*

It is immediate from the definitions that $S_n(s)$ is structured for all k-histories s and that an n, k-history is structured if, and only if, $S_n(F(s)) = s$.

We are now ready to characterise those Duplicator winning strategies for $\exists\mathbf{Peb}_k$ that can be lifted to $+\mathbf{Fun}^k_n$. First, we define a function that lifts a position in $\exists\mathbf{Peb}_k$ that Duplicator must respond to, i.e. a pair (s, p) where s is a k-history and p a pebble index, to a position in $+\mathbf{Fun}^k_n$, i.e. an n, k-history.

▶ **Definition 18.** *Suppose s is a k-history and s' is the last basic list in $S_n(s)$, so $S_n(s) = t; [s']$. Let $p \in [k]$ be a pebble index.*

*Define the n-*structuring *$\alpha_n(s, p)$ of (s, p) by*

$$\alpha_n(s,p) = \begin{cases} t; [s'] & \text{if } |s'| = n \text{ or } p \text{ occurs in } s' \\ t & \text{otherwise.} \end{cases}$$

▶ **Definition 19.** *Say that a Duplicator strategy $\Psi : ((A \times [k])^* \times [k]) \to (A \to B)$ in $\exists\boldsymbol{Peb}_k$ is n-consistent if for all k-histories s and s' and all pebble indices p and p':*

$$\alpha_n(s,p) = \alpha_n(s',p') \quad \Rightarrow \quad \Psi(s,p) = \Psi(s',p').$$

Intuitively, an n-consistent Duplicator strategy in the game $\exists\mathbf{Peb}_k(\mathcal{A}, \mathcal{B})$ is one where Duplicator plays the same function in all moves that could be part of the same Spoiler move in the game $+\mathbf{Fun}^k_n(\mathcal{A}, \mathcal{B})$. We are then ready to prove the main result of this subsection.

▶ **Lemma 20.** *Duplicator has an n-consistent winning strategy in $\exists\boldsymbol{Peb}_k(\mathcal{A}, \mathcal{B})$ if, and only if, it has a winning strategy in $+\boldsymbol{Fun}^k_n(\mathcal{A}, \mathcal{B})$.*

4.2 Lifting the comonad $\mathbb{P}_k$ to $\mathbb{G}_{n,k}$

Central to Abramsky et al.'s construction of the pebbling comonad is the observation that for I-structures (defined in Section 2), maps in the Kleisli category $\mathcal{K}(\mathbb{P}_k)$ correspond to Duplicator winning strategies in $\exists\mathbf{Peb}_k(\mathcal{A}, \mathcal{B})$.

▶ **Lemma 21** ([3]). *For $\mathcal{A}$ and $\mathcal{B}$ I-structures over the signature σ, there is a homomorphism $\mathbb{P}_k\mathcal{A} \to \mathcal{B}$ if and only if there is a (deterministic) winning strategy for Duplicator in the game $\exists\boldsymbol{Peb}_k(\mathcal{A}, \mathcal{B})$*

The relation to strategies is clear in the context of elements $s \in \mathbb{P}_k\mathcal{A}$ representing histories of Spoiler moves up to and including the current move in the game $\exists\mathbf{Peb}_k(\mathcal{A}, \mathcal{B})$. The relational structure given to this set by Abramsky, Dawar and Wang ensures that pebbled positions preserve relations in σ, while the caveat here about I-structures is a technicality to ensure that the pebbled positions when "playing" according to a map f all define partial homomorphisms, in particular they give well defined partial maps from A to B.

As we saw in Lemma 20 a Duplicator winning strategy in $+\mathbf{Fun}^k_n(\mathcal{A}, \mathcal{B})$ is given by an n-consistent strategy in $\exists\mathbf{Peb}_k(\mathcal{A}, \mathcal{B})$. The n-consistency condition can be seen as saying that the corresponding map $f : \mathbb{P}_k\mathcal{A} \to \mathcal{B}$ must, on certain "equivalent" elements of $\mathbb{P}_k\mathcal{A}$ give the same value. We can formally define the equivalence relation as follows.

▶ **Definition 22.** *For $n \in \mathbb{N}$ and $\mathcal{A}$ a relational structure. Define $\approx_n$ on the universe of $\mathbb{P}_k\mathcal{A}$ as follows:*

$$[s; (a, i)] \approx_n [t; (b, j)] \iff a = b \textit{ and } \alpha_n((s, i)) = \alpha_n((t, j))$$

In general, for any structured n, k-history t, we write $[t|a]$ to denote the $\approx_n$-equivalence class of an element $[s; (a, i)] \in \mathbb{P}_k\mathcal{A}$ with $\alpha_n(s, i) = t$.

This allows us to define the main construction of this section as a quotient of the relational structure $\mathbb{P}_k\mathcal{A}$. Note that the relation $\approx_n$ is not a congruence of this structure, so there is not a canonical quotient. This is because don't have that $\mathbf{a} \approx_n \mathbf{b}$ does not imply $\mathbf{a} \in R^{\mathcal{A}} \iff \mathbf{b} \in R^{\mathcal{A}}$. Given an arbitrary equivalence relation $\sim$ over a relational structure $\mathcal{M}$, there are two standard ways to define relations in a quotient $\mathcal{M}/\sim$. We could say that a tuple $(c_1, \ldots c_r)$ of equivalence classes is in a relation $R^{\mathcal{M}/\sim}$ if, and only if, *every* choice of representatives is in $R^{\mathcal{M}}$ or if *some* choice of representatives is in $R^{\mathcal{M}}$. The latter definition has the advantage that the quotient map from $\mathcal{M}$ to $\mathcal{M}/\sim$ is a homomorphism.

▶ **Definition 23.** *For $n, k \in \mathbb{N}$, $k \geq n$ and σ a relational signature, we define the functor $\mathbb{G}_{n,k} : \mathcal{R}(\sigma) \to \mathcal{R}(\sigma)$ by:*

- ***On objects*** *$\mathbb{G}_{n,k}\mathcal{A} := \mathbb{P}_k\mathcal{A}/\approx_n$.*
- ***On morphisms*** *$\mathbb{P}_k f/\approx_n$ is well-defined as $\mathbb{P}_k f$ only changes the elements not the pebble indices.*

Writing $q_n : \mathbb{P}_k\mathcal{A} \to \mathbb{G}_{n,k}\mathcal{A}$ for the quotient map enables us to establish the following useful property.

▶ **Observation 24.** *$f : \mathbb{G}_{n,k}\mathcal{A} \to \mathcal{B}$ is a homomorphism if, and only if, $f \circ q_n : \mathbb{P}_k\mathcal{A} \to \mathcal{B}$ is a homomorphism.*

Combining this with Lemma 20, we have the appropriate generalisation of Lemma 21.

▶ **Lemma 25.** *For I-structures $\mathcal{A}$ and $\mathcal{B}$, there is a homomorphism $f : \mathbb{G}_{n,k}\mathcal{A} \to \mathcal{B}$ if, and only if, there is a winning strategy for the Duplicator in the game $+\boldsymbol{Fun}^k_n$*

Furthermore, we can see that

▶ **Lemma 26.** *The counit ϵ and comultiplication δ for $\mathbb{P}_k$ lift to well-defined natural transformations for $\mathbb{G}_{n,k}$*

We will call these lifted natural transformations $\epsilon^{n,k} : \mathbb{G}_{n,k} \to 1$ and $\delta^{n,k} : \mathbb{G}_{n,k} \to \mathbb{G}_{n,k}\mathbb{G}_{n,k}$. As $q_n \circ \mathbb{P}_k q_n = \mathbb{G}_{n,k} q_n \circ q_n$, we have that for any $t \in (\mathbb{P}_k)^m \mathcal{A}$ the notion of "the" equivalence class of t, $\mathbf{q_n}(t) \in (\mathbb{G}_{n,k})^m \mathcal{A}$ is well-defined. So for any term T built from composing ϵ, δ and $\mathbb{P}_k$ we have that the term $\tilde{T}$, obtained by replacing ϵ by $\epsilon^{n,k}$, δ with $\delta^{n,k}$ and $\mathbb{P}_k$ with $\mathbb{G}_{n,k}$ satisfies $\mathbf{q_n}(T(t)) = \tilde{T}(\mathbf{q_n}(t))$ by the above proof. Now as the counit and coassociativity laws are equations in ϵ and δ which remain true on taking the quotient we have the following result.

▶ **Theorem 27.** *$(\mathbb{G}_{n,k}, \epsilon^{n,k}, \delta^{n,k})$ is a comonad on $\mathcal{R}(\sigma)$*

4.3 Classifying the morphisms of $\mathcal{K}(\mathbb{G}_{n,k})$

In Abramsky et al.'s treatment of the Kleisli category of $\mathbb{P}_k$ [3] they classify the morphisms according to whether their *branch maps* are injective, surjective or bijective. We extend this definition to the comonad $\mathbb{G}_{n,k}$. This gives us a way of classifying the morphisms to match the classification of strategies given in Section 3.

▶ **Definition 28.** *For $f : \mathbb{G}_{n,k}\mathcal{A} \to \mathcal{B}$ a Kleisli morphism of $\mathbb{G}_{n,k}$, the* branch maps *of f are defined as the following collection of functions $A \to B$, indexed by the structured n, k-histories $t \in ((A \times [k])^{\leq n})^*$ and defined as $\phi^f_t(x) := f([t|x])$. We say that such an f is* branch-bijective *(resp.* branch-injective, -surjective*) if for every t ϕ^f_t is bijective (resp. injective, surjective). We denote these maps by $\mathcal{A} \to^b_{n,k}$ (resp. $\mathcal{B}$ $\mathcal{A} \to^i_k \mathcal{B}$ and $\mathcal{A} \to^s_k \mathcal{B}$)*

Informally, the branch map ϕ^g_s is the response given by Duplicator in the $+\mathbf{Fun}^k_n(\mathcal{A}, \mathcal{B})$ when playing according to the strategy represented by g after Spoiler has made the series of plays in s. This gives us another way of classifying the Duplicator winning strategies for the games from Section 3.

▶ **Lemma 29.** *There is a winning strategy for Duplicator in the game $+\boldsymbol{Bij}^k_n(\mathcal{A}, \mathcal{B})$ (resp. $+\boldsymbol{Inj}^k_n(\mathcal{A}, \mathcal{B})$, $+\boldsymbol{Surj}^k_n(\mathcal{A}, \mathcal{B})$) if and only if $\mathcal{A} \to^b_{n,k} \mathcal{B}$ (resp. $\mathcal{A} \to^i_{n,k} \mathcal{B}$, $\mathcal{A} \to^s_{n,k} \mathcal{B}$).*

Expanding this connection between Kleisli maps and strategies, we define the following:

▶ **Definition 30.** *We say a a Kleisli map $f : \mathbb{G}_{n,k}\mathcal{A} \to \mathcal{B}$ is* strongly *branch-bijective (resp.* strongly *branch-injective, -surjective) if the strategy for $+\boldsymbol{Bij}^k_n(\mathcal{A}, \mathcal{B})$ (resp. $+\boldsymbol{Inj}^k_n(\mathcal{A}, \mathcal{B}), +\boldsymbol{Surj}^k_n(\mathcal{A}, \mathcal{B})$) is also a winning strategy for the game $\boldsymbol{Bij}^k_n(\mathcal{A}, \mathcal{B})$ (resp. $\boldsymbol{Inj}^k_n(\mathcal{A}, \mathcal{B}), \boldsymbol{Surj}^k_n(\mathcal{A}, \mathcal{B})$) and we denote these maps by $\mathcal{A} \rightarrowtriangle^b_{n,k}$ (resp. $\mathcal{B}$ $\mathcal{A} \rightarrowtriangle^i_k \mathcal{B}$ and $\mathcal{A} \rightarrowtriangle^s_k \mathcal{B}$)*

Now we generalise a result of Abramsky, Dawar and Wang to the Kleisli category $\mathcal{K}(\mathbb{G}_{n,k})$.

▶ **Lemma 31.** *For $\mathcal{A}, \mathcal{B}$ finite relational structures,*

$$\mathcal{A} \rightleftarrows^i_{n,k} \mathcal{B} \iff \mathcal{A} \rightleftarrows^s_{n,k} \mathcal{B} \iff \mathcal{A} \rightarrowtriangle^b_{n,k} \mathcal{B} \iff \mathcal{A} \cong_{\mathcal{K}(\mathbb{G}_{n,k})} \mathcal{B}$$

This lemma allows us to conclude that the isomorphisms in the category $\mathcal{K}(\mathbb{G}_{n,k})$ correspond with equivalence of structures up to k variable infinitary logic extended by all generalised quantifiers of arity at most n and thus with winning strategies for Hella's n-bijective k-pebble game.

▶ **Theorem 32.** *For two finite relational structures $\mathcal{A}$ and $\mathcal{B}$ the following are equivalent:*

- *$\mathcal{A} \cong_{\mathcal{K}(\mathbb{G}_{n,k})} \mathcal{B}$*
- *Duplicator has a winning strategy for $\boldsymbol{Bij}^k_n(\mathcal{A}, \mathcal{B})$*
- *$\mathcal{A} \equiv_{\mathcal{L}^k(\mathcal{Q}_n)} \mathcal{B}$*

Proof. Immediate from Lemma 31 and Hella [15]. ◀

5 Coalgebras and Decompositions

Abramsky et al. [3] show that the coalgebras of the comonad $\mathbb{P}_k$ are, in fact, objects of great interest to finite model theorists. That is, a coalgebra $\alpha : \mathcal{A} \to \mathbb{P}_k\mathcal{A}$ gives a tree decomposition of $\mathcal{A}$ of width at most $k-1$. Moreover, any such tree decomposition can be turned into a coalgebra. This result works, in essence, because $\mathbb{P}_k\mathcal{A}$ has a treelike structure where any pebble history, or branch, $s \in \mathbb{P}_k\mathcal{A}$ only witnesses the relations from the $\leq k$ elements of $\mathcal{A}$ which make up the pebbled position on s. So a homomorphism $\mathcal{A} \to \mathbb{P}_k\mathcal{A}$ witnesses a sort of treelike k-locality of the relational structure of $\mathcal{A}$ and the extra conditions of being a $\mathbb{P}_k$-coalgebra ensure this can be presented as a tree decomposition (of width $< k$).

In generalising this comonad to $\mathbb{G}_{n,k}$, we have given away some of the restrictive k-local nature of $\mathbb{P}_k$ which makes this argument work. For example, we note that the substructure induced on elements of the form $\{[\epsilon|x] \mid x \in \mathcal{A}\}$ witnesses all relations in $\mathcal{A}$ which have arity $\leq n$. So, in particular, if the maximum arity of σ the signature of $\mathcal{A}$ is less than n, then it is not hard to see how to construct a homomorphism, indeed a coalgebra, $A \to \mathbb{G}_{n,k}A$. So our notion of n-generalised tree decomposition should clearly be more permissive than the notion of tree decomposition, collapsing for $n \geq \text{arity}(\sigma)$ and otherwise allowing a controlled amount of non-locality (parameterised by n). The correct definition, as we prove in this section, is the following.

▶ **Definition 33.** *An* extended tree decomposition *of a σ-structure $\mathcal{A}$ is a triple (T, β, γ) with $\beta, \gamma : T \to 2^A$ such that:*

1. *(T, B) with $B : T \to 2^A$ given by $B(t) = \beta(t) \cup \gamma(t)$ is a tree-decomposition of $\mathcal{A}$; and*
2. *if $a \in \gamma(t)$ and $a \in B(t')$ then $t \leq t'$.*

Thus, we can see an extended tree decomposition as a tree decomposition (T, B) where, additionally, at each node t we pick out a subset $\gamma(t)$ of $B(t)$ with the property that every element a of $\mathcal{A}$ appears in at most one $\gamma(t)$ and when it does, this t is the root of the subtree of T in which a appears. We next define the width and arity of an extended tree decomposition.

▶ **Definition 34.** *The* width *of an extended tree decomposition (T, β, γ) is $\max_{t \in T} |\beta(t)|$.*

The arity *of an extended tree decomposition (T, β, γ) of width k is the least $n \leq k$ such that:*

1. *if $t < t'$ then $|\beta(t') \cap \gamma(t)| \leq n$; and*
2. *for every tuple $(a_1, \ldots, a_m)$ in every relation R of $\mathcal{A}$, there is a $t \in T$ such that $\{a_1, \ldots, a_m\} \subseteq B(t)$ and $|\{a_1, \ldots, a_m\} \cap \gamma(t)| \leq n$.*

Any extended tree decomposition $T(\beta, \gamma)$ of a structure $\mathcal{A}$ can be transformed into one in which each $a \in A$ appears in exactly one $\gamma(t)$. Indeed, suppose there is some a for which this is not true and let t be the order minimal element such that $s \in B(t)$. We simply split t into two nodes adding a parent t_a (with no other children) with $\gamma(t_a) = \{a\}$ and $\beta(t_a) = \beta(t) \setminus \{a\}$. This is easily seen to be an extended tree decomposition with the same width and arity. We call such a tree decomposition one in *normal form*.

We are particularly interested in extended tree decompositions that are further well-structured, in the sense that is related to the definition of structured n, k-histories in Section 4.

▶ **Definition 35.** *An extended tree decomposition with width k and arity n is* structured *if for every $a \in A$ there is a $t \in T$ s.t. $a \in \gamma(t)$, for every node t, $\gamma(t) \neq \emptyset$, for any child t' of t $\beta(t') \cap \gamma(t) \neq \emptyset$ and for any t'' a child of t' we have that either:*

- $|\beta(t') \cap \gamma(t)| = n$; *or*
- $|\beta(t')| < k$; *or*
- $\gamma(t) \cap \beta(t') \setminus \beta(t'') \neq \emptyset$

For a node t in an extended tree decomposition, we call $\beta(t)$ the *fixed bag* at t and $\gamma(t)$ the *floating bag* at t.

In general, extended tree decompositions of width k and arity 1 correspond exactly with tree decompositions of width k.

▶ **Lemma 36.** *A relational structure $\mathcal{A}$ has a tree decomposition of width k if, and only if, it has an extended tree decomposition of width k and arity 1*

Relating extended tree decompositions to our construction in Section 4, we note the following easy but important result.

▶ **Lemma 37.** *For any finite $\mathcal{A}$, there is a structured extended tree decomposition of $\mathbb{G}_{n,k}\mathcal{A}$ of width k and arity n for some $k, n \in \mathbb{N}$*

We now prove the main claim of this section, that the $\mathbb{G}_{n,k}$-coalgebras are in correspondence with structured extended tree decompositions of width k and arity n. The correspondence between tree decompositions and coalgebras of $\mathbb{P}_k$ was established in [3] through a partial order on a structure $\mathcal{A}$ called a *tree traversal*. We now introduce an analogous traversal structure to link $\mathbb{G}_{n,k}$-coalgebras and extended tree decompositions of width k and arity n. The following definitions provide precisely such a structure.

▶ **Definition 38.** *An n-*tree order *is a triple $(X, <, \sim)$ where $<$ is a partial order and $\sim$ an equivalence relation, both on the set X, such that:*
1. *for all $x, y, z \in X$, $x < y$ and $y \sim z$ implies $x < z$;*
2. *$(X/\sim, <)$ is a tree order; and*
3. *for each $x \in X$ and each $\sim$-equivalence class η, there are at most n elements $y \in \eta$ such that $y < x$.*

An n-tree order provides the order structure allowing us to define the traversals we need.

▶ **Definition 39.** *For a σ-structure $\mathcal{A}$, let $(A, <, \sim)$ be an n-tree order and $\iota : O \to 2^{[k]}$ a function, where $O = \{(a, b) \mid a < b\}$ such that*
1. *if $b < b'$ or $b \sim b'$, then $\iota(a, b) = \iota(a, b')$; and*
2. *if $a \neq a'$ and $a \sim a'$ then $\iota(a, b) \cap \iota(a', b) = \emptyset$.*
3. *if $\mathcal{C}$ is a $\sim$-equivalence class then $|\bigcup_{a \in \mathcal{C}} \iota(a, b)| \leq n$*

*This is an n, k-*traversal *of $\mathcal{A}$ if, for each tuple $(a_1, \dots, a_m)$ in any relation R of $\mathcal{A}$, we have:*
1. *for each $i, j \in [m]$ either $a_i < a_j$, $a_j < a_i$ or $a_i \sim a_j$;*
2. *no more than n elements of $\{a_1, \dots, a_m\}$ belong to any one $\sim$-equivalence class; and*
3. *if $a_i < a_j$, there exists $p_i \in \iota(a_i, a_j)$ such that for all $c \in A$ with $a_i < c < a_j$ then $p_i \notin \iota(c, a_j)$.*

An n, k-traversal is structured *if for any $a < b < c$ such that there is no d with $a < d < b$, we have that either:*
- $|\bigcup_{\{a' | a \sim a' and\ a' < c\}} \iota(a', c)| = n$; *or*
- $\bigcup_{\{a' | a \sim a' and\ a' < c\}} \iota(a', c) \cap \bigcup_{\{b' | b \sim b' and\ b' < c\}} \iota(b', c) \neq \emptyset$

We establish the relationship between extended tree decompositions and n, k-traversals in the following lemma.

▶ **Lemma 40.** *For a finite structure $\mathcal{A}$, if $\mathcal{A}$ has an extended tree decomposition of width k and arity n then it has an n, k-traversal. Furthermore, if the extended tree decomposition is structured then there is a structured n, k-traversal.*

We are ready to establish the relationship between n, k traversals and coalgebras of $\mathbb{G}_{n,k}$.

▶ **Lemma 41.** *There is a coalgebra $\alpha : \mathcal{A} \to \mathbb{G}_{n,k}\mathcal{A}$ if, and only if, there is a structured n, k-traversal of $\mathcal{A}$*

We finish this section by putting together these results into a single theorem.

▶ **Theorem 42.** *For $\mathcal{A}$ a finite relational structure the following are equivalent:*

1. *there is a $\mathbb{G}_{n,k}$-coalgebra $\alpha : \mathcal{A} \to \mathbb{G}_{n,k}\mathcal{A}$*
2. *there is a structured extended tree decomposition of $\mathcal{A}$ with width k and arity n*
3. *there is a structured n, k-traversal of $\mathcal{A}$*

6 Concluding Remarks

The work of Abramsky et al., giving comonadic accounts of pebble games and their relationship to logic has opened up a number of avenues of research. It raises the possiblilty of studying logical resources through a categorical lens and introduces the notion of *coresources*. This view has been applied to pebble games [3], Ehrenfeucht-Fraïssé games, bisimulation games [4] and also to quantum resources [1, 2]. In this paper we have extended this approach to logics with generalised quantifiers.

The construction of the comonad $\mathbb{G}_{n,k}$ introduces interesting new techniques to this project. The pebbling comonad $\mathbb{P}_k$ is graded by the value of k which we think of as a *coresource* increasing which constrains the morphisms. The new parameter n provides a second coresource, increasing which further constrains the moves of Duplicator. It is interesting that the resulting comonad can be obtained as a quotient of $\mathbb{P}_k$ and the strategy lifting argument developed in Section 4 could prove useful in other contexts. The morphisms in the Kleisli category correspond to winning strategies in a new game we introduce which characerises a natural logic: the positive logic of homomorphism-closed quantifiers. The isomorphisms correspond to an already established game: Hella's n-bijective game with k pebbles. This relationship allows for a systematic exploration of variations characterising a number of natural fragments of the logic with n-ary quantifiers. One natural fragment that is not yet within this framework and worth investigating is the logic of embedding-closed quantifiers of Haigora and Luosto [14].

This work opens up a number of perspectives. Logics with generalised quantifiers have been widely studied in finite model theory. They are less of interest in themselves and more as tools for proving inexpressibility in specific extensions of first-order or fixed-point logic. For instance, the logics with rank operators [7, 12], of great interest in descriptive complexity, have been analysed as fragments of a more general logic with linear-algebraic quantifiers [6]. It would be interesting to explore whether the comonad $\mathbb{G}_{n,k}$ could be combined with a vector space construction to obtain a categorical account of this logic.

More generally, the methods illustrated by our work could provide a way to deconstruct pebble games into their component parts and find ways of constructing entirely new forms of games and corresponding logics. The games we consider and classify are based on Duplicator playing different kinds of functions (i.e. morphisms on finite sets) and maintaining different kinds of homomorphisms (i.e. morphisms in the category of σ-structures). Could we build reasonable pebble games and logics on other categories? In particular, can we bring the algebraic pebble games of [9] into this framework?

References

1 Samson Abramsky, Rui Soares Barbosa, Nadish de Silva, and Octavio Zapata. The Quantum Monad on Relational Structures. In Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *LIPIcs*, pages 35:1–35:19, 2017. doi:10.4230/LIPIcs.MFCS.2017.35.

2 Samson Abramsky, Rui Soares Barbosa, Martti Karvonen, and Shane Mansfield. A comonadic view of simulation and quantum resources. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, 2019. doi:10.1109/LICS.2019.8785677.

3 Samson Abramsky, Anuj Dawar, and Pengming Wang. The pebbling comonad in finite model theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, June 2017. doi:10.1109/LICS.2017.8005129.

4 Samson Abramsky and Nihil Shah. Relating structure and power: Comonadic semantics for computational resources. In *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, volume 119 of *LIPIcs*, pages 2:1–2:17, 2018. doi:10.4230/LIPIcs.CSL.2018.2.

5 Anuj Dawar. Generalized Quantifiers and Logical Reducibilities. *Journal of Logic and Computation*, 5(2):213–226, 1995. doi:10.1093/logcom/5.2.213.

6 Anuj Dawar, Erich Grädel, and Wied Pakusa. Approximations of isomorphism and logics with linear-algebraic operators. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 112:1–112:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.112.

7 Anuj Dawar, Martin Grohe, Bjarki Holm, and Bastian Laubner. Logics with rank operators. In *Proceedings of the 2009 24th Annual IEEE Symposium on Logic In Computer Science*, LICS '09, pages 113–122, Washington, DC, USA, 2009. IEEE Computer Society. doi:10.1109/LICS.2009.24.

8 Anuj Dawar and Lauri Hella. The expressive power of finitely many generalized quantifiers. *Information and Computation*, 123(2):172–184, 1995.

9 Anuj Dawar and Bjarki Holm. Pebble games with algebraic rules. *Fundamenta Informaticae*, Vol. 150, nr 3/4:281–316, 2017.

10 Anuj Dawar, Steven Lindell, and Scott Weinstein. Infinitary logic and inductive definability over finite structures. *Information and Computation*, 119(2):160–175, 1995.

11 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory.* Springer, 2nd edition, 1999.

12 Erich Grädel and Wied Pakusa. Rank logic is dead, long live rank logic! *The Journal of Symbolic Logic*, 84(1):54–87, 2019. doi:10.1017/jsl.2018.33.

13 Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*, volume 47 of *Lecture Notes in Logic.* Cambridge University Press, 2017.

14 Jevgeni Haigora and Kerkko Luosto. On logics extended with embedding-closed quantifiers, 2014. arXiv:1401.6682.

15 Lauri Hella. Definability hierarchies of generalized quantifiers. *Annals of Pure and Applied Logic*, 43(3):235–271, 1989. doi:10.1016/0168-0072(89)90070-5.

16 Lauri Hella. Logical hierarchies in PTIME. *Information and Computation*, 129(1):1–19, 1996. doi:10.1006/inco.1996.0070.

17 Phokion G. Kolaitis and Jouko A. Väänänen. Generalized quantifiers and pebble games on finite structures. *Annals of Pure and Applied Logic*, 74(1):23–75, 1995. doi:10.1016/0168-0072(94)00025-X.

18 Phokion G. Kolaitis and Moshe Y. Vardi. Infinitary logic for computer science. In *Automata, Languages and Programming*, pages 450–473, 1992.

19 Phokion G. Kolaitis and Moshe Y. Vardi. A game-theoretic approach to constraint satisfaction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence*, pages 175–181, 2000.

20 Leonid Libkin. *Elements Of Finite Model Theory (Texts in Theoretical Computer Science. An Eatcs Series)*. SpringerVerlag, 2004.

21 Per Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32(3):186–195, 1966. doi:10.1111/j.1755-2567.1966.tb00600.x.

Semiring Provenance for Fixed-Point Logic

Katrin M. Dannert
RWTH Aachen University, Germany
dannert@logic.rwth-aachen.de

Erich Grädel
RWTH Aachen University, Germany
graedel@logic.rwth-aachen.de

Matthias Naaf
RWTH Aachen University, Germany
naaf@logic.rwth-aachen.de

Val Tannen
University of Pennsylvania, Philadelphia, PA, USA
val@cis.upenn.edu

Abstract

Semiring provenance is a successful approach, originating in database theory, to providing detailed information on how atomic facts combine to yield the result of a query. In particular, general provenance semirings of polynomials or formal power series provide precise descriptions of the evaluation strategies or "proof trees" for the query. By evaluating these descriptions in specific application semirings, one can extract practical information for instance about the confidence of a query or the cost of its evaluation.

This paper develops semiring provenance for very general logical languages featuring the full interaction between negation and fixed-point inductions or, equivalently, arbitrary interleavings of least and greatest fixed points. This also opens the door to provenance analysis applications for modal μ-calculus and temporal logics, as well as for finite and infinite model-checking games.

Interestingly, the common approach based on Kleene's Fixed-Point Theorem for ω-continuous semirings is not sufficient for these general languages. We show that an adequate framework for the provenance analysis of full fixed-point logics is provided by semirings that are (1) *fully continuous*, and (2) *absorptive*. Full continuity guarantees that provenance values of least and greatest fixed-points are well-defined. Absorptive semirings provide a symmetry between least and greatest fixed-points and make sure that provenance values of greatest fixed points are informative.

We identify *semirings of generalized absorptive polynomials* $\mathbb{S}^{\infty}[X]$ and prove universal properties that make them the most general appropriate semirings for our framework. These semirings have the further property of being (3) *chain-positive*, which is responsible for having *truth-preserving* interpretations that give non-zero values to all true formulae. We relate the provenance analysis of fixed-point formulae with provenance values of plays and strategies in the associated model-checking games. Specifically, we prove that the provenance value of a fixed point formula gives precise information on the evaluation strategies in these games.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases Finite Model Theory, Semiring Provenance, Absorptive Semirings, Fixed-Point Logics

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.17

Related Version Due to space reasons, a number of technical details and proofs have been omitted. They are available in the full version of this paper at `https://arxiv.org/abs/1910.07910`.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 17; pp. 17:1–17:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Provenance analysis for a logical statement ψ, evaluated on a finite structure $\mathfrak{A}$, aims at providing precise information why ψ is true or false in $\mathfrak{A}$. The approach of *semiring provenance*, going back to [16] relies on the idea of annotating the atomic facts by values from a commutative semiring, and to propagate these values through the statement ψ, keeping track whether information is used alternatively (as in disjunctions or existential quantifications) or jointly (as in conjunctions or universal quantifications). Depending on the chosen semiring, the provenance value may then give practical information for instance concerning the *confidence* we may have that $\mathfrak{A} \models \psi$, the *cost* of the evaluation of ψ on $\mathfrak{A}$, the number of successful evaluation strategies for ψ on $\mathfrak{A}$ in a game-theoretic sense, and so on. Beyond such provenance evaluations in specific application semirings, more general and more precise information is obtained by evaluations in so-called *provenance semirings* of polynomials or formal power series. Take, for instance, an abstract set X of *provenance tokens* that are used to label the atomic facts of a structure $\mathfrak{A}$, and consider the semiring $\mathbb{N}[X]$ of polynomials with indeterminates in X and coefficients from $\mathbb{N}$, which is the commutative semiring that is freely generated ("most general") over X. Such a labelling of the atomic facts then extends to a provenance valuation $\pi[\![\psi]\!] \in \mathbb{N}[X]$ for every Boolean query ψ from positive relational algebra RA^+ and, indeed, every negation-free first-order sentence $\psi \in \mathrm{FO}^+$. This provenance valuation gives precise information about the combinations of atomic facts that imply the truth of ψ in $\mathfrak{A}$. Indeed, we can write $\pi[\![\psi]\!]$ as a sum of monomials $m\, x_1^{e_1} \cdots x_k^{e_k}$. Each such monomial indicates that we have precisely m evaluation strategies (or "proof trees") to determine that $\mathfrak{A} \models \psi$ that make use of the atoms labelled by $x_1, \dots x_k$, and the atom labelled by x_i is used precisely e_i times by the strategy, see [16, 13].

Provenance for least fixed points. A similar analysis has been carried out for Datalog [6, 16]. Due to the need of unbounded least fixed-point iterations in the evaluation of Datalog queries, the underlying semirings have to satisfy the additional property of being ω-continuous. By Kleene's Fixed-Point Theorem, systems of polynomial equations then have least fixed-point solutions that can be computed by induction, reaching the fixed-point after at most ω stages. Most of the common application semirings are ω-continuous, or can easily be extended to one that is so; however, the most general ω-continuous provenance semiring over X is no longer a semiring of polynomials but the semiring of formal power series over X, denoted $\mathbb{N}^\infty[\![X]\!]$, with coefficients in $\mathbb{N}^\infty := \mathbb{N} \cup \{\infty\}$. As above, provenance valuations $\pi[\![\psi]\!] \in \mathbb{N}^\infty[\![X]\!]$ give precise information about the possible evaluation strategies for a Datalog query ψ on $\mathfrak{A}$. Even though $\mathfrak{A}$ is assumed to be finite there may be infinitely many such strategies, but each of them can use each atomic fact only a finite number of times; to put it differently, "proof trees" for $\mathfrak{A} \models \psi$ are still finite. This is closely related to the provenance analysis of reachability games on finite graphs [6, 14].

Negation: a stumbling block for wider applications. Semiring provenance has been applied to a number of other scenarios, such as nested relations, XML, SQL-aggregates, graph databases (see, e.g., the survey [17] as well as [21, 22]), and it is fair to say that in databases, semiring provenance analysis has been rather successful. However, its impact outside of databases has been very limited, despite the fact that the main questions addressed by provenance analysis, namely which parts of a large heterogeneous input structure are responsible for the evaluation of a logical statement, and the applications to cost, confidence, access control and so on are clearly interesting and relevant in many other branches of logics in computer science. The main obstacle for extending semiring provenance to such fields have been difficulties with handling negation. For a long time, semiring provenance has essentially been restricted to negation-free query languages, and although there have been algebraically interesting attempts to cover difference of relations [1, 10, 11, 15], they have not resulted in a systematic tracking of *negative information*. While there are many applications in databases where one can get quite far with using positive information only, logical applications in most other areas are based on formalisms that use negation in an essential way, often in combination with recursion or fixed-points.

Provenance semirings for logics with negation and recursion. This paper is part of a larger project with the objectives to

- develop semiring provenance systematically for a wide range of logics, including those featuring the notoriously difficult interaction between full negation and recursion,
- to employ algebraic methods for provenance analysis, in particular universal semirings of polynomials to obtain the most general provenance information,
- to exploit the connections between logics and various kinds of games and to use semiring valuations for an analysis of strategies in such games, and
- to explore practical applications of semiring provenance in new areas of logics in computer science, where this has not been used so far, such as knowledge representation, verification, and machine learning.

This project has been initiated in [13], where a provenance analysis of full first-order logic has been proposed. In this approach, negation is dealt with by transformation into negation normal form and, algebraically, by new provenance semirings of dual-indeterminate polynomials, which are obtained by taking quotients of traditional semirings of polynomials, such as $\mathbb{N}[X]$, by congruences generated by products of positive and negative provenance tokens, see Sect. 2 for details. In particular, the semiring $\mathbb{N}[X,\bar{X}]$ of dual-indeterminate polynomials is the most general provenance semiring for full first-order logic FO. These ideas have been used in [3, 4] to provide a provenance analysis of modal and guarded fragments of first-order logic, and to explore applications in description logic. Further, this approach has been applied to database repairs in [23], and it has been shown how this treatment of negation, or absent information, can be used to explain and repair missing query answers and the failure of integrity constraints in databases.

While the connection between provenance analysis of first-order logic and semiring valuations of games had only been hinted at in [13], it has then been developed more systematically in [14], first for games on acyclic graphs, which admit only finite plays, and then also for reachability games on acyclic game graphs. The latter are tightly connected with least fixed-point inductions, used positively. Combining the approach from [13] with the provenance analysis of least fixed-point inductions in ω-continuous semirings of formal power series, one obtains, by an analogous quotient construction, the semiring $\mathbb{N}^{\infty}[\![X,\bar{X}]\!]$ of dual-indeterminate power series [14]. This is the most general provenance semiring for Datalog with negated input predicates and, more generally, also for posLFP, the fragment of the full fixed-point logic LFP that consists of formulae in negation normal form such that all its fixed-point operators are least fixed-points. This is a powerful fixed-point calculus, which suffices to capture all polynomial-time computable properties of ordered finite structures [12]. An important simplification of dealing with posLFP is that the game-based analysis of model checking only requires reachability games rather than the much more complicated parity games that are needed for full LFP. At the end of [14] the problem of generalising semiring valuations and strategy analysis to infinite games with more general objectives than reachability has been discussed. In particular, a provenance approach for safety games has been proposed, with absorptive semirings as the central algebraic tools, and absorption-dominant strategies as a relevant game-theoretic notion.

Greatest fixed points. What has been missing so far, and what we want to provide in this paper, is an adequate and systematic treatment of greatest fixed points. There is a strong motivation for this: If provenance analysis should ever have an impact in fields such as verification (and we strongly believe it should) then dealing with greatest fixed points, e.g. for safety conditions or bisimulation, and with alternations between least and greatest fixed points is indispensable. The relevant formalisms in verification (such as LTL, CTL, μ-calculus etc.) are negation closed and based on both least and greatest fixed-points, with strict alternation hierarchies (even for finite structures), and without possibilities to eliminate greatest fixed-points. Even in finite model theory, where greatest fixed points can in principle be eliminated from LFP by means of the Stage Comparison Theorem [20, 12], it is usually not desirable to do so. Natural properties involving greatest fixed points (such as bisimilarity) would become very complicated to express, with the need to double the arity of the fixed-point variables. In addition, provenance valuations provide a refined semantics, and statements that are equivalent in the Boolean sense need not have the same provenance value. Therefore, we do not propose an approach that first tries to simplify formulae (e.g. by eliminating fixed-point alternations) and then computes semiring valuations for the translated formulae, but instead lay foundations of a provenance analysis for the general logics with arbitrary interleavings of least and greatest fixed points, such as full LFP or the modal μ-calculus (and for infinite games with more general objectives than reachability).

Provenance semirings for arbitrary fixed points. We first address the question, what kind of semirings are adequate for a meaningful and informative provenance analysis of unrestricted fixed point logics (Sect. 4). The common approach for dealing with least fixed point inductions, based on ω-continuous semirings and Kleene's Fixed-Point Theorem, is not sufficient to guarantee that both least and greatest fixed point are well-defined. Instead, we require that the semirings are *fully continuous* which means that every chain C has not only a supremum $\bigsqcup C$, but also an infimum $\bigsqcap C$, and that both semiring operations are compatible with these suprema and infima. For an informative provenance semantics, there is a second important condition that is connected with the symmetry between least and greatest fixed point computations. In the Boolean setting, fixed-point logic is based on complete lattices which are inherently symmetric. Moreover, conjunction and disjunction are dual in the sense that one leads to larger lattice elements while the other is decreasing. In the semiring setting, we compute fixed points with respect to the natural order induced by addition. The only constraint that relates this order with multiplication is distributivity, but this alone does not suffice to ensure a similar duality. We achieve this by requiring that the semiring is *absorptive*. This means that $a + ab = a$ for all a, b, and we shall see that this is equivalent with 1 being the greatest element or with multiplication being decreasing, giving us the desired duality with 0 and addition. As a result, absorptive and fully continuous semirings guarantee a well-defined and informative provenance semantics for arbitrary fixed-point formulae.

Generalized absorptive polynomials. For a most general provenance analysis, we further want the semiring semantics to be *truth-preserving*, which means that it gives non-zero values to true formulae. In positive semirings, this is guaranteed if infima of non-zero values are also non-zero, which we call *chain-positivity*. Our fundamental examples of absorptive, fully continuous, and chain-positive semirings are the semirings $\mathbb{S}^\infty[X]$ of *generalized absorptive polynomials* and its dual-indeterminate version $\mathbb{S}^\infty[X,\bar{X}]$, as introduced in [14]. Informally such a polynomial is a sum of monomials, with possibly infinite exponents, that are maximal with respect to absorption. For instance a monomial $x^2y^\infty z$ occurring in a provenance value $\pi[\![\psi]\!]$ indicates an absorption-dominant evaluation strategy that uses the atom labelled by x twice, the atom labelled by y an infinite number of times, and the atom labelled by z once. This monomial absorbs all those that have larger exponents for all variables, such as for instance $x^3y^\infty z^\infty u$, but not, say, $x^\infty y^3$. Absorptive polynomials thus describe model-checking proofs or evaluation strategies with a minimal use of atomic facts. A precise definition and analysis of these semirings will be given in Sect. 5. We prove that they do indeed have universal properties (see Theorem 17) that make $\mathbb{S}^\infty[X,\bar{X}]$ the most general absorptive semiring for LFP and thus also an indispensable tool to prove general results about provenance semantics in absorptive, fully continuous semirings.

Game-theoretic analysis. In the final Sect. 6 we illustrate the power of provenance interpretations for LFP in absorptive, fully-continuous semirings, and particularly in $\mathbb{S}^\infty[X,\bar{X}]$ by relating them to provenance values of plays and strategies in the associated model-checking games which in this case are parity games. Specifically we prove that, as in the case of FO and posLFP, the provenance value of an LFP-formula φ gives precise information on the evaluation strategies in these games.

2 Preliminaries: Commutative Semirings

▶ **Definition 1.** A *commutative semiring* is an algebraic structure $(K,+,\cdot,0,1)$, with $0 \neq 1$, such that $(K,+,0)$ and $(K,\cdot,1)$ are commutative monoids, $\cdot$ distributes over $+$, and $0 \cdot a = a \cdot 0 = 0$. It is *naturally ordered* if the relation $a \leq b :\Longleftrightarrow a + c = b$ for some $c \in K$ is a partial order. Further, a commutative semiring is *positive* if $a + b = 0$ implies $a = 0$ and $b = 0$ and if it has no divisors of 0 (i.e., $a \cdot b = 0$ implies that $a = 0$ and $b = 0$).

All semirings considered in this paper are commutative and naturally ordered (which excludes rings). In the following we just write "semiring" to denote a commutative, naturally ordered semiring. Standard semirings considered in provenance analysis are in fact also positive, but for an appropriate treatment of negation we need semirings (of dual-indeterminate polynomials or power series) that have divisors of 0.

Elements of semirings will be used as truth values for logical statements. The intuition is that $+$ describes the *alternative use* of information, as in disjunctions or existential quantifications, whereas $\cdot$ stands for the *joint use* of information, as in conjunctions or universal quantifications. Further, 0 is the value of false statements, whereas any element $a \neq 0$ of a semiring K stands for a "nuanced" interpretation of true. We briefly discuss some specific semirings that provide interesting information about a logical statement.

- The *Boolean semiring* $\mathbb{B} = (\{0,1\},\vee,\wedge,0,1)$ is the standard habitat of logical truth.
- $\mathbb{N} = (\mathbb{N},+,\cdot,0,1)$ is used for counting evaluation strategies for a logical statement.
- $\mathbb{T} = (\mathbb{R}_+^\infty,\min,+,\infty,0)$ is called the *tropical* semiring. It can be used for measuring the cost of evaluation strategies.
- The *Viterbi* semiring $\mathbb{V} = ([0,1],\max,\cdot,0,1)$ is used to compute *confidence scores* for logical statements. It is in fact isomorphic to $\mathbb{T}$.
- The *min-max* semiring on a totally ordered set $(A,\leq)$ with least element a and greatest element b is the semiring $(A,\max,\min,a,b)$.

Beyond these *application semirings*, (most general) abstract provenance can be calculated in freely generated (universal) *provenance semirings* of polynomials or formal power series. The abstract provenance can then be specialised via homomorphisms to provenance values in different application semirings as needed.

- For any set X, the semiring $\mathbb{N}[X] = (\mathbb{N}[X], +, \cdot, 0, 1)$ consists of the multivariate polynomials in indeterminates from X with coefficients from $\mathbb{N}$. This is the commutative semiring freely generated by X. Admitting also infinite sums of monomials we obtain the semiring $\mathbb{N}^\infty[\![X]\!]$ of formal power series over X, with coefficients in $\mathbb{N}^\infty := \mathbb{N} \cup \{\infty\}$.
- Given two disjoint sets $X, \bar{X}$ of "positive" and "negative" provenance tokens, together with a one-to-one correspondence $X \leftrightarrow \bar{X}$, mapping each positive token x to its corresponding negative token $\bar{x}$, the semiring $\mathbb{N}[X, \bar{X}]$ is the quotient of the semiring of polynomials $\mathbb{N}[X \cup \bar{X}]$ by the congruence generated by the equalities $x \cdot \bar{x} = 0$ for all $x \in X$. This is the same as quotienting by the ideal generated by the polynomials $x\bar{x}$ for all $x \in X$. The congruence classes in $\mathbb{N}[X, \bar{X}]$ are in one-to-one correspondence with the polynomials in $\mathbb{N}[X \cup \bar{X}]$ such that none of their monomials contain complementary tokens. We call these *dual-indeterminate polynomials*. $\mathbb{N}[X, \bar{X}]$ is freely generated by $X \cup \bar{X}$ for homomorphisms such that $h(x) \cdot h(\bar{x}) = 0$. By a completely analogous quotient construction, we obtain the semiring $\mathbb{N}^\infty[\![X, \bar{X}]\!]$ of dual-indeterminate power series.

3 Provenance Semantics for Fixed-Point Logic

Semiring provenance is well understood for first-order logic and for logics with only least fixed points, used positively. To extend it to logics with arbitrary interleavings of least and greatest fixed points, we discuss the general fixed-point logic LFP that extends first-order logic by least and greatest fixed-point operators, but our insights also apply to weaker logics such as the modal μ-calculus, dynamic logics, or temporal logics such as CTL.

Least Fixed-Point Logic. Least fixed-point logic, denoted LFP, extends first order logic by least and greatest fixed points of definable monotone operators on relations: If $\psi(R, \mathbf{x})$ is a formula of vocabulary $\tau \cup \{R\}$, in which the relational variable R occurs only positively and the length of $\mathbf{x}$ matches the arity of R, then $[\mathbf{lfp}\, R\mathbf{x} \,.\, \psi](\mathbf{x})$ and $[\mathbf{gfp}\, R\mathbf{x} \,.\, \psi](\mathbf{x})$ are also formulae (of vocabulary τ). The semantics of these formulae is that $\mathbf{x}$ is contained in the least (respectively the greatest) fixed point of the update operator $F_\psi : R \mapsto \{\mathbf{a} : \psi(R, \mathbf{a})\}$. Due to the positivity of R in ψ, any such operator F_ψ is monotone and has, by the Knaster-Tarski-Theorem, a least fixed point $\mathbf{lfp}(F_\psi)$ and a greatest fixed point $\mathbf{gfp}(F_\psi)$. See e.g. [12] for background on LFP. The duality between least and greatest fixed points implies that $[\mathbf{gfp}\, R\mathbf{x} \,.\, \psi](\mathbf{x}) \equiv \neg[\mathbf{lfp}\, R\mathbf{x} \,.\, \neg\psi[R/\neg R]](\mathbf{x})$. By this duality together with de Morgan's laws, every LFP-formula can be brought into *negation normal form*, where negation applies to atoms only. The fragment posLFP of LFP consists of the formulae in negation normal form in which all fixed-point operators are least fixed points. It is well-known that LFP, and even posLFP, captures all polynomial-time computable properties of ordered finite structures [12].

Provenance Semantics. Instead of truth-values, we now assign semiring values to literals. For a finite universe A and a finite relational vocabulary τ we denote the set of atoms as $\mathrm{Atoms}_A(\tau) = \{R\mathbf{a} \colon R \in \tau,\ \mathbf{a} \in A^{\mathrm{arity}(R)}\}$. The set $\mathrm{NegAtoms}_A(\tau)$ contains all negations $\neg R\mathbf{a}$ of atoms in $\mathrm{Atoms}_A(\tau)$ and we define the set of τ-literals on A as

$$\mathrm{Lit}_A(\tau) := \mathrm{Atoms}_A(\tau) \cup \mathrm{NegAtoms}_A(\tau) \cup \{a = b \colon a, b \in A\} \cup \{a \neq b \colon a, b \in A\}.$$

▶ **Definition 2.** For any semiring K, a K-*interpretation* (for A and τ) is a function $\pi\colon \mathrm{Lit}_A(\tau) \to K$ mapping true equalities and inequalities to 1 and false ones to 0.

For a finite universe A, we can extend K-interpretations π to provide provenance values $\pi[\![\varphi]\!]$ for any first-order formula φ in a natural way [13], by interpreting disjunctions and existential quantification via addition, and conjunctions and universal quantification via multiplication. Negation is not interpreted directly by an algebraic operation. We deal with it syntactically, by evaluating the negation normal form $\mathrm{nnf}(\psi)$ instead. To interpret fixed-point formulae $[\mathbf{lfp}\, R\mathbf{x}\,.\,\psi](\mathbf{a})$ and $[\mathbf{gfp}\, R\mathbf{x}\,.\,\psi](\mathbf{a})$, we generalize the update operators F_ψ to semiring semantics. If R has arity m, then its K-interpretations on A are functions $g : A^m \to K$. These functions are ordered, by $g \le g'$ if, and only if, $g(\mathbf{a}) \le g'(\mathbf{a})$ for all $\mathbf{a} \in A^m$ (recall that our semirings are naturally ordered). Given a K-interpretation $\pi\colon \mathrm{Lit}_A(\tau) \to K$, we denote by $\pi[R \mapsto g]$ the K-interpretation of $\mathrm{Lit}_A(\tau) \cup \mathrm{Atoms}_A(\{R\})$ obtained from π by adding values $g(\mathbf{c})$ for the atoms $R\mathbf{c}$. (Notice that R appears only positively in φ, so negated R-atoms are not needed). The formula $\varphi(R,\mathbf{x})$ now defines, together with π, a monotone update operator F_π^φ on functions $g : A^m \to K$. More precisely, it maps g to the function

$$F_\pi^\varphi(g)\colon\ \mathbf{a} \mapsto \pi[R \mapsto g][\![\varphi(R,\mathbf{a})]\!].$$

We obtain a well-defined provenance semantics for LFP if we can make sure that the update operators F_π^φ have least and greatest fixed points $\mathbf{lfp}(F_\pi^\varphi)$, $\mathbf{gfp}(F_\pi^\varphi)\colon A^m \to K$. However, this is not guaranteed in all semirings, and also the common approach to least fixed-point inductions based on ω-continuous semirings is not sufficient here, as these, in general, do not guarantee the existence of *greatest* fixed points. This raises the fundamental question: which semirings are really appropriate for LFP? We shall discuss this in detail in the next section. Once we have fixed a notion of appropriate semirings for LFP, we obtain a provenance semantics for LFP as follows.

▶ **Definition 3.** A K-interpretation $\pi\colon \mathrm{Lit}_A(\tau) \to K$ (for finite A and τ) in an *appropriate* semiring K extends to a K-*valuation* $\pi\colon \mathrm{LFP}(\tau) \to K$ by mapping an LFP-sentence $\psi(\mathbf{a})$ to a value $\pi[\![\psi]\!]$ using the following rules

$$\pi[\![\psi \vee \varphi]\!] := \pi[\![\psi]\!] + \pi[\![\varphi]\!] \qquad \pi[\![\psi \wedge \varphi]\!] := \pi[\![\psi]\!] \cdot \pi[\![\varphi]\!] \qquad \pi[\![\exists x \psi(x)]\!] := \sum_{a \in A} \pi[\![\varphi(a)]\!]$$

$$\pi[\![\forall x \psi(x)]\!] := \prod_{a \in A} \pi[\![\varphi(a)]\!] \qquad \pi[\![[\mathbf{lfp}\, R\mathbf{x}.\varphi(R,\mathbf{x})](\mathbf{a})]\!] := \mathbf{lfp}(F_\pi^\varphi)(\mathbf{a})$$

$$\pi[\![\neg\psi]\!] := \pi[\![\mathrm{nnf}(\neg\psi)]\!] \qquad \pi[\![[\mathbf{gfp}\, R\mathbf{x}.\varphi(R,\mathbf{x})](\mathbf{a})]\!] := \mathbf{gfp}(F_\pi^\varphi)(\mathbf{a}).$$

We remark that there is an important difference between the classical Boolean semantics and provenance semantics concerning the relationship of fixed-point logics with first-order logic. The (Boolean) evaluation of a fixed-point formula on a finite structure is computed by fixed-point inductions that terminate after a polynomial number of stages (with respect to the size of the structure). Hence, on any fixed finite universe, a fixed-point formula can be unraveled to an equivalent first-order formula. This is not the case for the provenance valuations in infinite semirings. Even for very simple Datalog queries, a fixed-point induction need not terminate after a finite number of steps. Provenance valuations provide more information than just the truth or falsity of a statement, and in a general setting, this provenance information, for instance about the number and properties of successful evaluation strategies, may also be infinite.

4 Semirings for Fixed-Point Logic

Given a naturally ordered semiring K, a *chain* is a totally ordered subset $C \subseteq K$. For $\circ \in \{+, \cdot\}$ we write $a \circ C$ for $\{a \circ c \mid c \in C\}$. Provided they exist, we write $\bigsqcup C$ and $\bigsqcap C$ for the *supremum* (least upper bound) and *infimum* (greatest lower bound) of $C \subseteq K$, and further $\bot$ and $\top$ for the least and greatest elements of K. We say that a function $f : K_1 \to K_2$ is fully chain-continuous or, for short, *fully continuous* if it preserves suprema and infima of nonempty chains, i.e., $f(\bigsqcup C) = \bigsqcup f(C)$ and $f(\bigsqcap C) = \bigsqcap f(C)$ for all chains $\varnothing \neq C \subseteq K_1$.

▶ **Definition 4.** A naturally ordered semiring K is *fully chain-complete* if every chain $C \subseteq K$ has a supremum $\bigsqcup C$ and an infimum $\bigsqcap C$ in K. It is additionally *fully continuous* if its operations are fully continuous in both arguments, i.e., $a \circ \bigsqcup C = \bigsqcup (a \circ C)$ and $a \circ \bigsqcap C = \bigsqcap (a \circ C)$ for all $a \in K$, chains $\varnothing \neq C \subseteq K$ and $\circ \in \{+, \cdot\}$.

Examples of fully continuous semirings include the Viterbi semiring, the semiring $\mathbb{N}^\infty$ of natural numbers extended by infinity, and semirings of formal power series $\mathbb{N}^\infty[\![X]\!]$ and $\mathbb{N}^\infty[\![X, \bar{X}]\!]$. For positive least fixed-point inductions, as in Datalog [16] or posLFP [14], the common approach is to use *ω-continuous* semirings. There, only suprema of ω-chains are required and both operations must preserve suprema. It would be tempting to work with a minimal generalization that imposes similar properties for descending ω-chains, using a dual version of Kleene's Fixed-Point Theorem. However, the following example shows that this approach will not work in general with alternating fixed points.

▶ **Example 5.** Let K be a naturally ordered semiring that has both suprema of ascending ω-chains and infima of descending ω-chains and let $f : K \times K \to K$ be a function that preserves these suprema and infima in each argument. For each $x \in K$, we can consider the function $g_x : K \to K$, $g_x(y) = f(x, y)$ and, further, the function $G : K \to K$, $G(x) = \mathbf{gfp}(g_x)$. Note that G is well-defined due to the preservation property of f and a dual version of Kleene's Fixed-Point Theorem. Now consider $\mathbf{lfp}(G)$. To guarantee the existence of this fixed point via Kleene's theorem, G has to preserve suprema of ω-chains. This is, however, not the case, in general. One counterexample is the the function $f(x, y) = x \diamond y$ in the (fully continuous) Łukasiewicz semiring $\mathbb{L} = ([0, 1], \max, \diamond, 0, 1)$ with $a \diamond b = \max(0, a + b - 1)$ on the ω-chain $(x_n)_{n<\omega}$ defined by $x_n = 1 - \frac{1}{1+n}$. Then $G(\bigsqcup_{n<\omega} x_n) = G(1) = \mathbf{gfp}(g_1) = 1$, whereas $\bigsqcup_{n<\omega} G(x_n) = \bigsqcup_{n<\omega} \mathbf{gfp}\left(g_{x_n}\right) = \bigsqcup_{n<\omega} 0 = 0$. Hence Kleene's theorem is not applicable (although the least fixed point exists with $\mathbf{lfp}(G) = 0$). ⌟

Instead, we rely on K being fully chain-complete to guarantee the existence of fixed points of monotone functions. We can then extend [20] the Kleene iteration $\bot$, $f(\bot)$, $f^2(\bot)$, $f^3(\bot)$, ... for $\mathbf{lfp}(f)$ to a transfinite fixed-point iteration $(x_\beta)_{\beta \in \mathrm{On}}$ by setting $x_0 = \bot$, $x_{\beta+1} = f(x_\beta)$ for ordinals β and $x_\lambda = \bigsqcup \{x_\beta \mid \beta < \lambda\}$ for limit ordinals λ. If f is monotone, this iteration forms a chain and is well-defined due to the chain-completeness of K. The iteration for $\mathbf{gfp}(f)$ can be defined analogously by $x_\lambda = \bigsqcap \{x_\beta \mid \beta < \lambda\}$ for limit ordinals and it follows that both $\mathbf{lfp}(f)$ and $\mathbf{gfp}(f)$ exist for any monotone function $f : K \to K$ on a fully chain-complete semiring K. Coming back to the question of appropriate semirings for LFP, we observe that the update operators F_π^φ are always monotone (this can be seen by structural induction on φ, using the monotonicity of $\cdot$ and $+$).

▶ **Theorem 6.** *Semiring semantics for* LFP *is well-defined for fully chain-complete semirings.*

We further remark that full chain-completeness is more general than the common notion of complete lattices, used in the Knaster-Tarski fixed-point theory, as we only require suprema (and infima) of chains instead of arbitrary sets. However, based on results in [19] it follows that every idempotent, fully chain-complete semiring is in fact a complete lattice (under its natural order); this applies in particular to the absorptive, fully continuous semirings discussed below.

The following fundamental property for provenance analysis (cf. [13]) establishes a closer connection between logic (the semantics of φ) and algebra (the semiring homomorphism h) and enables us to compute provenance information in a general semiring and then specialize the result to application semirings by applying homomorphisms.

▶ **Proposition 7** (Fundamental Property). *Let K_1, K_2 be fully chain-complete semirings and let $h : K_1 \to K_2$ be a fully continuous semiring homomorphism with $h(\top) = \top$. Then for every K_1-interpretation π, the mapping $h \circ \pi$ is a K_2-interpretation and for every $\varphi \in \mathrm{LFP}$, we have $h(\pi[\![\varphi]\!]) = (h \circ \pi)[\![\varphi]\!]$. In diagrammatic form:*

$$\begin{tikzcd}
 & \mathrm{Lit}_A(\tau) \arrow[dl, "\pi"'] \arrow[dr, "h \circ \pi"] & & & & \mathrm{LFP} \arrow[dl, "\pi"'] \arrow[dr, "h \circ \pi"] & \\
K_1 \arrow[rr, "h"] & & K_2 & \Longrightarrow & K_1 \arrow[rr, "h"] & & K_2
\end{tikzcd}$$

Fully continuous semirings. While fully chain-complete semirings suffice to guarantee well-defined semantics, our main results (the universal property in Theorem 17 and the connection to games in Sect. 6) require the technically slightly stronger notion of fully continuous semirings, in which addition and multiplication preserve suprema and infima of chains. This is an adaption of the standard notion of ω-continuity to our setting and all natural examples of fully chain-complete semirings we are aware of are in fact fully continuous. On a different note, the notion of chain-completeness is based on chains of arbitrary length. We do not know whether working with ascending and descending ω-chains would suffice in all cases, but we show in Sect. 5 that it suffices in absorptive, fully continuous semirings.

Absorptive and chain-positive semirings. Although the *existence* of fixed points is guaranteed in fully continuous semirings, we observe (in Example 10 below) that one may have valuations of greatest fixed-point formulae in such semirings that are not really informative and do not provide useful insights why a formula holds. This can be tied to two separate problems: the *lack of symmetry* between least and greatest fixed-point inductions in some such semirings, and the fact that such semirings are not necessarily *truth-preserving*, i.e. they may evaluate true statements to 0. To deal with these problems we propose to work with fully continuous semirings that are *absorptive*, to provide useful provenance information for greatest fixed points, and *chain-positive*, to guarantee truth-preservation.

We first address the issue of symmetry between least and greatest fixed points. In the Boolean setting, these are computed in the complete lattice of subsets which is inherently symmetric. For instance, a greatest fixed point of a monotone operator is the complement of the least fixed point of the dual operator (which is essential for a negation normal form). Moreover, conjunction and disjunction are symmetric in the sense that one increases values, acting as set union in the lattice of subsets, while the other is decreasing. In the semiring setting, we compute fixed points with respect to the natural order induced by addition. This order is always a complete lattice in absorptive semirings (in fact, idempotent semirings suffice) and it is clear that addition is increasing in the sense that $a + b \geq a$ for all a, b. The issue is with multiplication: The only constraint relating addition and multiplication is distributivity, but this alone does not ensure a symmetry similar to the Boolean setting. We achieve a sufficient degree of symmetry by requiring that the semiring is absorptive.

▶ **Definition 8.** A semiring K is *absorptive* if $a + ab = a$ for all $a, b \in K$, which is equivalent to saying that $1 + b = 1$, for all $b \in K$.

Clearly, every absorptive semiring is *idempotent*: $a+a = a$ for all a. For naturally ordered semirings, absorption indeed provides symmetry: multiplication becomes decreasing and 1 becomes the greatest element, symmetric to addition and the least element 0. We remark that while this adds a certain, fruitful degree of symmetry, it does not enforce complete symmetry of addition and multiplication (as in the Boolean setting). For instance, multiplication need not be idempotent. In particular, absorptive semirings need not be lattices (with $+$ and $\cdot$ as lattice operations), even if the natural order is always a complete lattice.

▶ **Proposition 9.** *In a naturally ordered semiring K, the following are equivalent:*

1. *K is absorptive,*
2. *K has the greatest element $\top = 1$, i.e., $a \leq 1$ for all $a \in K$,*
3. *multiplication in K is decreasing, i.e., $a \cdot b \leq b$ for all $a, b \in K$.*

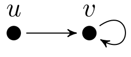
This symmetry helps, for instance, to avoid problems of increasing multiplication as in $\mathbb{N}^\infty$. Fixed-point theory often relies on symmetry and it is thus no surprise that more symmetry leads to more useful provenance information. This can be seen in Example 10 below when comparing the computations of greatest fixed points in the non-absorptive semiring $\mathbb{N}^\infty$ and the more informative Viterbi semiring.

A further motivation for absorptive semirings is that they give information about reduced proofs of a formula. The property $a + ab = a$ implies, for example, that a proof containing two literals mapped to a and b, thus having the value ab, is absorbed by a proof only using one literal, with provenance value a. This has the further benefit that, unlike formal power series $\mathbb{N}^\infty[\![X]\!]$, provenance information is always finitely representable (see Sect. 5).

We next address the issue of truth-preservation, which is defined as follows. As in [13], we say that a K-interpretation $\pi\colon \mathrm{Lit}_A(\tau) \to K$ is *model-defining* if for all atoms $R\mathbf{a}$ exactly one of the two values $\pi[\![R\mathbf{a}]\!]$ and $\pi[\![\neg R\mathbf{a}]\!]$ is zero. A model-defining K-interpretation induces a unique structure $\mathfrak{A}_\pi$ with universe A and $\mathbf{a} \in R^{\mathfrak{A}}$ if, and only if, $\pi(R\mathbf{a}) \neq 0$. For a truthful provenance analysis for a logic L, this should lift to arbitrary sentences $\varphi \in L$, so that $\mathfrak{A}_\pi \models \varphi$ if, and only if, $\pi[\![\varphi]\!] \neq 0$. If this is indeed the case for all model-defining K-interpretations π, we say that K is *truth-preserving* for L. This is illustrated in the following example.

▶ **Example 10.** The existence of an infinite path from u in a graph G is expressed by the LFP-formula

$$\varphi(u) = [\mathbf{gfp}\, Rx\, .\, \exists y(Exy \wedge Ry)](u)$$



For the Boolean semiring $\mathbb{B} = \{0, 1\}$ there is a unique $\mathbb{B}$-interpretation π that defines the displayed graph G. Provenance semantics in $\mathbb{B}$ coincides with standard semantics and we indeed obtain $\pi[\![\varphi(u)]\!] = 1$. The Viterbi semiring $\mathbb{V}$ instead allows us to assign confidence scores to the edges. If we set $\pi(Euv) = \pi(Evv) = 1$ as in the Boolean interpretation, we again obtain an overall confidence of $\pi[\![\varphi(u)]\!] = 1$. However, if we instead lower the score of the self-loop to $\pi(Evv) = 1 - \varepsilon$, we obtain an overall confidence of $\pi[\![\varphi(u)]\!] = 0$ due to the fixed-point iteration $1, 1 - \varepsilon, (1 - \varepsilon)^2, \ldots$. So while π still defines the model shown above, the formula evaluates to 0 which we usually interpret as *false*, illustrating that the Viterbi semiring is *not truth-preserving.* Since the loop occurs infinitely often in the unique infinite path from u, the value 0 makes sense as a confidence score. Thus, although it is not truth-preserving, the Viterbi semiring does provide useful information.

Consider next the semiring of formal power series $\mathbb{N}^\infty[\![X]\!]$. If we choose $\pi(Euv) = x$ and $\pi(Evv) = y$ (and keep the values 0 or 1 for the remaining literals), then $\pi[\![\varphi(u)]\!] = 0$, as result of the infinite iteration $\top$, $y \cdot \top$, $y^2 \cdot \top$, $y^3 \cdot \top$, $\dots$ with infimum 0 at node v (here, $\top$ is the power series in which all monomials have coefficient ∞). Thus, the semiring $\mathbb{N}^\infty[\![X]\!]$ is *not truth-preserving* either.

In the semiring $\mathbb{N}^\infty$, used to count proofs of formulae in FO and posLFP, the consideration of greatest fixed points imposes problems: Intuitively, the graph only has one infinite path that we would view as a proof of $\varphi(u)$. But setting $\pi(Euv) = \pi(Evv) = 1$ results in $\pi[\![\varphi(u)]\!] = \infty$, since the iteration for the evaluation of φ at v is ∞, $1 \cdot \infty$, $1 \cdot \infty$, $\dots$ which stagnates immediately. Although $\mathbb{N}^\infty$ is truth-preserving, the example hints at another general issue: Multiplication with non-zero values in $\mathbb{N}^\infty$ always increases values. The same is true for addition, so fixed-point iterations of **gfp**-formula are likely to result in ∞ and *do not give meaningful provenance information*, e.g. about the number of proofs. Since the computation in $\mathbb{N}^\infty[\![X]\!]$ yields 0, we further see that we cannot obtain the result in $\mathbb{N}^\infty$ from the computation in $\mathbb{N}^\infty[\![X]\!]$ by polynomial evaluation. Hence evaluation of formal power series *does not preserve provenance semantics* in general. This is a further reason why formal power series are not the right provenance semirings for LFP. ⌟

We shall define and investigate in the next section the semiring of generalized absorptive polynomials $\mathbb{S}^\infty[X]$ which, contrary to other fully continuous and absorptive semirings, is truth-preserving due to the following algebraic property.

▶ **Definition 11.** A fully chain-complete semiring K is *chain-positive* if for each non-empty chain $C \subseteq K$ of non-zero elements, the infimum $\bigsqcap C$ is non-zero as well.

▶ **Proposition 12.** *Every chain-positive, positive semiring is truth-preserving for* LFP*.*

Chain-positivity is not an indispensable requirement for provenance analysis, as shown by the Viterbi semiring (which is absorptive and fully continuous). However, we need this property for provenance semirings which should give insights into proofs or evaluation strategies and thus have to preserve truth.

5 Generalized Absorptive Polynomials

We now discuss the semirings $\mathbb{S}^\infty[X]$ and $\mathbb{S}^\infty[X, \bar{X}]$ of generalized absorptive polynomials. They were introduced in [14] and generalize the semiring of absorptive polynomials $\mathsf{Sorp}(X)$ from [6] by admitting exponents in $\mathbb{N}^\infty$ to guarantee chain-positivity. We show that these semirings are, in a well-defined sense, the most general absorptive, fully continuous semirings and we argue that $\mathbb{S}^\infty[X, \bar{X}]$ is the right provenance semiring for LFP.

▶ **Definition 13.** Let X be a *finite* set of provenance tokens. We generalize the notion of a monomial over X to admit exponents from $\mathbb{N}^\infty$. Monomials are here functions $m : X \to \mathbb{N}^\infty$, written $x_1^{m(x_1)} \cdots x_n^{m(x_n)}$. Multiplication adds the exponents, and $x^\infty \cdot x^n = x^\infty$. We say that m_2 *absorbs* m_1, denoted $m_2 \succeq m_1$, if m_2 has smaller exponents than m_1, i.e., $m_2(x) \le m_1(x)$ for all $x \in X$. This is the pointwise partial order given by the reverse order on $\mathbb{N}^\infty$.

The set of monomials inherits a lattice structure from $\mathbb{N}^\infty$ and is, of course, infinite. However, it has some crucial finiteness properties.

▶ **Proposition 14.** *Every antichain of monomials is finite. Further, while there are infinitely descending chains of monomials, such as $1 = x^0 \succ x^1 \succ x^2 \succ \dots$ there are no infinitely ascending such chains.*

Indeed, $(\mathbb{N}^\infty, \leq)$ is a well-order. The set of monomials $m : X \to \mathbb{N}^\infty$ with the *reverse order* of the absorption order is isomorphic to $(\mathbb{N}^\infty)^k$ with $k = |X|$ and with the component-wise order inherited from $(\mathbb{N}^\infty, \leq)$. This is a well-quasi-order and therefore has no infinite descending chains and no infinite antichains. This implies that in the set of monomials over X with the absorption order, all ascending chains and all antichains are finite.

▶ **Definition 15.** We define $\mathbb{S}^\infty[X]$ as the set of antichains of monomials with indeterminates from X and exponents in $\mathbb{N}^\infty$. We write such antichains as formal sums of their monomials and call them *(generalized absorptive) polynomials.* Addition and multiplication of polynomials proceed as usual, but keeping only the maximal monomials (w.r.t. $\succeq$) in the result (and disregarding coefficients).

Since antichains of monomials are finite, there is no difference between polynomials and power series here. The natural order on $\mathbb{S}^\infty[X]$ can be characterized by monomial absorption: $P \leq Q$ if, and only if, for each $m \in P$ there is $m' \in Q$ with $m' \succeq m$. With Proposition 14, it follows that there are no infinitely ascending chains of polynomials, and further that the supremum of $S \subseteq \mathbb{S}^\infty[X]$ is $\bigsqcup S = \text{maximals}\,(\bigcup S)$ which is the set of $\succeq$-maximal monomials in $\bigcup S$. Due to the exponent ∞ and the finiteness of X, there is a smallest monomial $m_\infty \neq 0$ with $m_\infty(x) = \infty$ for all $x \in X$. This ensures chain-positivity of $\mathbb{S}^\infty[X]$.

▶ **Proposition 16.** $(\mathbb{S}^\infty[X], +, \cdot, 0, 1)$ *is absorptive, fully continuous, and chain-positive.*

The central property of $\mathbb{S}^\infty[X]$ is the following universal property which says that it is the absorptive, fully continuous semiring freely generated by X w.r.t. fully continuous homomorphisms. These homomorphisms enable us to apply the fundamental property.

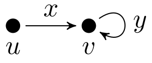
▶ **Theorem 17** (Universality)**.** *Every mapping $h : X \to K$ into an absorptive, fully continuous semiring K uniquely extends to a fully continuous semiring homomorphism $h : \mathbb{S}^\infty[X] \to K$.*

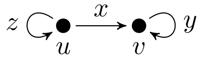
In absorptive semirings, the powers of an element a form a descending ω-chain $1 \geq a \geq a^2 \geq \cdots$ whose infimum we denote by a^∞. Since we want h to be fully continuous, the mapping $h(x) = a$ implies $h(x^\infty) = h(\bigsqcap_n x^n) = \bigsqcap_n h(x)^n = a^\infty$. By similar arguments, it is straightforward to see that h is uniquely defined. The nontrivial part of the proof is that the induced homomorphism h is fully continuous. Ascending chains are finite and thus impose no difficulties, so what remains is to prove that $h(\bigsqcap C) = \bigsqcap h(C)$ for chains $C \neq \varnothing$. Our proof constructs from $\bigsqcap C$ a canonical chain and makes use of Kőnig's lemma (recall that polynomials are finite) to relate the original chain C to the canonical chain, in a way that is preserved by h (see Appendix A for the full proof).

The fact that the universal property guarantees fully continuous homomorphisms should not be taken lightly: We have seen in Example 10 that this is not the case for formal power series. There, polynomial evaluation induces homomorphisms that are, in general, not fully continuous and hence do not preserve greatest fixed points. The following example shows how we can specialize provenance values in $\mathbb{S}^\infty[X]$ to application semirings.

▶ **Example 18.** We recall the setting from Example 10 and first consider the model-defining $\mathbb{S}^\infty[X]$-interpretation tracking the two edges labelled x and y, as indicated in the left graph.

$$\varphi(u) = [\mathbf{gfp}\, Rx\,.\, \exists y(Exy \wedge Ry)](u)$$





We obtain $\pi[\![\varphi(u)]\!] = xy^\infty$ corresponding to the infinite path $uvvv\ldots$. The confidence values from Example 10 can be obtained by polynomial evaluation: For $h(x) = h(y) = 1$, we get $(h\circ\pi)[\![\varphi(u)]\!] = 1\cdot 1^\infty = 1$ and for $h'(x) = 1$, $h'(y) = 1-\varepsilon$ we get $(h'\circ\pi)[\![\varphi(u)]\!] = 1\cdot(1-\varepsilon)^\infty = 0$.

We next consider the graph on the right by setting $\pi(Euu) = z$. There are now infinitely many infinite paths from u to v. However, we obtain only finitely many monomials due to absorption: $\pi[\![\varphi(u)]\!] = xy^\infty + z^\infty$. These correspond to the *simplest* infinite paths since monomials such as z^2xy^∞ (corresponding to the path $uuuvvv\dots$) are absorbed by xy^∞. ⌟

One consequence of the universal property is the existence of a *most general* $\mathbb{S}^\infty[X]$-*interpretation* π_0 by introducing variables $X = \{x_L \mid L \in \text{Atoms}_A(\tau) \cup \text{NegAtoms}_A(\tau)\}$ for all literals and setting $\pi_0(L) = x_L$. Any other K-interpretation π (where K is fully continuous and absorptive) results from π_0 by the evaluation $x_L \mapsto \pi(L)$ which lifts to a fully continuous homomorphism h. After computing $\pi_0[\![\varphi]\!]$ once, the computation for any K-interpretation π is then simply a matter of applying polynomial evaluation, since $\pi[\![\varphi]\!] = h(\pi_0[\![\varphi]\!])$.

The most general $\mathbb{S}^\infty[X]$-interpretation can also be used to prove that the update operators F^φ_π induced by LFP-formulae in $\mathbb{S}^\infty[X]$ are fully continuous. Hence Kleene's Fixed-Point Theorem applies and guarantees that the fixed-point iterations for $\mathbf{lfp}(F^\varphi_\pi)$ and $\mathbf{gfp}(F^\varphi_\pi)$ have closure ordinal at most ω. Using the universal property, the statement on the closure ordinal generalizes to all absorptive, fully continuous semirings – even to semirings in which update operators are not continuous in general (such as the semiring $\mathbb{L}$ in Example 5).

▶ **Proposition 19.** *Given a K-interpretation π into an absorptive, fully continuous semiring, all fixed-point iterations for* $\mathbf{lfp}(F^\varphi_\pi)$ *and* $\mathbf{gfp}(F^\varphi_\pi)$ *have closure ordinal at most ω.*

What we still have to provide for an adequate provenance analysis is a proper treatment of negation: If we track a literal and its negation by different variables x and y, respectively, we may obtain inconsistent monomials such as xy. As in other semirings of polynomials and power series we can also here take pairs of positive and negative indeterminates, with a correspondence $X \leftrightarrow \bar{X}$, and build the quotient with respect to the congruence generated by the equation $x \cdot \bar{x} = 0$. We thus obtain a new semiring $\mathbb{S}^\infty[X, \bar{X}]$ which, as a quotient, retains the properties of being absorptive, fully continuous and chain-positive. Of course, $\mathbb{S}^\infty[X, \bar{X}]$ is no longer positive, as x and $\bar{x}$ are divisors of 0. Most importantly, $\mathbb{S}^\infty[X, \bar{X}]$ inherits the universal property: If $h : X \cup \bar{X} \to K$ respects dual-indeterminates, so $h(x) \cdot h(\bar{x}) = 0$ for all $x \in X$, then it extends uniquely to a fully continuous homomorphism $h : \mathbb{S}^\infty[X, \bar{X}] \to K$. Together with the fundamental property, $\mathbb{S}^\infty[X, \bar{X}]$ is thus the most general appropriate provenance semiring for LFP that can represent negation, hence providing a natural framework for a provenance analysis for LFP and other fixed-point calculi.

Instead of model-defining interpretations, we consider *model-compatible* interpretations π. That is, for each atom $R\mathbf{a}$ we either have $\pi(R\mathbf{a}) = x$ and $\pi(\neg R\mathbf{a}) = \bar{x}$, or $\{\pi(R\mathbf{a}), \pi(\neg R\mathbf{a})\} = \{0, 1\}$. Additionally, π must not use the same indeterminate for two different atoms. We say that a model $\mathfrak{A}$ is *compatible* with π if $\mathfrak{A} \models L$ for all literals L with $\pi(L) = 1$ and denote the set of compatible models by Mod_π. Model-compatible interpretations can be used to reason about several models at once. Mapping certain literals to indeterminate pairs x and $\bar{x}$ leaves open the truth of these literals, but still encodes the semantics of opposing literals:

▶ **Proposition 20.** *Let π be a model-compatible $\mathbb{S}^\infty[X, \bar{X}]$-interpretation. An LFP-formula φ is Mod_π-satisfiable (Mod_π-valid) if, and only if, $\pi[\![\varphi]\!] \neq 0$ ($\pi[\![\neg\varphi]\!] = 0$).*

6 Game-theoretic analysis

It has been shown in [14] that the provenance analysis for FO and posLFP is intimately connected with the provenance analysis of reachability games. Evaluation strategies to establish the truth of first-order formulae are really winning strategies for reachability games on acyclic game graphs. For posLFP the situation is similar, but the associated model-checking games may have cycles and thus admit infinite plays, but the winning plays for the verifying player have to reach a winning position (a true literal) in a finite number of steps. By annotating such terminal positions with semiring values and propagating these values along the edges to the remaining positions, one obtains provenance values that coincide with the syntactically defined semantics $\pi[\![\psi]\!]$.

For full LFP or the modal μ-calculus, the model-checking games are parity games which are considerably more complex and do not allow for a simple propagation of values from terminal positions. We do not present here a general provenance analysis of parity games, but we show how provenance values $\pi[\![\varphi]\!]$ for fixed-point formulae can be understood from a game-theoretic point of view. For first-order logic or posLFP, provenance values $\pi[\![\varphi]\!]$ in $\mathbb{N}[X, \bar{X}]$ or $\mathbb{N}^\infty[\![X, \bar{X}]\!]$ are sums of monomials that correspond to the evaluation strategies for φ and provide information about the literals used by these strategies. We present an analogue of this statement for full fixed-point logic and the semiring $\mathbb{S}^\infty[X, \bar{X}]$.

Model-checking games for LFP. Model-checking games are classically defined for a formula and a fixed structure $\mathfrak{A}$ (see e.g. [2, Chap. 4]). However, the *game graph* of such a game depends only on the formula ψ and the *universe* of the given structure, and it is only the labelling of the terminal positions as winning for either the Verifier (Player 0) or the Falsifier (Player 1), that depends on which of the literals in $\mathrm{Lit}_A(\tau)$ are true in $\mathfrak{A}$. Hence the definition readily generalizes to a more abstract provenance scenario where we instead label terminal positions by semiring values. As the definition of the model-checking game $\mathcal{G}(A, \psi)$ itself is standard, we refer to the full version [5] or [2] for details. Most importantly, positions in the game $\mathcal{G}(A, \psi)$ correspond to subformulae of ψ and terminal positions are literals in $\mathrm{Lit}_A(\tau)$. The game may have cycles that admit infinite plays which are won according to the *parity condition*: We assign to each fixed-point variable a priority, and an infinite play is then won by Verifier precisely if the least priority occurring infinitely often is even.

Provenance values for plays and strategies. Given a parity game $\mathcal{G}(A, \psi)$, every K-interpretation $\pi : \mathrm{Lit}_A(\tau) \to K$ provides a valuation of the terminal positions. Based on this, we define provenance values for plays and strategies.

▶ **Definition 21.** A *finite* play $\rho = (\varphi_0, \dots, \varphi_t)$ ends in a terminal position $\varphi_t \in \mathrm{Lit}_A(\tau)$ which we call the *outcome* of ρ. We simply identify the provenance value of ρ with the value of its outcome, i.e. we put $\pi[\![\rho]\!] \coloneqq \pi[\![\varphi_t]\!]$. For an *infinite* play ρ we put $\pi[\![\rho]\!] \coloneqq 1$ if ρ is a wining play for the Verifier, and $\pi[\![\rho]\!] \coloneqq 0$ otherwise.

We denote by $\mathrm{Strat}(\varphi)$ the set of evaluation strategies for the subformula φ of ψ, i.e. the set of all (not necessarily positional) strategies that the Verifier has from position φ in the parity game $\mathcal{G}(A, \psi)$. Every strategy $\mathcal{S} \in \mathrm{Strat}(\varphi)$ induces the set $\mathrm{Plays}(\mathcal{S})$ of plays that are consistent with $\mathcal{S}$. Intuitively, the provenance value of a strategy is simply the product over the provenance values of all plays that it admits. However, a strategy may well admit an infinite set of plays and while it is possible to define infinite products in our setting, we instead observe that the set of possible outcomes is of course finite, since there exist only finitely many literals. As a consequence, we define the provenance value for a strategy by grouping those plays with identical outcome.

▶ **Definition 22.** The *provenance value* of a strategy $\mathcal{S}$ is $\pi[\![\mathcal{S}]\!] \coloneqq \prod_{L \in \mathrm{Lit}_A(\tau)} \pi(L)^{\#_{\mathcal{S}}(L)}$ if all infinite $\rho \in \mathrm{Plays}(\mathcal{S})$ are winning for Verifier, and $\pi[\![\mathcal{S}]\!] \coloneqq 0$ otherwise. Here, $\#_{\mathcal{S}}(L) \in \mathbb{N} \cup \{\infty\}$ denotes the number of plays $\rho \in \mathrm{Plays}(\mathcal{S})$ with outcome L.

The case for $\#_{\mathcal{S}}(L) = \infty$ is well-defined, as the infinitary power $a^\infty = \bigsqcap_n a^n$ can be defined in all absorptive, fully continuous semirings. For model-compatible interpretations in $\mathbb{S}^\infty[X, \bar{X}]$, the value $\pi[\![\mathcal{S}]\!]$ is a single monomial. The following central result justifies our game-theoretic analysis and precisely characterizes provenance semantics $\pi[\![\psi]\!]$ in terms of strategies in the associated model-checking game.

▶ **Theorem 23.** *Let $\psi \in \mathrm{LFP}$, and and let $\pi\colon \mathrm{Lit}_A(\tau) \to K$ be a K-interpretation into an absorptive, fully continuous semiring K. Then $\pi[\![\psi]\!] = \bigsqcup\{\pi[\![\mathcal{S}]\!] \mid \mathcal{S} \in \mathrm{Strat}(\psi)\}$.*

Model-checking games become large even for simple formulae over a small universe A; we thus refer to Appendix B and [5, Sect. 6.1] for examples. The proof of this result [5, Sect. 6.2] is not short either. The key idea is to view strategies $\mathcal{S}$ in the game of, say, $[\mathbf{gfp}\, R\mathbf{x} \,.\, \varphi](\mathbf{a})$ as trees and then define prefixes $\mathcal{S}|_n$ of these trees based on the number of fixed-point literals $R\mathbf{b}$ along a path. We prove by induction that these prefixes of increasing size correspond exactly to the steps of the fixed-point iteration via F_π^φ. For greatest fixed points, strategies can be infinite which leads to subtle obstacles. Perhaps the most challenging step is the so-called *puzzle lemma* which shows that, roughly speaking, computing infima of strategy prefixes leads to meaningful values corresponding to actual (infinite) strategies.

Consider now specifically the semiring $\mathbb{S}^\infty[X, \bar{X}]$ and model-compatible interpretations. By the above theorem, the provenance value of a sentence ψ is then a sum of monomials $x_1^{e_1} \cdots x_k^{e_k}$, each of which corresponds to a strategy $\mathcal{S}$ for Verifier that uses precisely the literals labelled by $x_1, \ldots, x_k$, and each literal x_i is used precisely e_i many times, that is, there are e_i plays consistent with $\mathcal{S}$ that have outcome x_i. By using dual indeterminates, we make sure that these literals are consistent and hence represent actual evaluation strategies.

In this sense, provenance semantics in absorptive, fully continuous semirings, and most prominently in $\mathbb{S}^\infty[X, \bar{X}]$, provide detailed information about evaluation strategies. Because of absorption, we do not obtain information about all evaluation strategies, as in first-order logic and $\mathbb{N}[X, \bar{X}]$, but instead only about the absorption-dominant strategies, corresponding to absorption-maximal monomials. These are strategies that allow the fewest different possible outcomes and are thus the simplest or canonical evaluation strategies.

7 Related Work

While our approach and our general project, as outlined in the introduction, is rooted in the work on semiring provenance in databases, there have also been a number of other areas of logic in computer science where semiring semantics have been used.

A prominent instance is the work on weighted automata (see, e.g., the Handbook [8]). In particular, weighted automata over finite and infinite words are discussed in [7, 9], and their expressive power is related to weighted monadic second-order logic (MSO) on words. In this setting, the weight of a word is defined as the sum over the weights of accepting paths, and the overall behaviour of an automaton is described by a formal power series over a semiring. To deal with infinite words, infinite sum and product operations of the semiring are assumed [9], roughly comparable to our assumptions on suprema and infima. Whereas the power series assign semiring values to *words*, we instead use indeterminates to track (combinations of) *literals* which then provide us with provenance information. As we have seen, formal power series are not the right tool for this purpose when confronted with greatest fixed points, so we consider absorptive polynomials $\mathbb{S}^\infty[X]$ instead. Moreover, in our setting the sum-of-strategies characterization is not a definition, but a non-trivial result. The definition of weighted MSO is similar to our semiring semantics for LFP and is also based on negation normal form. Main differences are that only logics over words are considered, and that semiring values are part of the formulae, whereas we assign values to literals. This reflects the different point of view: Weighted MSO is used to define series recognizable by weighted automata, whereas our goal is the provenance analysis of the logic itself.

Lluch-Lafuente and Montanari [18] have studied a semantics of CTL and μ-calculus in so-called constraint semirings, to reason about issues of quality of service such as delay or bandwidth. The choice of constraint semirings is motivated by applications for a particular class of constraint satisfaction problems, called soft CSP, and by useful closure properties, such as closure under Cartesian products, exponentials, and power constructions. Although this is not mentioned explicitly, constraint semirings are in fact also absorptive and satisfy a continuity requirement for suprema. However, the approach to negation is different from ours, requiring the extension of the semiring by new functions, and they do not have an abstract approach on the basis of polynomials with universal properties and reasoning over multiple constraint semirings. A main result of [18] is that the usual embedding of CTL into the μ-calculus fails for this semantics, which is another instance showing that a refined semiring semantics may distinguish between formulae that are equivalent under Boolean semantics.

8 Conclusion and Outlook

Let us summarize the contributions of this paper: We have laid foundations for the semiring provenance analysis of full fixed-point logics, with arbitrary interleavings of least and greatest fixed points, as part of the general project of developing provenance semantics of logical languages used in various branches of computer science. We have seen that absorptive and fully continuous semirings provide an adequate framework for this. We have identified the semiring of dual-indeterminate generalized absorptive polynomials $\mathbb{S}^\infty[X,\bar{X}]$ as the "right" provenance semiring for LFP. It satisfies the further algebraic property of chain-positivity which guarantees that provenance interpretations are truth-preserving, and we have established an important universal property of this semiring. Finally, we have shown how provenance for LFP is related to strategies in model-checking games.

Next steps will include the specific analysis of important logics such as temporal logics, dynamic logics, the modal μ-calculus, description logics (see initial work in [3]) etc. Applications require in particular the study of *algorithms* for computing provenance values – a non-trivial task, considering that greatest fixed-point iterations in semirings such as $\mathbb{S}^\infty[X,\bar{X}]$ can be infinite. Nevertheless, absorption and the infinitary power a^∞ can be used to short-circuit these iterations; forthcoming work will include results that show how an effective, and in important cases also efficient, computation of provenance values is possible in absorptive, fully continuous semirings.

References

1 Y. Amsterdamer, D. Deutch, and V. Tannen. On the limitations of provenance for queries with difference. In *3rd Workshop on the Theory and Practice of Provenance, TaPP'11*, 2011. See also CoRR abs/1105.2255.

2 K. Apt and E. Grädel, editors. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, 2011. doi:10.1017/CBO9780511973468.

3 K. Dannert and E. Grädel. Provenance analysis: A perspective for description logics? In C. Lutz et al., editor, *Description Logic, Theory Combination, and All That*, Lecture Notes in Computer Science Nr. 11560. Springer, 2019. doi:10.1007/978-3-030-22102-7_12.

4 K. Dannert and E. Grädel. Semiring provenance for guarded logics. In *Hajnal Andréka and István Németi on Unity of Science: From Computing to Relativity Theory through Algebraic Logic*, Outstanding Contributions to Logic. Springer, 2021.

5 K. Dannert, E. Grädel, M. Naaf, and V. Tannen. Generalized absorptive polynomials and provenance semantics for fixed-point logic. arXiv: 1910.07910 [cs.LO], 2019. URL: `https://arxiv.org/abs/1910.07910`.

6 D. Deutch, T. Milo, S. Roy, and V. Tannen. Circuits for datalog provenance. In *Proc. 17th International Conference on Database Theory ICDT*, pages 201–212. OpenProceedings.org, 2014. `doi:10.5441/002/icdt.2014.22`.

7 M. Droste and P. Gastin. Weighted automata and weighted logics. In *Handbook of weighted automata*, pages 175–211. Springer, 2009. `doi:10.1007/978-3-642-01492-5_5`.

8 M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata.* Springer, 2009. `doi:10.1007/978-3-642-01492-5`.

9 M. Droste and G. Rahonis. Weighted automata and weighted logics on infinite words. In *International Conference on Developments in Language Theory*, pages 49–58. Springer, 2006. `doi:10.1007/11779148_6`.

10 F. Geerts and A. Poggi. On database query languages for K-relations. *J. Applied Logic*, 8(2):173–185, 2010. `doi:10.1016/j.jal.2009.09.001`.

11 F. Geerts, T. Unger, G. Karvounarakis, I. Fundulaki, and V. Christophides. Algebraic structures for capturing the provenance of SPARQL queries. *J. ACM*, 63(1):7:1–7:63, 2016. `doi:10.1145/2810037`.

12 E. Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007. `doi:10.1007/3-540-68804-8`.

13 E. Grädel and V. Tannen. Semiring provenance for first-order model checking. arXiv:1712.01980 [cs.LO], 2017. URL: `https://arxiv.org/abs/1712.01980`.

14 E. Grädel and V. Tannen. Provenance analysis for logic and games. *Moscow Journal of Combinatorics and Number Theory*, 9(3):203–228, 2020. `doi:10.2140/moscow.2020.9.203`.

15 T. Green, Z. Ives, and V. Tannen. Reconcilable differences. In *Database Theory - ICDT 2009*, pages 212–224, 2009. `doi:10.1145/1514894.1514920`.

16 T. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Principles of Database Systems PODS*, pages 31–40, 2007. `doi:10.1145/1265530.1265535`.

17 T. Green and V. Tannen. The semiring framework for database provenance. In *Proceedings of PODS*, pages 93–99. ACM, 2017. `doi:10.1145/3034786.3056125`.

18 A. Lluch-Lafuente and U. Montanari. Quantitative mu-calculus and CTL defined over constraint semirings. *Theoretical Compututer Science*, 346(1):135–160, 2005. `doi:10.1016/j.tcs.2005.08.006`.

19 G. Markowsky. Chain-complete posets and directed sets with applications. *Algebra universalis*, 6(1):53–68, 1976.

20 Y. Moschovakis. *Elementary induction on abstract structures.* North Holland, 1974.

21 Y. Ramusat, S. Maniu, and P. Senellart. Semiring provenance over graph databases. In *10th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2018)*, London, 2018.

22 P. Senellart. Provenance and probabilities in relational databases: From theory to practice. *SIGMOD Record*, 46(4):5–15, 2017. `doi:10.1145/3186549.3186551`.

23 J. Xu, W. Zhang, A. Alawini, and V. Tannen. Provenance analysis for missing answers and integrity repairs. *IEEE Data Eng. Bull.*, 41(1):39–50, 2018.

A Proofs

This appendix contains proofs of two key results, the Fundamental Property (Proposition 7) and the Universality (Theorem 17) of the semiring $\mathbb{S}^{\infty}[X]$. Proofs of all remaining results, in particular for Sect. 6, are available in the full version of this paper [5].

A.1 Fundamental Property

▶ **Proposition 7** (Fundamental Property). *Let K_1, K_2 be fully chain-complete semirings and let $h : K_1 \to K_2$ be a fully continuous semiring homomorphism with $h(\top) = \top$. Then for every K_1-interpretation π, the mapping $h \circ \pi$ is a K_2-interpretation and for every $\varphi \in \mathrm{LFP}$, we have $h(\pi[\![\varphi]\!]) = (h \circ \pi)[\![\varphi]\!]$. In diagrammatic form:*

$$\begin{tikzcd} & \mathrm{Lit}_A(\tau) \arrow[dl, "\pi"'] \arrow[dr, "h \circ \pi"] & \\ K_1 \arrow[rr, "h"] & & K_2 \end{tikzcd} \quad \Longrightarrow \quad \begin{tikzcd} & \mathrm{LFP} \arrow[dl, "\pi"'] \arrow[dr, "h \circ \pi"] & \\ K_1 \arrow[rr, "h"] & & K_2 \end{tikzcd}$$

Proof. The proof is a mostly straightforward induction on the structure of φ. For fixed-point formulae, we proceed by transfinite induction on the fixed-point iterations and rely on full continuity of the homomorphism h for limit ordinals. Formally, we prove that for all LFP-formulae $\varphi(\mathbf{x})$ in negation normal form, $h(\pi[\![\varphi(\mathbf{a})]\!]) = (h \circ \varphi)[\![\varphi(\mathbf{a})]\!]$ holds for all K-interpretations π and all tuples $\mathbf{a}$ from the universe A.

- For literals, we trivially have $h(\pi[\![R\mathbf{a}]\!]) = h(\pi(R\mathbf{a})) = (h \circ \pi)(R\mathbf{a}) = (h \circ \pi)[\![R\mathbf{a}]\!]$.
- For $\varphi = \varphi_1 \wedge \varphi_2$ we use that h is a semiring homomorphism:

  $$h(\pi[\![\varphi]\!]) = h(\pi[\![\varphi_1]\!] \cdot \pi[\![\varphi_2]\!]) = h(\pi[\![\varphi_1]\!]) \cdot h(\pi[\![\varphi_2]\!]) = (h \circ \pi)[\![\varphi_1]\!] \cdot (h \circ \pi)[\![\varphi_2]\!] = (h \circ \pi)[\![\varphi]\!].$$

  The proof for $\vee$, $\exists$ and $\forall$ is analogous (recall that we assume a finite universe, so quantifiers translate to finite sums or products).
- For $\varphi = [\mathbf{gfp}\, R\mathbf{x} \,.\, \vartheta](\mathbf{y})$ with R of arity k, we consider the fixed-point iteration $(g_\beta)_{\beta \in \mathrm{On}}$ for π in K_1 and the iteration $(f_\beta)_{\beta \in \mathrm{On}}$ for $h \circ \pi$ in K_2. We show by induction that $h \circ g_\beta = f_\beta$ for all ordinals $\beta \in \mathrm{On}$, so h preserves all steps of the fixed-point iteration.
  - Initially, $g_0 \colon A^k \to K_1$, $\mathbf{a} \mapsto \top$ and $f_0 \colon A^k \to K_2$, $\mathbf{a} \mapsto \top$. So $h \circ g_0 = f_0$ by $h(\top) = \top$.
  - For successor ordinals, we can apply the induction hypothesis. By definition,

    $$\begin{aligned} g_{\beta+1}(\mathbf{a}) &= F^{\vartheta}_{\pi}(g_\beta)(\mathbf{a}) = \pi[R \mapsto g_\beta][\![\vartheta(\mathbf{a})]\!], \\ f_{\beta+1}(\mathbf{a}) &= F^{\vartheta}_{h\circ\pi}(f_\beta)(\mathbf{a}) = (h \circ \pi)[R \mapsto f_\beta][\![\vartheta(\mathbf{a})]\!] \overset{(*)}{=} (h \circ \pi[R \mapsto g_\beta])[\![\vartheta(\mathbf{a})]\!]. \end{aligned}$$

    In $(*)$, we use the induction hypothesis $h \circ g_\beta = f_\beta$. Using the (outer) induction hypothesis on ϑ in $(\dagger)$, we obtain

    $$(h \circ g_{\beta+1})(\mathbf{a}) = h(\pi[R \mapsto g_\beta][\![\vartheta(\mathbf{a})]\!]) \overset{(\dagger)}{=} (h \circ \pi[R \mapsto g_\beta])[\![\vartheta(\mathbf{a})]\!] = f_{\beta+1}(\mathbf{a}).$$
  - For limit ordinals, we exploit that h is fully continuous:

    $$\begin{aligned} h(g_\lambda(\mathbf{a})) &= h(\bigsqcap\{g_\beta(\mathbf{a}) \mid \beta < \lambda\}) \\ &= \bigsqcap\{h(g_\beta(\mathbf{a})) \mid \beta < \lambda\} = \bigsqcap\{f_\beta(\mathbf{a}) \mid \beta < \lambda\} = f_\lambda(\mathbf{a}). \end{aligned}$$

  This closes the proof for **gfp**-formulae, as for sufficiently large β, we have

  $$h(\pi[\![\varphi(\mathbf{a})]\!]) = h(g_\beta(\mathbf{a})) = f_\beta(\mathbf{a}) = (h \circ \pi)[\![\varphi(\mathbf{a})]\!].$$

  The proof for **lfp**-formulae is analogous. ◀

A.2 Universal Property of Absorptive Polynomials

▶ **Theorem 17** (Universality). *Every mapping $h : X \to K$ into an absorptive, fully continuous semiring K uniquely extends to a fully continuous semiring homomorphism $h : \mathbb{S}^\infty[X] \to K$.*

Towards the proof, we need an auxiliary lemma on descending ω-chains, that is, sequences of the form $(a_i)_{i<\omega}$ with $a_0 \geq a_1 \geq \dots$ of elements from an absorptive, fully continuous semiring. For instance, the powers of an element a form a descending ω-chain, since $a \geq a^2 \geq a^3 \geq \dots$. We refer to the infimum of this chain as infinitary power $a^\infty = \bigsqcap_n a^n$.

▶ **Lemma A1** (Splitting Lemma). *Let K be a fully continuous semiring and let $(a_i)_{i<\omega}$ and $(b_i)_{i<\omega}$ be two descending ω-chains. Then, $\bigsqcap_{i<\omega}(a_i \circ b_i) = \big(\bigsqcap_{i<\omega} a_i\big) \circ \big(\bigsqcap_{j<\omega} b_j\big)$, with $\circ \in \{+, \cdot\}$. Analogous statements hold for suprema.*

Proof. We only show the statement for infima, the proof for suprema is analogous. We have the following equality, where $(*)$ holds since K is fully continuous:

$$\bigsqcap_{i<\omega} a_i \circ b_i \overset{(1)}{=} \bigsqcap_{i<\omega} \bigsqcap_{j<\omega} a_i \circ b_j \overset{(*)}{=} \bigsqcap_{i<\omega} (a_i \circ \bigsqcap_{j<\omega} b_j) \overset{(*)}{=} \bigsqcap_{i<\omega} a_i \;\circ\; \bigsqcap_{j<\omega} b_j$$

We prove both directions of (1). Fix i, j and let $k = \max(i, j)$. Then $a_i \circ b_j \geq a_k \circ b_k \geq \bigsqcap_k a_k \circ b_k$ by monotonicity of $\circ$. As i, j are arbitrary, this proves $\bigsqcap_i \bigsqcap_j a_i \circ b_j \geq \bigsqcap_k a_k \circ b_k$.

For the other direction, we have $a_i \circ b_i \geq a_i \circ \bigsqcap_j b_j$ for every i by monotonicity of $\circ$. By continuity, $a_i \circ b_i \geq \bigsqcap_j a_i \circ b_j$ for every i, and thus $\bigsqcap_i a_i \circ b_i \geq \bigsqcap_i \bigsqcap_j a_i \circ b_j$. ◀

Recall that full continuity of the induced homomorphism h means that it preserves suprema and infima of all chains, i.e., of all totally ordered sets. By observing that $\mathbb{S}^\infty[X]$ is countable (given that X is finite), it in fact suffices to show that suprema and infima of ω-chains are preserved:

▶ **Lemma A2** (Countable Chains). *Let K, K' be fully chain-complete semirings and $C \subseteq K$ a countable chain. Then there is a descending ω-chain $(x_i)_{i<\omega}$ such that $\bigsqcap C = \bigsqcap_i x_i$. Moreover, if $f : K \to K'$ is a monotone function, then additionally $\bigsqcap f(C) = \bigsqcap_i f(x_i)$. Analogous statements hold for suprema.*

Proof. We only show the statement involving f, as it implies the first, and only consider infinite C (otherwise the statement is trivial). Fix a bijection $g : \omega \to C$ and recursively define $x_0 = g(0)$ and $x_{i+1} = \min(g(i+1), x_i)$. This defines an ω-chain with $x_i \in C$ and thus $\bigsqcap_i f(x_i) \geq \bigsqcap f(C)$. Conversely, for every $c \in C$ there is an i with $g(i) = c$ and thus $c \geq x_i$. By monotonicity, $f(c) \geq f(x_i)$ and thus $\bigsqcap f(C) \geq \bigsqcap_i f(x_i)$. ◀

We are now ready to prove the universal property. The main difficulty is to show that h preserves infima of chains; we achieve this by simplifying the chain to a well-behaved *canonical chain* with similar convergence properties, as illustrated in Figure 1.

Proof of Theorem 17. Due to the additivity and multiplicity requirements for homomorphisms, h uniquely extends to monomials. For the exponent ∞, notice that continuity requires $h(x^\infty) = \bigsqcap_{n<\omega} h(x)^n$ for $x \in X$. It further follows that $h(m_1 + m_2) = h(m_1) + h(m_2)$, hence h is uniquely defined on $\mathbb{S}^\infty[X]$. Care has to be taken regarding absorption: If $m_1 \preceq m_2$, then $m_1 + m_2 = m_2$. Since h is order-preserving and K is absorptive, we also have $h(m_1 + m_2) = h(m_1) + h(m_2) = h(m_2)$ and it follows that h is well-defined.

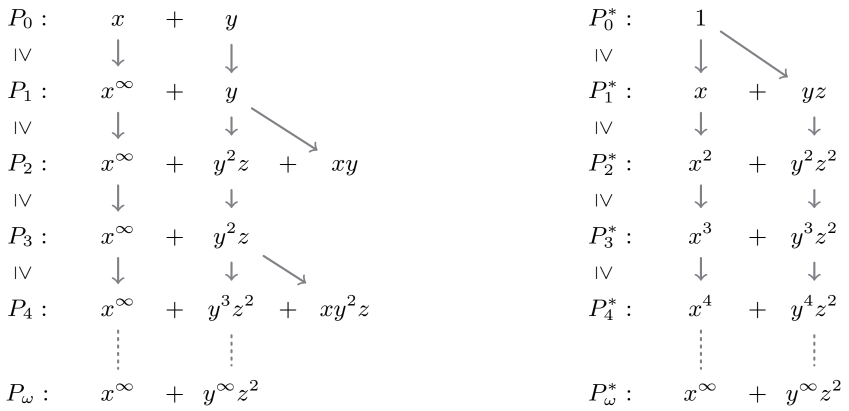

P_0 : $x + y$
P_1 : $x^\infty + y$
P_2 : $x^\infty + y^2z + xy$
P_3 : $x^\infty + y^2z$
P_4 : $x^\infty + y^3z^2 + xy^2z$
P_ω : $x^\infty + y^\infty z^2$
IV
P_0^* : 1
P_1^* : $x + yz$
P_2^* : $x^2 + y^2z^2$
P_3^* : $x^3 + y^3z^2$
P_4^* : $x^4 + y^4z^2$
P_ω^* : $x^\infty + y^\infty z^2$

Figure 1 An example of an ω-chain of polynomials (left) and the corresponding canonical chain (right) for the proof of Theorem 17. The arrows indicate absorption between monomials of consecutive polynomials and induce a directed graph on which we then apply Kőnig's lemma.

It remains to show that h is fully continuous. Ascending chains are always finite (because of $\bigsqcup S = \text{maximals}\,(\bigcup S)$), so we only have to consider descending chains. By Lemma A2, it further suffices to consider ω-chains. Hence it suffices to prove that

$$\bigsqcap_{i<\omega} h(P_i) = h\big(\bigsqcap_{i<\omega} P_i\big)$$

for any descending ω-chain $(P_i)_{i<\omega}$ of polynomials in $\mathbb{S}^\infty[X]$. The homomorphism h preserves addition and is thus monotone, which entails the direction "$\geq$".

For the other direction, we first consider the case of single monomials. Let $(m_i)_{i<\omega}$ be a descending ω-chain of monomials. Recall that X is finite, so we can write $m_i = \prod_{x\in X} x^{m_i(x)}$. As the m_i form a descending chain, the exponents $(m_i(x))_{i<\omega}$ form an ascending chain for each $x \in X$. By Lemma A1 and the definition of h,

$$\bigsqcap_{i<\omega} h(m_i) = \prod_{x\in X} \bigsqcap_{i<\omega} h(x)^{m_i(x)} \overset{(*)}{=} \prod_{x\in X} h(x)^{(\bigsqcup_{i<\omega} m_i(x))} = h\big(\bigsqcap_{i<\omega} m_i\big).$$

where $(*)$ can easily be seen by case distinction whether $\bigsqcup_{i<\omega} m_i(x)$ is finite or ∞.

For the general case of polynomials, let $P_\omega = \bigsqcap_{i<\omega} P_i$ be the infimum, which is of the form $P_\omega = m_1 + \cdots + m_n$. We define a second, canonical ω-chain $(P_i^*)_{i<\omega}$ with the same infimum. To this end, we define the canonical monomial chain $(m_j^*)_{j<\omega}$ of a given monomial m as follows (see Figure 1 for an example),

$$m_j^*(x) = \min(j, m(x)), \qquad \text{for all } x \in X,$$

which satisfies the following properties needed for the proof:

1. If m, v are two monomials with $m \preceq v$, then $m_j^* \preceq v_j^*$ for all $j < \omega$.
2. If $m = \bigsqcap_{i<\omega} m_i$ for an ω-chain $(m_i)_{i<\omega}$ of monomials, then $\forall j\, \exists i : m_j^* \succeq m_i$.
3. In particular, $\bigsqcap_{j<\omega} m_j^* = m$.

The canonical polynomial chain $(P_j^*)_{j<\omega}$ is then defined by $P_j^* = (m_1)_j^* + \cdots + (m_n)_j^*$ for each $j < \omega$. We make the following observation:

▷ Claim.

$\forall j\, \exists i\colon P_j^* \geq P_i$.

We first show that the claim implies the theorem:

$$\begin{aligned}\prod_{i<\omega} h(P_i) \overset{(1)}{\leq} \prod_{j<\omega} h(P_j^*) = \prod_{j<\omega} \Big(h((m_1)_j^*) + \dots + h((m_n)_j^*) \Big) \\ \overset{(2)}{=} \prod_{j<\omega} h((m_1)_j^*) + \dots + \prod_{j<\omega} h((m_n)_j^*) \\ \overset{(3)}{=} h\Big(\prod_{j<\omega} (m_1)_j^* \Big) + \dots + h\Big(\prod_{j<\omega} (m_n)_j^* \Big) \\ \overset{(4)}{=} h(m_1) + \dots + h(m_n) = h(P_\omega),\end{aligned}$$

where (1) follows from the claim, (2) holds by Lemma A1, (3) was shown above and (4) holds due to property 3 above. Hence the claim suffices to prove the theorem.

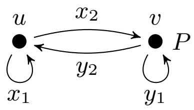
To prove the claim, assume towards a contradiction that there is a j such that $P_j^* \not\geq P_i$ for all $i < \omega$. Let us fix an $i < \omega$ for the moment. Because of $P_j^* \not\geq P_i$, there is a monomial $m_i \in P_i$ with $P_j^* \not\geq m_i$. Because of $P_{i-1} \geq P_i$, there is further $m_{i-1} \in P_{i-1}$ with $m_{i-1} \succeq m_i$. But then also $P_j^* \not\geq m_{i-1}$ (as otherwise $P_j^* \geq m_{i-1} \geq m_i$). By repeating this argument, we obtain a finite chain $m_0 \succeq m_1 \succeq \dots \succeq m_i$ of monomials with the property that $m_k \in P_k$ and $P_j^* \not\geq m_k$ for all $0 \leq k \leq i$.

This argument applies to all $i < \omega$, so we obtain arbitrarily long finite chains with this property. By Kőnig's lemma (recall that all polynomials P_i are finite), there must be an infinite monomial chain $(m_i)_{i<\omega}$ with $m_i \in P_i$ and $P_j^* \not\geq m_i$ for all $i < \omega$. Let $m_\omega = \prod_{i<\omega} m_i$. Because of $m_i \leq P_i$ for all i, we have $m_\omega \leq P_\omega$, so there is a monomial $v \in P_\omega$ with $m_\omega \preceq v$. By considering the corresponding canonical monomial chains $(v_k^*)_{k<\omega}$ and $((m_\omega)_k^*)_{k<\omega}$ at $k = j$, we obtain a contradiction: We know from the above properties that there is an i with $(m_\omega)_j^* \succeq m_i$ and further $v_j^* \geq (m_\omega)_j^*$. Because of $v_j^* \in P_j^*$, we obtain $P_j^* \geq v_j^* \geq (m_\omega)_j^* \geq m_i$, contradicting our assumption. The claim follows, closing the overall proof. ◀

B Example of a Model-Checking Game

The proof of the main result in Sect. 6 requires some preparations and is deferred to the full version [5]. Here, we attempt to convince the reader by an example instead of rigorous arguments. Due to space reasons, we consider a small graph comprised of only two nodes. The formula $\varphi(u)$, on the other hand, features alternating least and greatest fixed points and is thus non-trivial to analyse. It expresses that there is a path from u on which P holds infinitely often. We evaluate $\varphi(u)$ using the model-compatible $\mathbb{S}^\infty[X, \bar{X}]$-interpretation π over $A = \{u, v\}$ indicated on the right, with $\pi(Pu) = 0$ and $\pi(Pv) = 1$.

$$\varphi(u) = [\mathbf{gfp}\, Xx\,.\,[\mathbf{lfp}\, Yx\,.\,\exists y\big(Exy \wedge ((Xy \wedge Py) \vee Yy)\big)](x)\,](u)$$



Intuitively, witnesses for $\varphi(u)$ are infinite paths that infinitely often visit v. There are infinitely many such paths, but the simplest ones (in terms of the different edges they use) are the paths $uvvvv\dots$ and $uvuvuv\dots$ which correspond to the monomials $x_2 y_1^\infty$ and $x_2^\infty y_2^\infty$. And indeed, $\pi[\![\varphi(u)]\!] = x_2 y_1^\infty + x_2^\infty y_2^\infty$. Notice that the edge x_1 does not appear in the result and we can conclude that its existence does not affect the truth of $\varphi(u)$.

Let us now consider the evaluation strategies for $\varphi(u)$ from the game-theoretic perspective. The complete model-checking game is shown in Figure 2, where rounded nodes belong to Verifier, rectangular nodes to Falsifier, and the small numbers indicate the priorities

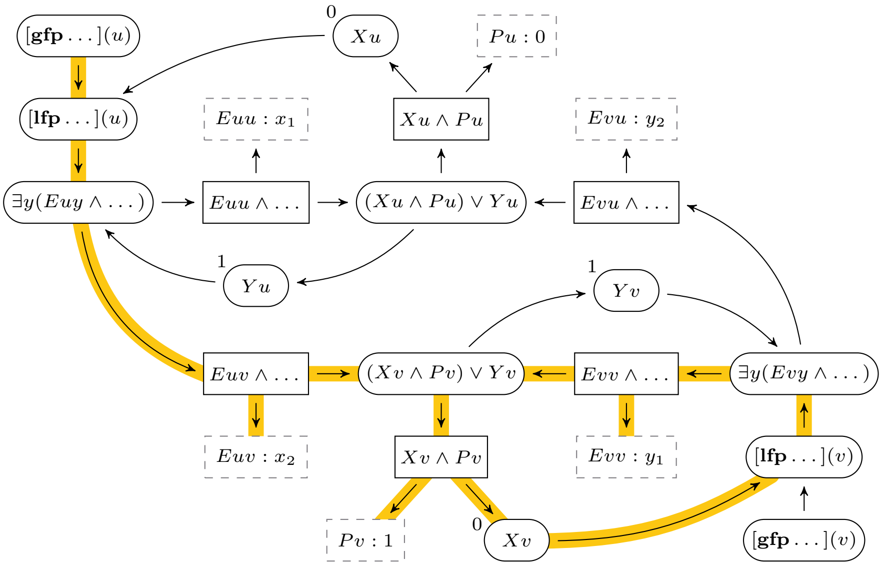


Figure 2 Model-checking game for $\varphi(u)$, with highlighted winning strategy.

assigned to fixed-point relations. Terminal positions have dashed borders and include the value assigned by π. Verifier can make decisions at four positions (two nodes labelled $\exists y(\dots)$ and two disjunctions in the center), hence there are 16 positional strategies in total. One of these strategies is highlighted (yellow color) and has the provenance value $x_2 y_1^\infty$, having one play ending in Euv and arbitrarily long plays ending either in Evv or in Pv (depending on the choices of Falsifier). Most of the other strategies allow infinite plays with least priority 1 which lead to provenance value 0 (for instance by choosing the cycle $\exists y(\dots) \to Euu \wedge \dots \to (Xu \wedge Pu) \vee Yu \to Yu$). The only remaining strategy has the provenance value $x_2^\infty y_2^\infty$. One can further observe that non-positional strategies only lead to monomials with additional variables which are then absorbed, so summing over all strategies gives $x_2 y_1^\infty + x_2^\infty y_2^\infty$ as expected.

Extension Preservation in the Finite and Prefix Classes of First Order Logic

Anuj Dawar
Department of Computer Science and Technology, University of Cambridge, UK
anuj.dawar@cl.cam.ac.uk

Abhisekh Sankaran
Department of Computer Science and Technology, University of Cambridge, UK
abhisekh.sankaran@cl.cam.ac.uk

Abstract

It is well known that the classic Łoś-Tarski preservation theorem fails in the finite: there are first-order definable classes of finite structures closed under extensions which are not definable (in the finite) in the existential fragment of first-order logic. We strengthen this by constructing for every n, first-order definable classes of finite structures closed under extensions which are not definable with n quantifier alternations. The classes we construct are definable in the extension of Datalog with negation and indeed in the existential fragment of transitive-closure logic. This answers negatively an open question posed by Rosen and Weinstein.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases finite model theory, preservation theorems, extension closed, composition, Datalog, Ehrenfeucht-Fraisse games

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.18

Funding Research supported by the Leverhulme Trust through a Research Project Grant on "Logical Fractals"

1 Introduction

The failure of classical preservation theorems of model theory has been a topic of persistent interest in finite model theory. In the classical setting, preservation theorems provide a tight link between the syntax and semantics of first-order logic (FO). For instance, the Łoś-Tarski preservation theorem (see [8]) implies that any sentence of first-order logic whose models are closed under extensions is equivalent to an existential sentence. This, like many other classical preservation theorems, is false when we retrict ourselves to finite structures. Tait [17] and Gurevich [7] provide examples of sentences whose finite models are closed under extensions, but which are not equivalent, over finite structures, to any existential sentence. Many other classical preservation theorems have been studied in the context of finite model theory (e.g. [12, 14]), but our focus in this paper is on extension-closed properties.

The failure of the Łoś-Tarski theorem in the finite opens a number of different avenues of research. One line of work has sought to investigate restricted classes of structures on which a version of the preservation theorem holds (see [2, 4]). Another direction is prompted by the question of whether we can identify some proper syntactic fragment of FO, beyond the existential, which contains definitions of all extension-closed FO-definable properties. For instance, the examples from Tait and Gurevich are both Σ_3 sentences. Could it be that every FO sentence whose finite models are closed under extensions is equivalent to a Σ_3 sentence? Or, indeed, a Σ_n sentence for some constant n? We answer these questions negatively in this paper. That is, we show that we can construct, for each n, a sentence φ whose finite models are extension closed but which is not equivalent in the finite to a Σ_n sentence.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 18; pp. 18:1–18:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A related question is posed by Rosen and Weinstein [13]. They observe that the constructions due to Tait and Gurevich both yield classes of finite structures that are definable in $\texttt{Datalog}(\neg)$, the existential fragment of fixed-point logic in which only extension-closed properties can be expressed. They ask if it might be the case that FO $\cap$ $\texttt{Datalog}(\neg)$ is contained in some level of the first-order quantifier alternation hierarchy, be it not the lowest level. That is, could it be that every property that is first-order definable and also definable in $\texttt{Datalog}(\neg)$ is definable by a Σ_n sentence for some constant n? Our construction answers this question negatively as we show that the sentences we construct are all equivalent to formulas of $\texttt{Datalog}(\neg)$.

Our result also greatly strengthens a previous result by Sankaran [15] which showed that for each k there is an extension-closed property of finite structures definable in FO but not in Π_2 with k leading universal quantifiers. Indeed, our result answers (negatively) Problem 2 in [15].

In Section 2 we give the necessary background definitions. We construct the sentences in Section 3 and show that they can all be expressed in $\texttt{Datalog}(\neg)$. Section 4 contains the proof that the sequence of sentences contains, for each n, a sentence that is not equivalent to any Σ_n sentence. We conclude with some suggestions for further investigation.

2 Preliminaries

We work with logics: first-order logic (FO) and extensions of Datalog over finite relational vocabularies. We assume the reader is familiar with the basic definitions of first-order logic (see, for instance [10]). A *vocabulary* τ is a set of predicate and constant symbols. In the vocabularies we use, all predicate symbols are either unary or binary. We denote by FO(τ) the set of all FO formulas over the vocabulary τ. A sequence $(x_1, \ldots, x_k)$ of variables is denoted by $\bar{x}$. We use $\psi(\bar{x})$ to denote a formula ψ whose free variables are among $\bar{x}$. A formula without free variables is called a *sentence*. A formula which begins with a string of quantifiers that is followed by a quantifier-free formula, is said to be in *prenex normal form (PNF)*. The string of quantifiers in a PNF formula is called the *quantifier prefix* of the formula. It is well known that every formula is equivalent to a formula in PNF. We denote by Σ_n, the collection of all formulas in PNF whose quantifier prefix contains at most n blocks of quantifiers beginning with a block of existential quantifiers. Equivalently, a PNF formula is in Σ_n if it starts with a block of existential quantifiers and contains at most $n-1$ alternations in its quantifier prefix. Similarly, a formula is Π_n if it begins with a block of universal quantifiers and contains at most $n-1$ alternations in its quantifier prefix. We write $\Sigma_{n,k}$ for the subclass of Σ_n consisting of those formulas in which every quantifier block has at most k quantifiers. Similarly, $\Pi_{n,k}$ is the subclass of Π_n where each block has at most k quantifiers. Thus $\Sigma_n = \bigcup_{k\geq 1} \Sigma_{n,k}$ and $\Pi_n = \bigcup_{k\geq 1} \Pi_{n,k}$.

We use standard notions concerning τ-structures as defined in [3]. We denote τ-structures as $\mathfrak{A}, \mathfrak{B}$ etc., and refer to them simply as structures when τ is clear from the context. We denote by $\mathfrak{A} \subseteq \mathfrak{B}$ that $\mathfrak{A}$ is a substructure of $\mathfrak{B}$, and by $\mathfrak{A} \cong \mathfrak{B}$ that $\mathfrak{A}$ is isomorphic to $\mathfrak{B}$.

We now introduce some notation with respect to the classes of formulas $\Sigma_{n,k}$ and $\Pi_{n,k}$.

▶ **Definition 1.** *We say $\mathfrak{A} \Rrightarrow_{n,k} \mathfrak{B}$ if every $\Sigma_{n,k}$ sentence true in $\mathfrak{A}$ is also true in $\mathfrak{B}$.*
We say $\mathfrak{A}$ and $\mathfrak{B}$ are $\equiv_{n,k}$-equivalent, *and write $\mathfrak{A} \equiv_{n,k} \mathfrak{B}$, if $\mathfrak{A} \Rrightarrow_{n,k} \mathfrak{B}$ and $\mathfrak{B} \Rrightarrow_{n,k} \mathfrak{A}$.*

By extension, for tuples $\bar{a}$ and $\bar{b}$ of elements of $\mathfrak{A}$ and $\mathfrak{B}$ respectively, we also write $(\mathfrak{A}, \bar{a}) \Rrightarrow_{n,k} (\mathfrak{B}, \bar{b})$ to indicate that every formula φ which is satisfied in $\mathfrak{A}$ when its free variables are instantiated with $\bar{a}$ is also satisfied in $\mathfrak{B}$ when they are instantiated by $\bar{b}$, and similarly for $\equiv_{n,k}$. Note that $\mathfrak{A} \Rrightarrow_{n,k} \mathfrak{B}$ holds if every $\Pi_{n,k}$ sentence true in $\mathfrak{B}$ is also true in $\mathfrak{A}$. The following useful fact is now immediate from the definition.

▶ **Lemma 2.** *$\mathfrak{A} \Rrightarrow_{n+1,k} \mathfrak{B}$ if, and only if, for every k-tuple $\bar{a}$ of elements of $\mathfrak{A}$, there is a k-tuple $\bar{b}$ of elements of $\mathfrak{B}$ such that $(\mathfrak{B}, \bar{b}) \Rrightarrow_{n,k} (\mathfrak{A}, \bar{a})$.*

We assume the reader is familiar with the standard Ehrenfeucht-Fraïssé game characterizing the equivalence of two structures with respect to sentences of a given quantifier nesting depth (see for example [10, Chapter 3]). In this paper, we use a "prefix" variant of this Ehrenfeucht-Fraïssé game. For $n, k \geq 1$, the (n, k)-*prefix Ehrenfeucht-Fraïssé game* on a pair $(\mathfrak{A}, \mathfrak{B})$ of structures, is the usual Ehrenfeucht-Fraïssé game on $\mathfrak{A}$ and $\mathfrak{B}$ but with two restrictions: (i) In every odd round, the Spoiler plays on $\mathfrak{A}$ and in every even round, on $\mathfrak{B}$, and (ii) in each round the Spoiler chooses a k-tuple of elements from the relevant structure (as opposed to a single element in the usual Ehrenfeucht-Fraïssé game). The winning condition for the Duplicator is the same as that in the usual Ehrenfeucht-Fraïssé game: Duplicator wins at the end of n rounds, if when $\bar{a}_1, \dots, \bar{a}_n$ are the k-tuples picked in $\mathfrak{A}$ and $\bar{b}_1, \dots, \bar{b}_n$ are the k-tuples picked in $\mathfrak{B}$, then the map taking the nk-tuple $(\bar{a}_1 \cdots \bar{a}_n)$ to $(\bar{b}_1 \cdots \bar{b}_n)$ pointwise is a partial isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. Entirely analogously to the usual Ehrenfeucht-Fraïssé game theorem (see [10, Theorem 3.18]), we have the following.

▶ **Theorem 3.** *Duplicator has a winning strategy in the* (n, k)-prefix Ehrenfeucht-Fraïssé game *on a pair* $(\mathfrak{A}, \mathfrak{B})$ *of structures if, and only if,* $\mathfrak{A} \Rrightarrow_{n,k} \mathfrak{B}$

Note in particular that, given the fact that any two linear orders of length $\geq 2^n$ are equivalent with respect to all sentences of quantifier nesting depth n [10, Theorem 3.6], it follows that there is a winning strategy for the Duplicator in the (n, k)-prefix Ehrenfeucht-Fraïssé game on any pair of linear orders, each of length $\geq 2^{n \cdot k}$ and any two such linear orders are $\equiv_{n,k}$-equivalent.

Where it causes no confusion, we still use $\equiv_m$ to denote the usual equivalence up to *quantifier rank* m. Note that $\mathfrak{A} \equiv_m \mathfrak{B}$ implies $\mathfrak{A} \equiv_{n,k} \mathfrak{B}$ whenever $m \geq nk$.

Formulas in Σ_1 are said to be *existential.* A Σ_1 formula that also contains no occurrences of the negation symbol is said to be *existential positive.* It is easy to see that the class of models of any Σ_1 sentence φ is closed under extensions: if $\mathfrak{A} \models \varphi$ and $\mathfrak{A} \subseteq \mathfrak{B}$, then $\mathfrak{B} \models \varphi$. Dually, the class of models of any Π_1 sentence is closed under taking substructures. Similarly, the class of models of any existential positive sentence is closed under homomorphisms.

`Datalog` is a database query language which can be seen as an extension of existential positive first-order logic with a recursion mechanism. Equivalently, it can be seen as the existential positive fragment of the logic of least fixed points `LFP` (see [10, Chapter 10]). We briefly review the definitions of this language, along with its extension `Datalog`$(\neg)$.

A *Datalog program* is a finite set of rules of the form $T_0 \leftarrow T_1, \dots, T_m$, where each T_i is an atomic formula. T_0 is called the *head* of the rule, while the right-hand side is called the *body.* These atomic formulas use relational symbols from a vocabulary $\sigma \cup \tau$, where the symbols in σ are called *extensional* predicates and those in τ are *intensional* predicates. Every symbol that occurs in the head of a rule is an intensional predicate, while both intensional and extensional predicates can occur in the body of a rule. The semantics of such a program is defined with respect to a σ-structure $\mathfrak{A}$. Say that a rule $T_0 \leftarrow T_1, \dots, T_m$ is satisfied in a $\sigma \cup \tau$ expansion $\mathfrak{A}'$ of $\mathfrak{A}$ if $\mathfrak{A}' \models \forall \bar{x}\big((\bigwedge_{1 \leq i \leq m} T_i) \rightarrow T_0\big)$, where $\bar{x}$ enumerates all the variables occurring in the rule. The interpretation of a `Datalog` program in $\mathfrak{A}$ is the smallest expansion of $\mathfrak{A}$ (when ordered by pointwise inclusion of the relations interpreting τ) satisfying all the rules in the program. This is uniquely defined as it is obtained as the simultaneous least fixed-point of the existential closure of the right-hand side of the rules. We distinguish one intensional predicate G and call it the *goal predicate.* Then, the query computed by a program π is the interpretation of G in the interpretation of π in $\mathfrak{A}$. In particular, if G is a 0-ary predicate symbol (i.e. a Boolean variable), π defines a Boolean query, i.e. a class of structures.

Since the interpretation of π is obtained as the least fixed-point of an existential positive formula, it is easily seen that the query defined is closed under homomorphisms and hence also under extensions. We can understand `Datalog` as the existential positive fragment of the least-fixed point logic LFP, though it is known that there are homomorphism-closed properties definable in LFP that are not expressible in `Datalog` (see [5]).

We get more general queries by allowing limited forms of negation. Specifically, in `Datalog`$(\neg)$, in a rule $T_0 \leftarrow T_1, \dots, T_m$, each T_i on the right-hand side is either an atom *or* a negated atom involving an *extensional* predicate symbol or equality. In short, we allow negation on the predicate symbols in σ and on equalities but the fixed-point variables (i.e. the predicate symbols in τ) still only appear positively, so the least fixed-point is still well defined. As it is still the least-fixed point of existential formulas, the formula still defines a property closed under extensions. For more on the extensions of `Datalog` with negation, see [1, 9].

3 The Extension-Closed Properties

We now construct a family of properties, each of which is definable in first-order logic and closed under extensions. Indeed, we show that each of the properties is definable in `Datalog`$(\neg)$.

3.1 First-Order Definitions

To begin, we define, for each $n \in \mathbb{N}$, a vocabulary σ_n. These are defined by induction on n. The vocabulary σ_1 consists of three binary relation symbols $\leq, S, R$. For all $n > 1$, $\sigma_n = \sigma_{n-1} \cup \{S_n, R_n, P_n\}$ where S_n and R_n are binary relation symbols and P_n is a unary relation symbol.

Consider first the sentence NLO of FO which asserts that $\leq$ is *not* a linear order. This is easily seen to be an existential sentence and so also definable in `Datalog`$(\neg)$. Suppose now that φ is any sentence whose models, restricted to ordered structures (i.e. those structures which interpret $\leq$ as a linear order), are extension closed. Then, it follows that NLO $\vee\ \varphi$ defines an extension-closed class of structures. Moreover, this class is FO or `Datalog`$(\neg)$ definable if φ is in the respective logic. Also, if φ is a Σ_n sentence, then so is NLO $\vee\ \varphi$, and if $n > 1$ and φ is a Π_n sentence then so is NLO $\vee\ \varphi$. Thus, in what follows, we restrict our attention to the class of ordered structures. We construct our sentences on the assumption that structures are ordered, and show that they define extension-closed classes on ordered structures.

With this in mind, we use some convenient notational abbreviations. We write $x < y$ as short-hand for $x \leq y \wedge x \neq y$. We also write "y is the successor of x", "x is the minimum element", etc. with their obvious meanings. Also, let φ be any formula of FO, and x and y be variables not occuring in φ. We write $\varphi^{[x,y]}$ for the formula $x \leq y \wedge \varphi^\star$ where $\varphi^\star$ is the formula obtained by relativizing every quantifier in φ to the interval $[x, y]$. That is to say, inductively, every subformula $\exists z\theta$ is replaced by $\exists z(x \leq z \wedge z \leq y) \wedge \theta^\star$ and every subformula $\forall z\theta$ by $\forall z(x \leq z \wedge z \leq y) \rightarrow \theta^\star$. Where the variables x and y do appear in φ, the formula $\varphi^{[x,y]}$ is defined by first renaming variables in φ to avoid clashes and then applying the relativization.

Next, consider the sentence PartialSucc defined as follows.

$$\text{PartialSucc} \ := \ \forall x \forall y \ S(x, y) \rightarrow \text{“}y \text{ is the successor of } x\text{”}.$$

This is a Π_1 sentence, asserting that the relation S is a "partial successor" relation. Its negation is an existential sentence and hence closed under extensions. Thus, if φ defines an extension-closed class of structures when restricted to ordered structures in which S is a partial successor relation, then $\mathrm{NLO} \vee \neg\mathrm{PartialSucc} \vee \varphi$ defines an extension-closed class.

We write $\mathrm{Total}(x, y)$ for the formula that asserts, on structures in which PartialSucc is true, that in the interval $[x, y]$, S is, in fact, total. That is:

$$\mathrm{Total}(x, y) \ := \ x < y \wedge \forall z\big(x \leq z \wedge z < y\big) \rightarrow \exists w\big(z < w \wedge w \leq y \wedge S(z, w)\big).$$

Now, we can define the sentence $\mathrm{SomeTotalR}_1$.

$$\mathrm{SomeTotalR}_1 \ := \ \neg\mathrm{PartialSucc} \vee \exists x \exists y \left(R(x, y) \wedge \mathrm{Total}(x, y)\right).$$

Note that $\mathrm{Total}(x, y)$ is a Π_2 formula, and $\mathrm{SomeTotalR}_1$ is a Σ_3 sentence. The latter is the first in our family of sentences. To see why this sentence is closed under extensions on ordered structures, suppose $\mathfrak{A}$ is an ordered model of $\mathrm{SomeTotalR}_1$ on which S is a partial successor. Thus, there is an interval $[x, y]$ in $\mathfrak{A}$ on which S is total. Let $\mathfrak{B}$ be an extension of $\mathfrak{A}$. If $\mathfrak{B}$ contains no additional elements in the interval $[x, y]$, then $\mathrm{Total}(x, y)$ still holds in $\mathfrak{B}$ and therefore $\mathfrak{B}$ is a model of $\mathrm{SomeTotalR}_1$. On the other hand, suppose $\mathfrak{B}$ contains an additional element w in the interval $[x, y]$. Let a and b be the two successive elements of $\mathfrak{A}$ between which w appears. Since S is total in the interval $[x, y]$ in $\mathfrak{A}$, we know that $S(a, b)$ holds in $\mathfrak{A}$ and, by extension, in $\mathfrak{B}$. Since b is not the successor of a in $\mathfrak{B}$, we conclude that $\neg$PartialSucc is true in $\mathfrak{B}$ and therefore the structure is a model of $\mathrm{SomeTotalR}_1$.

The sentence $\mathrm{SomeTotalR}_1$ is essentially the example constructed by Tait that exhibits an existential-closed first-order property that is not expressible by an existential sentence. We now define σ_n-sentences $\mathrm{SomeTotalR}_n$, for $n > 0$ by induction.

First, we define a formula $\mathrm{Succ}_n(x, y)$ as follows.

$$\mathrm{Succ}_n(x, y) \ := \ P_n(x) \wedge P_n(y) \wedge S_n(x, y) \wedge \mathrm{SomeTotalR}_{n-1}^{[x,y]}.$$

We further define the formula $\mathrm{PartialSucc}_n$ which asserts that Succ_n is a partial successor relation when restricted to the elements in the relation P_n. That is,

$$\mathrm{PartialSucc}_n \ := \ \forall x \forall y \ \mathrm{Succ}_n(x, y) \rightarrow \forall z(P_n(z) \rightarrow z \leq x \vee y \leq z).$$

We can now define the formula $\mathrm{Total}_n(x, y)$ which defines, in those structures in which $\mathrm{PartialSucc}_n$ is true, those intervals $[x, y]$ where the successor defined by Succ_n is total. That is,

$$\mathrm{Total}_n(x, y) \ := \ x < y \wedge \forall z\big(P_n(z) \wedge x \leq z \wedge z < y\big) \rightarrow \exists w\big(z < w \wedge w \leq y \wedge \mathrm{Succ}_n(z, w)\big).$$

Finally, we define the sentence

$$\mathrm{SomeTotalR}_n \ := \ \neg\mathrm{PartialSucc}_n \vee \exists x \exists y \left(R_n(x, y) \wedge \mathrm{Total}_n(x, y)\right).$$

Note that, $\mathrm{SomeTotalR}_n$ is a Σ_{2n+1} sentence. This can be established by induction on n. Indeed, as we noted, $\mathrm{SomeTotalR}_1$ is a Σ_3 sentence. Assuming $\mathrm{SomeTotalR}_n$ is a Σ_{2n+1} sentence for some n, we note that Succ_{n+1} is a Σ_{2n+1} formula, and so is $\neg\mathrm{PartialSucc}_{n+1}$. Then Total_{n+1} is a Π_{2n+2} formula and $\mathrm{SomeTotalR}_{n+1}$ is Σ_{2n+3}.

3.2 Datalog Definitions

Next, we show that these formulas also admit a definition in `Datalog(¬)`, which establishes, in particular, that they define extension-closed classes. We use the same names for formulas in `Datalog(¬)` as we used for FO formulas above, when they define the same property. As we noted, the sentences NLO and ¬PartialSucc are both Σ_1 sentences and we therefore assume they are available as `Datalog(¬)` predicates. We now define Total by the following rules.

$$\begin{aligned}\mathrm{Total}(x,y) &\longleftarrow S(x,y)\\ \mathrm{Total}(x,y) &\longleftarrow S(x,z),\mathrm{Total}(z,y)\end{aligned}$$

This just defines Total as the transitive closure of S. It is clear that, in ordered structures where S is a partial successor relation, the pair (x,y) is in the transtive closure of S precisely when $x<y$ and S is total in the interval $[x,y)$. Thus, we can now define:

$$\mathrm{RTotal}_1(x,y) \longleftarrow x\le u, v\le y, R(u,v),\mathrm{Total}(u,v)$$

This defines those pairs (x,y) such that for some u,v in the interval $[x,y]$, $R(u,v)$ holds and the successor relation is total. In other words, it defines $\mathrm{SomeTotalR}_1^{[x,y]}$. We can obtain $\mathrm{SomeTotalR}_1$ as the existential closure of this. For the inductive definition, the predicate RTotal_n is useful.

Inductively, we define the relation, Succ_n as follows.

$$\mathrm{Succ}_n(x,y) \longleftarrow P_n(x),P_n(y),S_n(x,y),\mathrm{RTotal}_{n-1}(x,y)$$

The negation of $\mathrm{PartialSucc}_n$ is now defined by the following

$$\mathrm{NotPartialSucc}_n \longleftarrow \mathrm{Succ}_n(x,y),P_n(z),x\le z,z\le y,x\neq z,y\neq z$$

Now, entirely analogously to Total above, we can give a definition of Total_n as the transitive closure of Succ_n and this is equivalent to the FO definition given above on ordered structures on which $\mathrm{PartialSucc}_n$ is true.

$$\begin{aligned}\mathrm{Total}_n(x,y) &\longleftarrow \mathrm{Succ}_n(x,y)\\ \mathrm{Total}_n(x,y) &\longleftarrow \mathrm{Succ}_n(x,z),\mathrm{Total}_n(z,y)\end{aligned}$$

Inductively we define the relation RTotal_n, and its existential closure, giving the sentence $\mathrm{SomeTotalR}_n$.

$$\mathrm{RTotal}_n(x,y) \longleftarrow x\le u, v\le y, R_n(u,v),\mathrm{Total}_n(u,v)$$

It should be noted that the only use of the recursive features of `Datalog(¬)` that we made use in writing the formulas above was to define the transitive closure of the relations Total and Total_n. Thus, the definitions could equally well be formalized in the existential fragment of transitive closure logic.

4 Proof of the Main Result

In this section, we establish our main result. We establish that $\mathrm{SomeTotalR}_n$, which we noted is a Σ_{2n+1} sentence, is not equivalent to a Π_{2n+1} sentence. To do this, we construct ordered structures $\mathfrak{M}_{n,k}$ and $\mathfrak{N}_{n,k}$ for every k such that $\mathfrak{M}_{n,k}$ is a model of $\mathrm{SomeTotalR}_n$, $\mathfrak{N}_{n,k}$ is not a model of $\mathrm{SomeTotalR}_n$ but $\mathfrak{N}_{n,k} \Rrightarrow_{2n+1,k} \mathfrak{M}_{n,k}$. The main lemma establishing this is Lemma 5 below. Here we state the theorem that is a consequence.

▶ **Theorem 4.** *For every n, there is a Σ_{2n+1} sentence whose finite models are closed under extensions and which is equivalent to a* `Datalog`$(\neg)$ *program, but which is not equivalent over finite structures to any Π_{2n+1} sentence.*

Proof. The sentence is $\mathrm{NLO} \vee \mathrm{SomeTotalR}_n$ which we have already noted is a Σ_{2n+1} sentence, expressible as a `Datalog`$(\neg)$ program and its models are extension-closed. Suppose it were expressible as a Π_{2n+1} sentence. Then, since it is satisfied in $\mathfrak{M}_{n,k}$ as we show in Section 4.1 and since $\mathfrak{N}_{n,k} \Rrightarrow_{2n+1,k} \mathfrak{M}_{n,k}$ by Lemma 5 we have that the sentence is true in $\mathfrak{N}_{n,k}$. But, as we show in Section 4.1, $\mathfrak{N}_{n,k}$ is not a model of $\mathrm{SomeTotalR}_n$, yielding a contradiction. ◀

4.1 Construction of the Structures

We describe the construction of structures $\mathfrak{M}_{n,k}$ and $\mathfrak{N}_{n,k}$ for each n and k. The construction is by induction on n, simultaneously for all k. In the course of the construction we also define, for all n and k structures $\mathsf{Tot}_{n,k}$ and $\mathsf{Gap}_{n,k}$ which we use as auxilliary structures. For all n and k, $\mathfrak{M}_{n,k}, \mathfrak{N}_{n,k}, \mathsf{Tot}_{n,k}$ and $\mathsf{Gap}_{n,k}$ are structures over the vocabulary σ_n.

All structures we consider interpret the relation symbol $\leq$ as a linear order of the universe and S as a partial successor relation. It is useful to formally define the notion of an ordered sum of structures. For a pair $\mathfrak{A}$ and $\mathfrak{B}$ of ordered structures the *ordered sum* $\mathfrak{A} \oplus \mathfrak{B}$ is a structure whose universe is the disjoint union of the universes of $\mathfrak{A}$ and $\mathfrak{B}$ *except* that the maximum element of $\mathfrak{A}$ is identified with the minimum element of $\mathfrak{B}$. The relation $\leq$ is interpreted in $\mathfrak{A} \oplus \mathfrak{B}$ by taking the union of its interpretations in $\mathfrak{A}$ and $\mathfrak{B}$ and letting $a \leq b$ for all a in $\mathfrak{A}$ and b in $\mathfrak{B}$. All other relation symbols are interpreted in $\mathfrak{A} \oplus \mathfrak{B}$ by the union of their interpretations in the two structures. The operation of ordered sum is clearly associative and we can thus write $\bigoplus_{i \in I} \mathfrak{A}_i$ for the ordered sum of a sequence of structures indexed by an ordered set I. Note that our use of the term "ordered sum" differs somewhat from its use, say by Ebbinghaus and Flum [6, Sec. 1.A.3]. The key difference is that in their definition we do not identify the maximum element of $\mathfrak{A}$ with the minimum element of $\mathfrak{B}$ but rather simply take the disjoint union of the two universes.

The structure $\mathsf{Tot}_{1,k}$ has $m = 6(k+2)^2$ elements which we identify with the initial segment of the positive integers $[1, \dots, m]$ with $\leq$ the natural linear order on these, S the successor relation and the relation R containing just the pair $(1, m)$. The structure $\mathsf{Gap}_{1,k}$ is obtained from $\mathsf{Tot}_{1,k}$ by removing from the relation S the *central* pair of elements, i.e. $(m/2, m/2+1)$.

We now obtain $\mathfrak{N}_{1,k}$ as the ordered sum of $4(k+3)^3 + 2k + 1$ copies of $\mathsf{Gap}_{1,k}$. That is $\mathfrak{N}_{1,k} = \bigoplus_{i \in [4(k+3)^3+2k+1]} \mathfrak{G}_i$ where each $\mathfrak{G}_i$ is isomorphic to $\mathsf{Gap}_{1,k}$. We also let $\mathfrak{M}_{1,k} = \bigoplus_{i \in [2(k+3)^3+k]} \mathfrak{G}_i \oplus \mathsf{Tot}_{1,k} \oplus \bigoplus_{i \in [2(k+3)^3+k]} \mathfrak{G}_i$. In short, $\mathfrak{M}_{1,k}$ is obtained from $\mathfrak{N}_{1,k}$ by replacing the central copy of $\mathsf{Gap}_{1,k}$ with a copy of $\mathsf{Tot}_{1,k}$.

Let now $n \geq 2$ and suppose we have defined the σ_{n-1}-structures $\mathfrak{N}_{n-1,k}, \mathfrak{M}_{n-1,k}, \mathsf{Tot}_{n-1,k}$ and $\mathsf{Gap}_{n-1,k}$. Write $\mathfrak{N}^+_{n-1,k}$ and $\mathfrak{M}^+_{n-1,k}$ for the σ_n-structures that are obtained from $\mathfrak{N}_{n-1,k}$ and $\mathfrak{M}_{n-1,k}$ respectively by interpreting P_n as the two element set $\{\min, \max\}$ containing the minimum and maximum elements of the structure and S_n as the relation containing the single pair $(\min, \max)$ (R_n is empty in both these structures). Now, $\mathsf{Tot}_{n,k}$ is the structure obtained from $\bigoplus_{i \in [4(k+3)^{2n}+2k+1]} \mathfrak{M}^+_{n-1,k}$ (i.e. the ordered sum of $4(k+3)^{2n} + 2k + 1$ copies of $\mathfrak{M}^+_{n-1,k}$) by adding to the relation R_n the pair relating the minimum and maximum elements of the linear order. Similarly $\mathsf{Gap}_{n,k}$ is obtained from $\bigoplus_{i \in [2(k+3)^{2n}+k]} \mathfrak{M}^+_{n-1,k} \oplus \mathfrak{N}^+_{n-1,k} \oplus \bigoplus_{i \in [2(k+3)^{2n}+k]} \mathfrak{M}^+_{n-1,k}$ by adding to the relation R_n the pair relating the minimum and maximum elements of the linear order. Equivalently, $\mathsf{Gap}_{n,k}$ is obtained from $\mathsf{Tot}_{n,k}$ by replacing the central copy of $\mathfrak{M}_{n-1,k}$ by a copy of $\mathfrak{N}_{n-1,k}$.

Finally, we can define $\mathfrak{N}_{n,k}$ as the ordered sum of $4(k+3)^{2n+1}+2k+1$ copies of $\mathsf{Gap}_{n,k}$ and $\mathfrak{M}_{n,k}$ as the structure obtained from $\mathfrak{N}_{n,k}$ by replacing the central copy of $\mathsf{Gap}_{n,k}$ by a copy of $\mathsf{Tot}_{n,k}$. This completes the definition of the structures.

We now argue that for all values of n and k, $\mathfrak{M}_{n,k}$ is a model of $\mathrm{SomeTotalR}_n$ and $\mathfrak{N}_{n,k}$ is not. This is an easy induction on n. For $n=1$, every interval $[x,y]$ of $\mathfrak{N}_{1,k}$ for which $R(x,y)$ holds induces a copy of $\mathsf{Gap}_{1,k}$. By construction S is not a complete successor relation in $\mathsf{Gap}_{1,k}$, and so $\mathfrak{N}_{1,k}$ does not satisfiy $\mathrm{SomeTotalR}_1$. On the other hand, $\mathfrak{M}_{1,k}$ contains an interval $[x,y]$ with $R(x,y)$ that induces a copy of $\mathsf{Tot}_{1,k}$ and so $\mathfrak{M}_{1,k} \models \mathrm{SomeTotalR}_1$.

Inductively, assume that $\mathfrak{M}_{n-1,k} \models \mathrm{SomeTotalR}_{n-1}$ and $\mathfrak{N}_{n-1,k} \not\models \mathrm{SomeTotalR}_{n-1}$. Now, in both $\mathsf{Tot}_{n,k}$ and $\mathsf{Gap}_{n,k}$, the relation S_n relates successive elements that are in P_n. If x,y is a pair of such successive elements then in $\mathsf{Tot}_{n,k}$ the interval $[x,y]$ always induces a structure whose σ_{n-1}-reduct is a copy of $\mathfrak{M}_{n-1,k}$ and therefore satisfies $\mathrm{SomeTotalR}_{n-1}$. Hence $\mathrm{Succ}_n(x,y)$ is satisfied in $\mathsf{Tot}_{n,k}$ for all such pairs. On the other hand, in $\mathsf{Gap}_{n,k}$ there is an interval $[x,y]$ with $S_n(x,y)$ which induces a structure whose σ_{n-1}-reduct is a copy of $\mathfrak{N}_{n-1,k}$ and therefore fails to satisfy $\mathrm{SomeTotalR}_{n-1}$. Hence $\mathrm{Total}_n(x_0,y_0)$ is true in $\mathsf{Tot}_{n,k}$ and false in $\mathsf{Gap}_{n,k}$ when x_0 and y_0 are interpreted as the minimum and maximum elements in the structure respectively. Since in $\mathfrak{N}_{n,k}$ all intervals $[x,y]$ for which $R_n(x,y)$ holds induce a copy of $\mathsf{Gap}_{n,k}$ and in $\mathfrak{M}_{n,k}$ there is such an interval which induces a copy of $\mathsf{Tot}_{n,k}$, we conclude that $\mathfrak{M}_{n,k} \models \mathrm{SomeTotalR}_n$ and $\mathfrak{N}_{n,k} \not\models \mathrm{SomeTotalR}_n$.

4.2 The Game Argument

Our aim in this section is to establish the following lemma using an Ehrenfeucht-Fraïssé game argument:

▶ **Lemma 5.** *For each n,k, $\mathfrak{N}_{n,k} \Rrightarrow_{2n+1,k} \mathfrak{M}_{n,k}$.*

Our development of the Duplicator winning strategy in the game follows the inductive construction of the structures themselves. For this, we first develop some tools for constructing strategies on ordered sums and expansions of structures from strategies on their component parts. First, we introduce some useful notation.

For any ordered structure $\mathfrak{A}$, write $\mathfrak{A}^*$ for the expansion of $\mathfrak{A}$ with constants min and max interpreted by the minimum and maximum elements of the structure. The main reason for introducing these is that we generally want to restrict attention to Duplicator strategies that respect the minimum and maximum elements and a notationally convenient way to do this is to have constants for these elements.

It is a standard fact that the equivalence relation $\equiv_m$ is a congruence with respect to various ways of combining structures. In particular, it is so with respect to the notion of ordered sum defined by Ebbinghaus and Flum [6, Prop. 2.3.10]. The same method of composition of strategies can be used to show the following.

▶ **Lemma 6.** *If $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{B}_1, \mathfrak{B}_2$ are ordered structures and $\bar{a}_1, \bar{a}_2, \bar{b}_1, \bar{b}_2$ tuples of elements from $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{B}_1$ and $\mathfrak{B}_2$ respectively, such that $(\mathfrak{A}_1, \bar{a}_1)^* \Rrightarrow_{n,k} (\mathfrak{B}_1, \bar{b}_1)^*$ and $(\mathfrak{A}_2, \bar{a}_2)^* \Rrightarrow_{n,k} (\mathfrak{B}_2, \bar{b}_2)^*$, then*

$$(\mathfrak{A}_1 \oplus \mathfrak{A}_2, \bar{a}_1\bar{a}_2)^* \Rrightarrow_{n,k} (\mathfrak{B}_1 \oplus \mathfrak{B}_2, \bar{b}_1\bar{b}_2)^*.$$

Note that it is an immediate consequence that the same is true with $\equiv_{n,k}$ in place of $\Rrightarrow_{n,k}$.

Furthermore, this also extends to ordered sums of sequences. Moreover, we do not have to match the lengths of the sequences as long as they are long enough. Again, this is standard for the equivalence relation $\equiv_m$ [6, Ex. 2.3.13], for sequences of length at least 2^m and a slightly different notion of ordered sum. For our relations we obtain a tighter bound, so we prove this explicitly as the proof is instructive.

Define the following function ρ on pairs of natural numbers by recursion.

$$\begin{aligned}\rho(1,k) &= 2k+2\\ \rho(n+1,k) &= (k+2)(\rho(n,k)+1).\end{aligned}$$

A simple induction on n shows that $2(k+3)^n > \rho(n,k)$ for all $k,n \geq 1$.

▶ **Lemma 7.** *If $\mathfrak{A}$ and $\mathfrak{B}$ are ordered structures, $\mathfrak{A}^* \Rrightarrow_{n,k} \mathfrak{B}^*$ and $s,t \geq \rho(n,k)$, then $\big(\bigoplus_{1\leq i\leq s}\mathfrak{A}_i\big)^* \Rrightarrow_{n,k} \big(\bigoplus_{1\leq j\leq t}\mathfrak{B}_i\big)^*$, where $\mathfrak{A}_i \equiv_{n,k} \mathfrak{A}$ and $\mathfrak{B}_i \equiv_{n,k} \mathfrak{B}$ for all i.*

Proof. The proof is by induction on n. Suppose $n = 1$ and Spoiler plays a move choosing k elements from $\bigoplus_{1\leq i\leq s}\mathfrak{A}_i$. Suppose these are chosen from $\mathfrak{A}_{i_1},\dots,\mathfrak{A}_{i_l}$ with $1 \leq i_1 < \cdots < i_l \leq s$ for some $l \leq k$. Further, let $i_0 = 0$ and $i_{l+1} = t$. Since $l+2 \leq k+2 < 2k+2 = \rho(1,k) \leq s$, there is some p such that $i_{p+1} > i_p+1$. Duplicator must choose structures $(\mathfrak{B}_{j_q})_{1\leq q\leq l}$ in which to respond. Moreover, since $\mathfrak{A}_i$ and $\mathfrak{A}_{i+1}$ share an element for all i, whenever $i_{q+1} = i_q + 1$, we must choose $j_{q+1} = j_q + 1$.

Duplicator chooses values $0 = j_0 < j_1 < \cdots < j_l \leq j_{l+1} = t$ as follows. For all values of q from 0 to $p-1$, choose $j_{q+1} = j_q + 1$ if $i_{q+1} = i_q + 1$ and choose $j_{q+1} = j_q + 2$ otherwise. For all values of q from l down to $p+1$, choose $j_q = j_{q+1} - 1$ if $i_q = i_{q+1} - 1$ and choose $j_q = j_{q+1} - 2$ otherwise. Because $t \geq 2k+2$, this guarantees that $j_{p+1} > j_p + 1$. Thus, Duplicator can respond to the elements picked in $\mathfrak{A}_{i_p}$ with elements in $\mathfrak{B}_{j_p}$ by composing the winning strategies for $\mathfrak{A}_{i_p} \equiv_{1,k} \mathfrak{A} \Rrightarrow_{1,k} \mathfrak{B} \equiv_{1,k} \mathfrak{B}_{j_p}$ and this is a winning response. Moreover, by construction, this strategy maps the minimum and maximum elements of $\bigoplus_{1\leq i\leq s}\mathfrak{A}_i$ to the corresponding elements of $\bigoplus_{1\leq i\leq t}\mathfrak{B}_j$.

Now suppose $n \geq 2$ and the statement has been proved for $n-1$. Let Spoiler play a move choosing k elements from $\bigoplus_{1\leq i\leq s}\mathfrak{A}_i$, and again say these are chosen from $\mathfrak{A}_{i_1},\dots,\mathfrak{A}_{i_l}$ with $1 \leq i_1 < \cdots < i_l \leq s$ for some $l \leq k$. Further, define $i_0 = 0$ and $i_{l+1} = s$. Since $l+2 \leq k+2$ and $t \geq \rho(n,k) = (k+2)(\rho(n-1,k)+1)$, Duplicator can choose indices $0 = j_0 < j_1 \cdots < j_l < j_{l+1} = t$ so that for all p either $j_{p+1} - j_p = i_{p+1} - i_p$ or $(j_{p+1} - j_p), (i_{p+1} - i_p) \geq \rho(n-1,k)+1$. Now choose, for each p with $1 \leq p \leq l$ a response $\bar{b}_p$ for Duplicator in $\mathfrak{B}_{j_p}$ to the elements $\bar{a}_p$ chosen by Spoiler in $\mathfrak{A}_{i_p}$. We claim that

$$\Big(\bigoplus_{1\leq j\leq t}\mathfrak{B}_j, (\bar{b}_p)_{1\leq p\leq l}\Big)^* \Rrightarrow_{n-1,k} \Big(\bigoplus_{1\leq i\leq s}\mathfrak{A}_i, (\bar{a}_p)_{1\leq p\leq l}\Big)^*.$$

To prove this, it suffices to show for each p with $0 \leq p \leq l$ that

▷ Claim 8.

$$\Big(\bigoplus_{j_p<j\leq j_{p+1}}(\mathfrak{B}_j,\bar{b}_p)\Big)^* \Rrightarrow_{n-1,k} \Big(\bigoplus_{i_p<i\leq i_{p+1}}(\mathfrak{A}_i,\bar{a}_p)\Big)^*,$$

for then the claim follows by l applications of Lemma 6. Note that we have,

1. for all i,j that $\mathfrak{B}_j \Rrightarrow_{n-1,k} \mathfrak{A}_i$ by the assumption that $\mathfrak{A} \Rrightarrow_{n,k} \mathfrak{B}$; and
2. for all p we have $(\mathfrak{B}_{j_p},\bar{b}_p) \Rrightarrow_{n-1,k} (\mathfrak{A}_{i_p},\bar{a}_p)$ by the choice of $\bar{b}_p$ as Duplicator's winning response to Spoiler's choice of $\bar{a}_p$.

Now, for each value of p there are two possibilities:

case (i): $j_{p+1} - j_p = i_{p+1} - i_p$. In this case, the two sides of Claim 8 are the ordered sums of sequences of equal length. The corresponding pieces are all related by $\Rrightarrow_{n-1,k}$, either by 1. above for all except the last piece or by 2. for the last piece. Thus, Claim 8 is established by application of Lemma 6;

case (ii): $(j_{p+1}-j_p), (i_{p+1}-i_p) \geq \rho(n-1,k)+1$. In this case, the structures on the two sides of Claim 8 can be expressed as $\big(\bigoplus_{j_p<j\leq j_{p+1}-1}\mathfrak{B}_j\big)\oplus(\mathfrak{B}_{j_{p+1}},\bar{b}_{p+1})$ and $\big(\bigoplus_{i_p<i\leq i_{p+1}-1}\mathfrak{A}_i\big)\oplus(\mathfrak{A}_{i_{p+1}},\bar{a}_{p+1})$, respectively. Since $(j_{p+1}-j_p-1), (i_{p+1}-i_p-1) \geq \rho(n-1,k)$, we have by the induction hypothesis and 1. that $\big(\bigoplus_{j_p<j\leq j_{p+1}-1}\mathfrak{B}_j\big)^* \Rrightarrow_{n-1,k} \big(\bigoplus_{i_p<i\leq i_{p+1}-1}\mathfrak{A}_i\big)^*$. This, together with 2. and Lemma 6 establishes Claim 8, and hence the inductive step of the proof.

◀

Besides ordered sums, another key step in the inductive constructions of our structures is adding unary relations which include the minimum and maximum elements of a structure and adding binary relations which relate the minimum and maximum elements. These operations also behave well with respect to games. To be precise, suppose U is a unary relation symbol and T a binary relation symbol. Let $\mathfrak{A}$ be an ordered structure with minimum and maximum elements min and max respectively. Write $\mathfrak{A}_U$ for the structure obtained from $\mathfrak{A}$ by including min and max in the interpretation of U. Similarly, write $\mathfrak{A}_T$ for the structure obtained from $\mathfrak{A}$ by adding the pair $(\min, \max)$ to the interpretation of T. Note that we do not assume that U or T are in the vocabulary of $\mathfrak{A}$. If they are not, then their interpretations in $\mathfrak{A}_U$ and $\mathfrak{A}_T$ respectively contain nothing other than the elements added.

▶ **Lemma 9.** *Let $n, k \geq q$. If $\mathfrak{A}$ and $\mathfrak{B}$ are ordered structures for which $\mathfrak{A}^* \Rrightarrow_{n,k} \mathfrak{B}^*$ then $\mathfrak{A}_U \Rrightarrow_{n,k} \mathfrak{B}_U$ and $\mathfrak{A}_T \Rrightarrow_{n,k} \mathfrak{B}_T$.*

Proof. This is immediate from the fact that a Duplicator winning strategy between $\mathfrak{A}^*$ and $\mathfrak{B}^*$ must map the minimum elements of the two structures to each other, and similarly for the maximum. ◀

We are now ready to start inductively constructing the Duplicator winning strategy that establishes Lemma 5. We begin with games on some simple structures. For any $m \geq 1$ write L_m for the structure with exactly m elements and two binary relations $\leq$ and S where $\leq$ is a linear order and S the corresponding successor relation.

▶ **Lemma 10.** *If $m_1, m_2 > \rho(n,k)$ then $L^*_{m_1} \Rrightarrow_{n,k} L^*_{m_2}$.*

Proof. Note that L_m is the ordered sum of a sequence of $m-1$ copies of L_2, so the result follows immediately from Lemma 7. ◀

Without loss of generality, assume that the universe of L_m is $\{1,\dots,m\}$ and write G_m for the structure obtained from L_{m+1} by deleting the element $\lceil \frac{m}{2} \rceil$. Note that G_m is isomorphic to the structure obtained from L_m by removing from the relation S the pair $(\lceil \frac{m}{2} \rceil - 1, \lceil \frac{m}{2} \rceil)$.

▶ **Lemma 11.** *If $m_1, m_2 \geq 2k+2$ then $G^*_{m_1} \Rrightarrow_{1,k} L^*_{m_2}$.*

Proof. Since $G^*_{m_1}$ is a substructure of $L^*_{m_1+1}$, every existential sentence true in the former is also satisfied in the latter. Now, since $L^*_{m_1+1} \Rrightarrow_{1,k} L^*_{m_2}$ by Lemma 10, the result follows. ◀

The next two lemmas give us the base case of the inductive proof of Lemma 5.

▶ **Lemma 12.** $\mathsf{Tot}_{1,k} \Rrightarrow_{2,k} \mathsf{Gap}_{1,k}$

Proof. Recall that the $\{\leq, S\}$-reduct of $\mathsf{Tot}_{1,k}$ is the structure L_m for $m = 6(k+2)^2$. Note that $m > 2\rho(2,k) + (k+2)(2k+4)$. We think of this as composed of three segments: the first $\rho(2,k)$ elements; the last $\rho(2,k)$ elements and a *middle segment* containing the remainder. Suppose now that Spoiler chooses k elements $a_1 < \cdots < a_k$ from $\mathsf{Tot}_{1,k}$ in the first round of the game. Since the middle segment contains more than $(k+2)(2k+4)$ elements, it must contain an interval $[x, y]$ of $2k+2$ consecutive elements which are not chosen. Let us say that $a_i < x$ and $y < a_{i+1}$. Thus, we can write the $\{\leq, S\}$-reduct of $\mathsf{Tot}_{1,k}$ with the chosen elements as

$$(L_{m_1}, a_1, \ldots, a_i) \oplus L_{m_2} \oplus (L_{m_3}, a_{i+1}, \ldots, a_k),$$

where $m_1, m_3 \geq \rho(2,k)$ and $m_2 \geq 2k+2$.

Consider now $\mathsf{Gap}_{1,k}$. The $\{\leq, S\}$-reduct of this structure is G_m, which can also be written as $L_{\frac{m}{2}-k-1} \oplus G_{2k+2} \oplus L_{\frac{m}{2}-k-1}$. Since $\frac{m}{2} - k - 1 > \rho(2,k)$, we have by Lemma 10 that $L^*_{m_1} \Rrightarrow_{2,k} L^*_{\frac{m}{2}-k-1}$ and hence there is a choice of elements $b_1, \ldots, b_i$ such that $(L_{\frac{m}{2}-k-1}, b_1, \ldots, b_i)^* \Rrightarrow_{1,k} (L_{m_1}, a_1, \ldots, a_i)^*$. Similarly, there is a choice of elements $b_{i+1}, \ldots, b_k$ such that $(L_{\frac{m}{2}-k-1}, b_{i+1}, \ldots, b_k)^* \Rrightarrow_{1,k} (L_{m_3}, a_{i+1}, \ldots, a_k)^*$. Further, we know that $G^*_{2k+2} \Rrightarrow_{1,k} L^*_{m_3}$ by Lemma 11. Hence, by Lemma 6 we have that $L^*_m \Rrightarrow_{2,k} G^*_m$. The result now follows by Lemma 9 as $\mathsf{Tot}_{1,k}$ and $\mathsf{Gap}_{1,k}$ are obtained from L_m and G_m respectively by relating the minimum and maximum elements with the relation R. ◀

A similar pattern of argument is repeated in the second base case, and we will be less detailed in spelling it out.

▶ **Lemma 13.** $\mathfrak{N}^*_{1,k} \Rrightarrow_{3,k} \mathfrak{M}^*_{1,k}$.

Proof. Recall that $\mathfrak{N}_{1,k}$ is the ordered sum of $m = 4(k+3)^3 + 2k + 1$ copies of $\mathsf{Gap}_{1,k}$. So $\mathfrak{N}_{1,k} = \bigoplus_{i \in [m]} \mathfrak{G}_i$. Note that $m > 2\rho(3,k) + k + 1$. Suppose now that Spoiler chooses k elements $a_1 < \cdots < a_k$ in the first round of the game. Thus, there is an index i in the middle segment of $[m]$ of length $k+1$ such that $\mathfrak{G}_i$ does not contain a chosen element and we can write $\mathfrak{N}_{1,k}$ with the chosen elements as $(\bigoplus_{i \in [m_1]} \mathfrak{G}_i, a_1, \ldots, a_j) \oplus \mathsf{Gap}_{1,k} \oplus (\bigoplus_{i \in [m_2]} \mathfrak{G}_i, a_{j+1}, \ldots, a_k)$, where $m_1, m_2 > \rho(3,k)$.

On the other side, $\mathfrak{M}_{1,k} = \bigoplus_{i \in [(m-1)/2]} \mathfrak{G}_i \oplus \mathsf{Tot}_{1,k} \oplus \bigoplus_{i \in [(m-1)/2]} \mathfrak{G}_i$. Since $(m-1)/2 > \rho(3,k)$, by Lemma 7 we have $\left(\bigoplus_{i \in [m_1]} \mathfrak{G}_i\right)^* \Rrightarrow_{3,k} \left(\bigoplus_{i \in [(m-1)/2]} \mathfrak{G}_i\right)^*$ and $\left(\bigoplus_{i \in [m_2]} \mathfrak{G}_i\right)^* \Rrightarrow_{3,k} \left(\bigoplus_{i \in [(m-1)/2]} \mathfrak{G}_i\right)^*$. Thus, we can find elements $b_1, \ldots, b_k$ such that

$$\Big(\bigoplus_{i \in [\frac{m-1}{2}]} \mathfrak{G}_i, b_1, \ldots, b_j \Big)^* \Rrightarrow_{2,k} \Big(\bigoplus_{i \in [m_1]} \mathfrak{G}_i, a_1, \ldots, a_j \Big)^*$$

and

$$\Big(\bigoplus_{i \in [\frac{m-1}{2}]} \mathfrak{G}_i, b_{j+1}, \ldots, b_k \Big)^* \Rrightarrow_{2,k} \Big(\bigoplus_{i \in [m_2]} \mathfrak{G}_i, a_{j+1}, \ldots, a_k \Big)^*.$$

Combining this with the fact that $\mathsf{Tot}_{1,k} \Rrightarrow_{2,k} \mathsf{Gap}_{1,k}$ by Lemma 12, we get by Lemma 6 that $(\mathfrak{M}_{1,k}, b_1, \ldots, b_k)^* \Rrightarrow_{2,k} (\mathfrak{N}_{1,k}, a_1, \ldots, a_k)^*$ and the result follows. ◀

These last two lemmas form the base case of the induction that establishes the main result. Where the argument is analogous to the previous ones, we skim over the details.

Proof of Lemma 5. We prove the following two statements by induction for all $n, k \geq 1$.

1. $\mathsf{Tot}_{n,k} \Rrightarrow_{2n,k} \mathsf{Gap}_{n,k}$
2. $\mathfrak{N}^*_{n,k} \Rrightarrow_{2n+1,k} \mathfrak{M}^*_{n,k}$

The case of $n = 1$ is established in Lemmas 12 and 13 respectively. Suppose now $n \geq 2$ and we have established both statements for $n - 1$.

First, recall that $\mathfrak{M}^+_{n-1,k}$ and $\mathfrak{N}^+_{n-1,k}$ are obtained from $\mathfrak{M}_{n-1,k}$ and $\mathfrak{N}_{n-1,k}$ respectively by including their minimum and maximum elements in the unary relation P_n and the binary relation S_n. Thus, by the induction hypothesis and Lemma 9, we have $\mathfrak{N}^+_{n-1,k} \Rrightarrow_{2n-1,k} \mathfrak{M}^+_{n-1,k}$.

$\mathsf{Tot}_{n,k}$ consists of the ordered sum of a sequence of $m = 4(k+3)^{2n}+2k+1 > 2\rho(2n, k)+k+1$ copies of $\mathfrak{M}^+_{n-1,k}$, along with a relation R_n containing just the pair with the minumum and maximum elements. When Spoiler chooses k elements from this structure, we can find a copy of $\mathfrak{M}^+_{n-1,k}$ in the middle $k+1$ copies that has no element chosen and there are $m_1 > \rho(2n, k)$ copies before it and $m_2 > \rho(2n, k)$ copies after it. We can similarly express $\mathsf{Gap}_{n,k}$ as the ordered sum of a sequence of $(m - 1)/2 > \rho(2n, k)$ copies of $\mathfrak{M}^+_{n-1,k}$, followed by a copy of $\mathfrak{N}^+_{n-1,k}$ and a further $(m - 1)/2 > \rho(2n, k)$ copies of $\mathfrak{M}^+_{n-1,k}$. Lemma 7 tells us that we can find a response to the chosen elements in the first and third parts. This combined with the fact that $\mathfrak{N}^+_{n-1,k} \Rrightarrow_{2n-1,k} \mathfrak{M}^+_{n-1,k}$ and using Lemma 9 to expand to the relation R_n gives us the desired result.

The argument for the second statement is entirely analogous. $\mathfrak{N}_{n,k}$ is the ordered sum of a sequence of $m = 4(k + 3)^{2n+1} + 2k + 1 > 2\rho(2n + 1, k) + k + 1$ copies of $\mathsf{Gap}_{n,k}$. When Spoiler chooses k elements from this structure, we can find a copy of $\mathsf{Gap}_{n,k}$ in the middle $k+1$ copies that has no element chosen and there are $m_1 > \rho(2n + 1, k)$ copies before it and $m_2 > \rho(2n + 1, k)$ copies after it. Since $\mathfrak{M}_{n,k}$ has $(m - 1)/2 > \rho(2n + 1, k)$ copies of $\mathsf{Gap}_{n,k}$ followed by a copy of $\mathsf{Tot}_{n,k}$ and a further $(m - 1)/2 > \rho(2n + 1, k)$ copies of $\mathsf{Gap}_{n,k}$, by Lemma 7 we can find responses to the chosen elements in the first and third parts. We have already proved that $\mathsf{Tot}_{n,k} \Rrightarrow_{2n,k} \mathsf{Gap}_{n,k}$. We can combine these to complete the proof. ◀

5 Concluding Remarks

We have established in this paper that the extension-closed properties of finite structures that are definable in first-order logic are not contained in any fixed quantifier-alternation fragment of the logic. The construction of the sentences demonstrating this is recursive. It builds on a base of known counter-examples for the Łoś-Tarski theorem in the finite and lifts them up inductively. Also, the argument for showing inexpressibility in fixed levels of the quantifier-alternation hierarchy builds on the game arguments used with previously known examples and builds on them systematically using a form of Feferman-Vaught decomposition (see [11]) for ordered sums of structures.

Our result actually establishes that for all *odd* $n > 1$, there is a Σ_n-definable property that is closed under extensions but not definable by a Π_n sentence. Interestingly, this is not true for even values of n. In particular, it is known that every Σ_2-definable extension-closed property is already definable in Σ_1. This observation is credited to Compton in [7] and may also be found in [16]. We do not know, however, whether for even $n > 2$, the extension closed properties in Σ_n can all be expressed in Π_n or even Σ_{n-1}. On the other hand, we are able to observe that for all even $n > 1$, there is a Π_n-definable extension-closed property that is not in Σ_n. This is a direct consequence of our proof. Indeed, consider the sentence φ_n obtained from $\mathrm{SomeTotalR}_n$ by removing the two outer existential quantifiers and replacing the resulting free variables with new constants a and b. This is a Π_{2n} sentence and is easily seen to define an extension-closed class. By our construction, φ_n is satisfied in the expansion of $\mathsf{Tot}_{n,k}$ where a and b are the minimum and maximum elements respectively. At the same time φ_n is false in the similar expansion of $\mathsf{Gap}_{n,k}$. Since we showed that $\mathsf{Tot}^*_{n,k} \Rrightarrow_{2n,k} \mathsf{Gap}^*_{n,k}$, it follows that φ_n is not equivalent to a Σ_{2n} sentence. It would be interesting to complete the picture of extension-closed properties in the remaining quantifier-alternation fragments, specifically Σ_n for even values of n and Π_n for odd values of n.

It is a feature of our construction that the vocabulary σ_n in which we construct the sentences which separate extension-closed Σ_{2n+1} from Π_{2n+1} grows with n. Could our results be established in a fixed vocabulary? Indeed, does something like Theorem 4 hold for finite graphs?

Another interesting direction left open from our work is the relation with `Datalog`$(\neg)$. All the extension-closed FO-definable properties we construct are also definable in `Datalog`$(\neg)$. Rosen and Weinstein [13] ask whether this is true for all FO-definable extension-closed properties, and this remains open. Indeed, it is conceivable that we have an extension-preservation theorem for least fixed-point logic in the finite, so that even all LFP-definable extension-closed properties are in `Datalog`$(\neg)$.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

2 Albert Atserias, Anuj Dawar, and Martin Grohe. Preservation under extensions on well-behaved finite structures. *SIAM Journal on Computing*, 38:1364–1381, 2008.

3 Chen C. Chang and Howard J. Keisler. *Model Theory*, volume 73. Elsevier, 1990.

4 Anuj Dawar. Finite model theory on tame classes of structures. In *MFCS*, volume 4708 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2007.

5 Anuj Dawar and Stephan Kreutzer. On Datalog vs. LFP. In *International Colloquium on Automata, Languages, and Programming*, pages 160–171. Springer, 2008.

6 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Springer Science & Business Media, 2005.

7 Yuri Gurevich. Toward logic tailored for computational complexity. In M. Richter et al., editors, *Computation and Proof Theory*, pages 175–216. Springer Lecture Notes in Mathematics, 1984.

8 Wilfrid Hodges. *Model theory*, volume 42. Cambridge University Press, 1993.

9 Phokion G Kolaitis and Moshe Y Vardi. On the expressive power of Datalog: tools and a case study. *Journal of Computer and System Sciences*, 51(1):110–134, 1995.

10 Leonid Libkin. *Elements of Finite Model Theory*. Springer-Verlag, 2004.

11 Johann A. Makowsky. Algorithmic uses of the Feferman-Vaught theorem. *Ann. Pure Appl. Log.*, 126:159–213, 2004. doi:10.1016/j.apal.2003.11.002.

12 Eric Rosen. Some aspects of model theory and finite structures. *Bulletin of Symbolic Logic*, 8:380–403, 2002.

13 Eric Rosen and Scott Weinstein. Preservation theorems in finite model theory. In *Logical and Computational Complexity. Selected Papers.*, pages 480–502, 1994. doi:10.1007/3-540-60178-3_99.

14 Benjamin Rossman. Homomorphism preservation theorems. *Journal of the ACM*, 55, 2008.

15 Abhisekh Sankaran. Revisiting the generalized Łoś-Tarski theorem. In *Logic and Its Applications - 8th Indian Conference, ICLA 2019*, pages 76–88, 2019. doi:10.1007/978-3-662-58771-3_8.

16 Abhisekh Sankaran, Bharat Adsul, Vivek Madan, Pritish Kamath, and Supratik Chakraborty. Preservation under substructures modulo bounded cores. In *Logic, Language, Information and Computation - 19th International Workshop, WoLLIC 2012*, pages 291–305, 2012. doi:10.1007/978-3-642-32621-9_22.

17 William W. Tait. A counterexample to a conjecture of Scott and Suppes. *J. Symb. Logic*, 24(1):15–16, 1959.

Realizability with Stateful Computations for Nonstandard Analysis

Bruno Dinis
Faculdade de Ciências, University of Lisbon, Portugal
bmdinis@fc.ul.pt

Étienne Miquey
ÉNS de Lyon, Université de Lyon, LIP, France
etienne.miquey@ens-lyon.fr

Abstract

In this paper we propose a new approach to realizability interpretations for nonstandard arithmetic. We deal with nonstandard analysis in the context of intuitionistic realizability, focusing on the Lightstone-Robinson construction of a model for nonstandard analysis through an ultrapower. In particular, we consider an extension of the λ-calculus with a memory cell, that contains an integer (the state), in order to indicate in which slice of the ultrapower $\mathcal{M}^{\mathbb{N}}$ the computation is being done. We shall pay attention to the nonstandard principles (and their computational content) obtainable in this setting. We then discuss how this product could be quotiented to mimic the Lightstone-Robinson construction.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Constructive mathematics

Keywords and phrases realizability, nonstandard analysis, states, glueing, ultrafilters, Łoś' theorem

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.19

Funding *Bruno Dinis*: The author acknowledges the support of FCT - Fundação para a Ciência e Tecnologia under the projects UIDB/04561/2020 and UIDP/04561/2020, and the research center Centro de Matemática, Aplicações Fundamentais e Investigação Operacional, Universidade de Lisboa.
Étienne Miquey: The author was supported by the Paris Ile-de-France Region and the Inria research team Deducteam (LSV, ÉNS Paris-Saclay).

Acknowledgements The authors would like to thank Alexandre Miquel for suggesting several ideas at the root of this work and Valentin Blot and Mikhail Katz, as well as the anonymous reviewers, for their accurate remarks and suggestions.

1 Introduction

In this paper we propose a new approach to realizability interpretations for nonstandard arithmetic. On the one hand, we deal with nonstandard analysis in the context of intuitionistic realizability. On the other hand, we focus on Lightstone and Robinson's construction of a model for nonstandard analysis through an ultrapower [23].

Throughout the history of mathematics, infinitesimals were crucial for the intuitive development of mathematical knowledge by authors such as Archimedes, Stevin, Fermat, Leibniz, Euler and Cauchy, to name but a few (see e.g. [15, 4, 3]). In particular, in Leibniz's Calculus one may recognize calculation rules – sometimes called the *Leibniz rules* [24, 7, 10] – which correspond to heuristic intuitions for how the infinitesimals should operate under calculations: the sum and product of infinitesimals is infinitesimal, the product of a limited number (i.e. not infinitely large) with an infinitesimal is infinitesimal, . . .

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 19; pp. 19:1–19:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In [35, 36] Robinson showed that, in the setting of model theory, it is possible to extend usual mathematical sets ($\mathbb{N}$, $\mathbb{R}$, etc.) witnessing the existence of new elements, the so-called *nonstandard* individuals. In this way, it is possible to deal consistently with infinitesimal and infinitely large numbers via ultraproducts and ultrapowers, in a way that is consistent with the Leibniz rules. Since the extended structures are nonstandard models of the original structures, this new setting was dubbed *nonstandard analysis*.

These constructions are meant to simplify doing mathematics: notions like limits or continuity can for instance be given a simpler form in nonstandard analysis. Later in the 70s, Nelson developed a syntactical approach to nonstandard analysis, introducing in particular three key principles: idealization, standardization and transfer [31]. The validity of these principles for constructive mathematics has been studied in many different settings, in particular, following some pioneer work by Moerdijk, Palmgren and Avigad [29, 30, 2] in nonstandard intuitionistic arithmetic, several recent works, inspired by Nelson's approach, lead to interpretations of nonstandard theories in intuitionistic realizability models [6, 8, 13, 9].

The very first ideas of *realizability* are to be found in the Brouwer-Heyting-Kolmogorov interpretation [14, 17], which identifies evidences and computing proofs (the realizers). Realizability was designed by Kleene to interpret the computational content of the proofs of Heyting arithmetic [16], and was later extended to more expressive frameworks [11, 18, 20]. While the Curry-Howard isomorphism focuses on a syntactical correspondence between proofs and programs, realizability rather deals with the (operational) semantics of programs: a *realizer* of a formula A is a program which *computes* adequately with the specification that A provides. As such, realizability constitutes a technique to develop new models of a wide class of theories (from Heyting arithmetic to Zermelo-Fraenkel set theory), whose algebraic structures has been studied in [38, 22, 27].

With the development of his classical realizability, Krivine evidences the fact that extending the λ-calculus with new programming instructions may result in getting new reasoning principles: `call/cc` to get classical logic [12, 20], `quote` for dependent choice [19], etc. In this paper, we follow this path to show how the addition of a monotonic reference allows us to get a realizability interpretation for nonstandard analysis. The realizability interpretation proposed here can be understood as a computational interpretation of the ultraproduct construction in [23], where the value of the reference indicates the slice of the product in which the computation takes place. In particular, we obtain a realizer for the idealization principle whose computational behaviour increases the reference in the manner of a diagonalization process.

Outline

We start this paper by recalling the main ideas of the ultraproduct construction (Section 2) and the definition of a standard realizability interpretation for second-order Heyting arithmetic (Section 3). We then introduce stateful computations and our notion of realizability with slices in Section 4. As shown in Section 5, this interpretation provides us with realizers for several nonstandard reasoning principles. Finally, we discuss the possibility of taking a quotient for this interpretation in Section 6.1 and we conclude the paper in Section 6.2 with a comparison to related works and questions left for future work.

N.B.: due to the page limit, proofs sketches are given in the appendices.

2 The ultrapower construction

The main contribution of this paper consists in defining a realizability interpretation to give a computational content to the ultrapower construction of Robinson and Lightstone in [23]. We shall begin by briefly explaining how this construction works in the realm of model theory.

First, recall that an *ultrafilter* over a set I is a filter $\mathcal{U} \subseteq \mathcal{P}(I)$ such that for any $F \in \mathcal{P}(I)$, either F or its complement $\overline{F}$ are in $\mathcal{U}$. For instance, the set of cofinite subsets of $\mathbb{N}$ defines the so-called *Fréchet filter*, which is not an ultrafilter since it contains neither the set of even natural numbers nor the set of odd natural numbers. Nonetheless, it is well-known that any filter $\mathcal{F}$ over an infinite set I is contained in an ultrafilter $\mathcal{U}$ over I: this is the so-called *ultrafilter principle*. An ultrafilter that contains the Fréchet filter is called a *free ultrafilter*. The existence of free ultrafilters was proved by Tarski in 1930 [37] and is in fact a consequence of the axiom of choice.

Given two sets V and I and an ultrafilter $\mathcal{U}$ over I, we can define an equivalence relation $\cong_{\mathcal{U}}$ over V^I by $u \cong_{\mathcal{U}} v \triangleq \{i \in I : u_i = v_i\} \in \mathcal{U}$. We write $V^I/\mathcal{U}$ for the set obtained by performing a quotient on the set V^I by this equivalence relation, which is called an *ultrapower*.

Consider a theory $\mathcal{T}$ (say ZFC) and its language $\mathcal{L}$, for which we assume the existence of a model $\mathcal{M}$. The goal is to build a nonstandard model $\mathcal{M}^*$ of the theory $\mathcal{T}$ that validates new principles. Let us denote by $\mathcal{V}$ the set which interprets individuals in $\mathcal{M}$, and let us fix a free ultrafilter $\mathcal{U}$ over $\mathbb{N}$. Roughly speaking, the new model $\mathcal{M}^*$ is defined as the ultrapower $\mathcal{M}^{\mathbb{N}}/\mathcal{U}$. Individuals are interpreted by functions in $\mathcal{V}^{\mathbb{N}}$ while the validity of a relation $R(x_1, ..., x_k)$ (where the x_i are interpreted by f_i, for $i \in \{1, ..., k\}$) is defined by

$$\mathcal{M}^* \vDash R(f_1, ..., f_k) \qquad \text{iff} \qquad \{n \in \mathbb{N} : \mathcal{M} \vDash R(f_1(n), ..., f_k(n))\} \in \mathcal{U}.$$

We can now extend the language with a new predicate $\mathrm{st}(x)$ to express that x is *standard*. Standard elements are defined as the ones that, with respect to $\cong_{\mathcal{U}}$, are equivalent to constant functions, i.e. $\mathcal{M}^* \vDash \mathrm{st}(f)$ if and only if there exists $p \in \mathbb{N}$ such that $\{n \in \mathbb{N} : f(n) = p\} \in \mathcal{U}$. Formulas that involve this new predicate are called *external*, while formulas of the original language $\mathcal{L}$ are called *internal*.

Lightstone and Robinson's construction relies on the well-known Łoś' theorem [33] which states that if φ is an internal formula (with parameters in $\mathcal{V}^{\mathbb{N}}$), then $\mathcal{M}^* \vDash \varphi$ if and only $\{n \in \mathbb{N} : \mathcal{M} \vDash \overline{\varphi}^n\} \in \mathcal{U}$, where $\overline{\varphi}^n$ refers to the formula φ whose parameters have been replaced by their values in n. This construction indeed defines a model of $\mathcal{T}$ which satisfies other relevant properties, namely transfer, idealization and standardization. As a consequence of Łoś' theorem, to see that an internal formula $\varphi(x)$ holds for all elements, it is enough to see that it holds for all standard elements: this is the *transfer* principle. In our setting, *idealization* amounts to a diagonalization process: it is for instance easy to see that if one defines $\delta : n \mapsto n$ (where we, with abuse of notation, write n for both the natural number n and its interpretation in $\mathcal{V}$), then $\mathcal{M}^* \vDash \forall x.(\mathrm{st}(x) \to x < \delta)$. Finally, *standardization* is a sort of "comprehension scheme" which states that we can specify subsets of standard sets by giving a membership criterion for standard elements (by means of an internal formula).

3 Realizability in a nutshell

3.1 Heyting second-order arithmetic

We start by introducing the terms and formulas of Heyting second-order arithmetic (HA2), for which we follow Miquel's presentation [25]. Second-order formulas are build on top of first-order arithmetical expressions, by means of logical connectives, first- and second-order quantifications and primitive predicates. We use upper case letters for second-order variables and lower case for first-order ones. We use a primitive predicate $\mathrm{Nat}(e)$ to denote that e is a natural number (0 then has type $\mathrm{Nat}(0)$ and the term $\mathsf{s}\,t$ has type $\mathrm{Nat}(S(e))$ provided that t has type $\mathrm{Nat}(e)$). We consider the usual λ-calculus terms extended with pairs, projections (written π_i), injections (written ι_i), case analysis, natural numbers and a recursion operator:

$$\begin{array}{lrcl}
\textbf{1st-order expressions} & e & ::= & x \mid 0 \mid S(e) \mid f(e_1,\dots,e_n) \\
\textbf{Formulas} & A,B & ::= & \mathrm{Nat}(e) \mid X(e_1,\dots,e_n) \mid A \to B \mid A \wedge B \mid A \vee B \\
 & & & \mid \forall x.A \mid \exists x.A \mid \forall X.A \mid \exists X.A \\
\textbf{Terms} & t,u & ::= & x \mid 0 \mid \mathsf{s} \mid \mathsf{rec} \mid \lambda x.t \mid t\,u \mid (t,u) \mid \pi_1(t) \mid \pi_2(t) \\
 & & & \mid \iota_1(t) \mid \iota_2(t) \mid \mathsf{case}\ t\,\{\iota_1(x_1) \mapsto t_1 | \iota_2(x_2) \mapsto t_2\}
\end{array}$$

where $f : \mathbb{N}^n \to \mathbb{N}$ is any arithmetical function. We write Λ for the set of all closed λ-terms.

As in Miquel's presentation, we consider formulas up to the following congruences:

$$(\exists x.A) \to B \cong \forall x.(A \to B) \qquad\qquad (\exists X.A) \to B \cong \forall X.(A \to B) \tag{1}$$

These congruences allow us to avoid having elimination rules for the existential quantifiers, thus simplifying the resulting type system. The type system, which is given in Figure 1, corresponds to the usual rules of natural deduction. The reader may observe that we do not give computational content to quantifications.

In the sequel, we make use of the following usual abbreviations:

$$\begin{array}{rcl|rcl|rcl}
\mathsf{s}^{n+1}0 & \triangleq & \mathsf{s}\,(\mathsf{s}^n 0) & \top & \triangleq & \exists X.X & e = e' & \triangleq & \forall Z.(Z(e) \to Z(e')) \\
\overline{n} & \triangleq & \mathsf{s}^n 0 & \bot & \triangleq & \forall X.X & \forall^{\mathbb{N}} x.A & \triangleq & \forall x.(\mathrm{Nat}(x) \to A) \\
 & & & \neg A & \triangleq & A \to \bot & \exists^{\mathbb{N}} x.A & \triangleq & \exists x.(\mathrm{Nat}(x) \wedge A)
\end{array}$$

It is well-known that the above definition of equality (often called *Leibniz law*) enjoys the usual expected properties (reflexivity, symmetry, transitivity) and allows to perform substitution of equal terms. The quantifications $\forall^{\mathbb{N}} x.A$ and $\exists^{\mathbb{N}} x.A$ are often said to be *relativized* to natural numbers.

The one-step (weak) reduction over terms is defined by the following rules:

$$\frac{}{(\lambda x.t)u \rhd_\beta t[u/x]} \qquad \frac{}{\mathsf{rec}\ u_0\,u_1\,0 \rhd_\beta u_0} \qquad \frac{}{\mathsf{rec}\ u_0\,u_1\,(\mathsf{s}\,t) \rhd_\beta u_1\,t\,(\mathsf{rec}\ u_0\,u_1\,t)}$$

$$\frac{}{\pi_1(t,u) \rhd_\beta t} \qquad \frac{}{\pi_2(t,u) \rhd_\beta u} \qquad \frac{}{\mathsf{case}\ \iota_i(t)\,\{\iota_1(x_1) \mapsto t_1 | \iota_2(x_2) \mapsto t_2\} \rhd_\beta t_i[t/x_i]}$$

We write $\to_\beta$ for the congruent reflexive-transitive closure of $\rhd_\beta$. The reduction $\to_\beta$ is known to be confluent, type-preserving and normalizing on typed terms [5].

3.2 Realizability interpretation of HA2

In this subsection we define the realizability interpretation of the type system defined in Figure 1, in which formulas are interpreted as *saturated sets* of terms, i.e. as sets of closed terms $S \subseteq \Lambda$ such that $t \to_\beta t'$ and $t' \in S$ imply that $t \in S$. We write $\mathbf{SAT}$ to denote the set of all saturated sets and, given a formula A, we call *truth value* its realizability interpretation.

▶ **Definition 1** (Valuation). A valuation *is a function ρ that associates a natural number $\rho(x)$ to every first-order variable x and* a truth value function *$\rho(X)$, i.e. a function in $\mathbb{N}^k \to \mathbf{SAT}$ to every second-order variable X of arity k.*

1. *Given a valuation ρ, a first-order variable x and a natural number n, we denote by $\rho, x \mapsto n$ the valuation defined by $(\rho, x \mapsto n) \triangleq \rho_{|\,\mathrm{dom}(\rho)\setminus\{x\}} \cup \{x \mapsto n\}$.*

$$\frac{}{\Gamma \vdash 0 : \mathrm{Nat}(0)}\ (0) \qquad \frac{}{\Gamma \vdash \mathsf{s} : \forall^{\mathbb{N}} x.\mathrm{Nat}(S(x))}\ (S)$$

$$\frac{}{\Gamma \vdash \mathsf{rec} : \forall Z.Z(0) \to (\forall^{\mathbb{N}} y.(Z(y) \to Z(S(y)))) \to \forall^{\mathbb{N}} x.Z(x)}\ (\mathrm{rec}) \qquad \frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}\ (\mathrm{Ax})$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x\,.\,t : A \to B}\ (\to_I) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}\ (\to_E) \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash (t, u) : A \wedge B}\ (\wedge_I)$$

$$\frac{\Gamma \vdash t : A \wedge B}{\Gamma \vdash \pi_1(t) : A}\ (\wedge_E^1) \qquad \frac{\Gamma \vdash t : A \wedge B}{\Gamma \vdash \pi_2(t) : B}\ (\wedge_E^2) \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash \iota_1(t) : A \vee B}\ (\vee_I^1) \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \iota_2(t) : A \vee B}\ (\vee_I^2)$$

$$\frac{\Gamma \vdash t : A_1 \vee A_2 \quad \Gamma, x_i : A_i \vdash t_i : C}{\Gamma \vdash \mathsf{case}\ t\,\{\iota_1(x_1) \mapsto t_1 | \iota_2(x_2) \mapsto t_2\} : C}\ (\vee_E) \qquad \frac{\Gamma \vdash t : A[x := n]}{\Gamma \vdash t : \exists x.A}\ (\exists_I^1) \qquad \frac{\Gamma \vdash t : \forall x.A}{\Gamma \vdash t : A[x := n]}\ (\forall_E^1)$$

$$\frac{\Gamma \vdash t : A \quad x \notin FV(\Gamma)}{\Gamma \vdash t : \forall x.A}\ (\forall_I^1) \qquad \frac{\Gamma \vdash t : A[X(x_1, \dots, x_n) := B]}{\Gamma \vdash t : \exists X.A}\ (\exists_I^2)$$

$$\frac{\Gamma \vdash t : \forall X.A}{\Gamma \vdash t : A[X(x_1, \dots, x_n) := B]}\ (\forall_E^2) \qquad \frac{\Gamma \vdash t : A \quad X \notin FV(\Gamma)}{\Gamma \vdash t : \forall X.A}\ (\forall_I^2) \qquad \frac{\Gamma \vdash t : A' \quad A \cong A'}{\Gamma \vdash t : A}\ (\cong)$$

Figure 1 Type system.

2. *Given a valuation ρ, a second-order variable X of arity k and a truth value function $F : \mathbb{N}^k \to \mathbf{SAT}$, the valuation defined by $(\rho, X \mapsto F) \triangleq \rho_{|\,\mathrm{dom}(\rho) \setminus \{X\}} \cup \{X \mapsto F\}$ will be denoted by $\rho, X \mapsto F$.*

We say that a valuation ρ is closing *the formula A if $FV(A) \subseteq \mathrm{dom}(\rho)$.*

▶ **Definition 2** (Realizability interpretation). *We interpret closed arithmetical expressions e in the standard model of first-order Peano arithmetic $\mathbb{N}$. Given a valuation ρ and a first-order expression e (whose variables are in the domain of ρ) we denote its interpretation by $[\![e]\!]_\rho$. The interpretation of a formula A together with a valuation ρ closing A is the set $|A|_\rho$ defined inductively according to the following clauses:*

$$\begin{array}{rcl} |\mathrm{Nat}(e)|_\rho & \triangleq & \{t \in \Lambda : t \to_\beta \mathsf{s}^n 0,\ \textit{where}\ n = [\![e]\!]_\rho\} \\ |X(e_1, \dots, e_n)|_\rho & \triangleq & \rho(X)([\![e_1]\!]_\rho, \dots, [\![e_n]\!]_\rho) \\ |A \to B|_\rho & \triangleq & \{t \in \Lambda : \forall u \in |A|_\rho.(t\,u \in |B|_\rho)\} \\ |A_1 \wedge A_2|_\rho & \triangleq & \{t \in \Lambda : \pi_1(t) \in |A_1|_\rho \wedge \pi_2(t) \in |A_2|_\rho\} \\ |A_1 \vee A_2|_\rho & \triangleq & \{t \in \Lambda : \exists i \in \{1, 2\}.\,\mathsf{case}\ t\,\{\iota_1(x_1) \mapsto x_1 | \iota_2(x_2) \mapsto x_2\} \in |A_i|_\rho\} \\ |\forall x.A|_\rho & \triangleq & \bigcap_{n \in \mathbb{N}} |A|_{\rho, x \mapsto n} \qquad\qquad |\forall X.A|_\rho \ \triangleq\ \bigcap_{F : \mathbb{N}^k \to \mathbf{SAT}} |A|_{\rho, X \mapsto F} \\ |\exists x.A|_\rho & \triangleq & \bigcup_{n \in \mathbb{N}} |A|_{\rho, x \mapsto n} \qquad\qquad |\exists X.A|_\rho \ \triangleq\ \bigcup_{F : \mathbb{N}^k \to \mathbf{SAT}} |A|_{\rho, X \mapsto F} \end{array}$$

Observe that in the previous definition, the universal quantifications cannot be seen as generalized conjunctions. Indeed, the conjunction is given computational content through pairs, while the universal quantifications are defined as intersections of truth values.

It is easy to see that for any formula A and any valuation ρ closing A, one has $|A|_\rho \in \mathbf{SAT}$. As it turns out, the congruences defined by Equation (1) are sound w.r.t. the interpretation.

▶ **Proposition 3** ([25]). *If A and A' are two formulas of HA2 such that $A \cong A'$, then for all valuations ρ closing both A and A' we have $|A|_\rho = |A'|_\rho$.*

In order to show that the realizability interpretation is adequate with respect to the type system we need the following preliminary notions.

▶ **Definition 4** (Substitution). *A* substitution *is a finite function σ from λ-variables to closed λ-terms. Given a substitution σ, a λ-variable x and a closed λ-term u, we denote by $(\sigma, x := u)$ the substitution defined by $(\sigma, x := u) \triangleq \sigma_{|\,\mathrm{dom}(\sigma)\setminus\{x\}} \cup \{x := u\}$.*

▶ **Definition 5.** *Given a context Γ and a valuation ρ closing the formulas in Γ, we say that a substitution σ* realizes *$\rho(\Gamma)$ and write $\sigma \Vdash \rho(\Gamma)$ if $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\sigma)$ and $\sigma(x) \in |A|_\rho$ for every declaration $(x : A) \in \Gamma$.*

▶ **Definition 6.** *A typing judgement $\Gamma \vdash t : A$ is* adequate *if for all valuations ρ closing A and Γ and for all substitutions $\sigma \Vdash \rho(\Gamma)$ we have $\sigma(t) \in |A|_\rho$. More generally, we say that an inference rule $\dfrac{J_1 \quad \cdots \quad J_n}{J_0}$ is adequate if the adequacy of all typing judgements $J_1, \ldots, J_n$ implies the adequacy of the typing judgement J_0.*

▶ **Theorem 7** (Adequacy [25]). *The typing rules of Figure 1 are adequate.*

▶ **Corollary 8.** *If $\Gamma \vdash t : A$ is derivable, then it is adequate.*

The adequacy theorem is the key result when defining realizability interpretations in that fundamental properties stem from it. For example, we have the following corollary.

▶ **Corollary 9** (Consistency). *There is no proof term t such that $\vdash t : \bot$.*

We would like to point out that the proof of adequacy is very flexible. Indeed, if one wants to add a new instruction to the language of terms via its typing rule, it is enough to check that this typing rule is adequate while the remainder of the proof is exactly the same.

3.3 Introducing value restrictions

The realizability interpretation of Definition 2 is also flexible regarding the set of formulas that are interpreted. We illustrate this point here by introducing a new construction extending formulas. For these formulas we shall not give any typing rule, instead we will see how this construction allows us to enforce value restrictions, which will turn out to be crucial afterwards in a setting where stateful computations occur. We start by defining the subset $\mathcal{V} \subseteq \Lambda$ of *values* by the following grammar:

$$\textbf{Values} \qquad V \quad ::= \quad 0 \mid \mathsf{s}\, V \mid \lambda x.t \mid (V_1, V_2) \mid \iota_i(V)$$

Observe that variables are not values, otherwise the system would not be stable by substitution. In the remainder of this paper, we adopt the convention that λ-terms are denoted by lowercase letters $t, u, ...$ while uppercase letters $V, W, ...$ refer to values.

Distinguishing the set of values allows for instance to restrict the β-reduction rule to applications of functions to values:

$$\frac{}{(\lambda x.t)V \rhd_{\mathrm{v}} t[V/x]} \qquad \frac{t \rhd_{\mathrm{v}} t'}{t\,u \rhd_{\mathrm{v}} t'\,u} \qquad \frac{u \rhd_{\mathrm{v}} u'}{V\,u \rhd_{\mathrm{v}} V\,u'}$$

The reflexive transitive closure $\to_{\mathrm{v}}$ of the one-step reduction $\rhd_{\mathrm{v}}$ is known as the (left-to-right) *call-by-value* evaluation strategy. While it is well-known that the reduction system of the λ-calculus is confluent, so that the choice of a particular evaluation strategy does not have any consequence in terms of expressiveness, this is no longer the case when side effects (such as stateful computations in the next sections) come into play.

To enforce value restrictions, let us now extend the language of formulas with a new construction $\{A\} \mapsto B$ and the realizability interpretation accordingly by

$$|\{A\} \mapsto B|_\rho \quad \triangleq \quad \{t \in \Lambda : \forall V \in |A|_\rho.(t\,V \in |B|_\rho)\}$$

In particular, we have $|\{\mathrm{Nat}(e)\} \mapsto B|_\rho \;=\; \{t \in \Lambda : t\,\overline{n} \in |B|_\rho$, where $n = [\![e]\!]_\rho\}$. It is easy to check that for any formulas A and B, $|\{A\} \mapsto B|_\rho$ is a saturated set, and the adequacy of the $(\forall_E^2)$-rule is thus preserved.

While there is currently no rule to type a term t with a formula of the shape $\{A\} \mapsto B$, we can nonetheless extend the type system with any rule as long as it is adequate (see Proposition 45). We can also extend, maintaining the adequacy of the interpretation of $\{A\} \mapsto B$ (see Proposition 46), the congruence relation with the following rules:

$$\{\exists x.A\} \mapsto B \cong \forall x.\{A\} \mapsto B \qquad\qquad \{\exists X.A\} \mapsto B \cong \forall X.\{A\} \mapsto B$$

We will make use of the following abbreviations:

$$\forall^{\{\mathbb{N}\}}x.A \triangleq \forall x.(\{\mathrm{Nat}(x)\} \mapsto A) \qquad\qquad \exists^{\{\mathbb{N}\}}x.A \triangleq \forall X.(\forall^{\{\mathbb{N}\}}x.(A \to X)) \to X$$

While the first definition is natural, the second one may be a bit more puzzling at first sight. As we saw, the truth value of any formula has to be a saturated set. However, given a formula $A(x)$, the set $\{(\overline{n}, t) : t \in |A(n)|_\rho\}$ is not saturated, and so we cannot define a formula $\exists x.\{\mathrm{Nat}(x)\} \wedge A(x)$ whose realizers would be this set. Nonetheless, the definition of $\exists^{\{\mathbb{N}\}}x.A$ is somehow doing the trick in continuation-passing style, in the sense that we have:

▶ **Proposition 10.** *For any formula A, any valuation ρ and any term t, if $t \in |\exists^{\{\mathbb{N}\}}x.A|_\rho$ then there exists a natural number $n \in \mathbb{N}$ and a term $u \in |A[x := n]|_\rho$ s.t.: $t\,(\lambda xy.(x,y)) \to_\beta (\overline{n}, u)$.*

▶ **Definition 11.** *We define $T \triangleq \lambda zx.(\mathsf{rec}\,(\lambda y.y\,0)\,(\lambda xyz.y\,(\lambda x.z\,(\mathsf{s}\,x)))\,x)\,z$.*

The next proposition relates these new quantifications with the relativized quantifications $\forall^{\mathbb{N}}x.A$ and $\exists^{\mathbb{N}}x.A$ using the term T.

▶ **Proposition 12.** *We have*

1. $T \Vdash \forall^{\{\mathbb{N}\}}x.A \to \forall^{\mathbb{N}}x.A$
2. $\lambda x.x \Vdash \forall^{\mathbb{N}}x.A \to \forall^{\{\mathbb{N}\}}x.A$
3. $\lambda z.z\,\lambda xy.(x,y) \Vdash \exists^{\{\mathbb{N}\}}x.A \to \exists^{\mathbb{N}}x.A$
4. $\lambda xy.T\,y\,\pi_1(x)\,\pi_2(x) \Vdash \exists^{\mathbb{N}}x.A \to \exists^{\{\mathbb{N}\}}x.A$

The term T, which forces the evaluation of an argument of type $\mathrm{Nat}(n)$ to get the underlying value $\overline{n}$ to make it compatible with a function $\forall^{\{\mathbb{N}\}}x.A$, is somehow simulating a call-by-value evaluation (for natural numbers). Such a term is usually called a *storage operator* [20].

While Proposition 12 indicates that the different ways of relativizing the quantifiers are equivalent (in the sense that one admits a realizer if and only if the other does), it is important to keep in mind that this result is peculiar to the current effect-free settings. In particular, this result no longer holds once stateful computations are allowed.

4 Realizability with slices

4.1 Stateful computations

The first step in the Lightstone-Robinson construction aims at getting a product $\mathcal{M}^{\mathbb{N}}$ of the (initial) model $\mathcal{M}$. In order to achieve this goal in our setting, we add a memory cell to our calculus that contains an integer, which we call the *state*. The purpose of the state is to keep track of which "slice" of the product is the interpretation being done. This product allows us to interpret first-order individuals as functions in $\mathbb{N}^{\mathbb{N}}$, so that the interpretation accounts for new elements – the so-called nonstandard elements – for instance the diagonal function (see Proposition 30).

In our extended calculus, the first-order expressions are the same, while second-order formulas now use a value restriction for natural numbers and include a predicate $\mathrm{st}(e)$, as per usual in nonstandard analysis, denoting that the expression e is standard. This means that in our framework we will also have two types of nonstandard quantifications: the usual $\forall^{\mathrm{st}}$, $\exists^{\mathrm{st}}$ and the relativised $\forall^{\{\mathrm{st}\}}$,$\exists^{\{\mathrm{st}\}}x$. We say that a formula is *internal* if it does not contain the predicate $\mathrm{st}(\cdot)$, and *external* otherwise. Terms are extended with two new instructions get and set. The former allows to obtain the content of the current state while the latter allows to increase its content. Formally, we extend the different grammars as follows:

$$\begin{array}{llcl}
\textbf{Formulas} & A, B & ::= & \mathrm{st}(e) \mid X(e_1, \ldots, e_n) \mid \{\mathrm{Nat}(e)\} \mapsto A \mid A \to B \\
 & & \mid & A \wedge B \mid A \vee B \mid \forall x.A \mid \exists x.A \mid \forall X.A \mid \exists X.A \\
\textbf{Terms} & t, u & ::= & \ldots \mid \mathsf{get} \mid \mathsf{set} \\
\textbf{States} & \mathfrak{S} & \triangleq & \mathbb{N}
\end{array}$$

Since the formulas no longer include an unrestricted constructor $\mathrm{Nat}(e)$, the typing rules for 0, s and rec are no longer required[1]. Other than that, the type system is unchanged. In particular, the get and set instructions are not given any typing rule. We will make use of the following abbreviations:

$$\begin{array}{rcl|rcl}
\forall^{\mathrm{st}}x.A & \triangleq & \forall x.(\mathrm{st}(x) \to A) & \exists^{\mathrm{st}}x.A & \triangleq & \exists x.(\mathrm{st}(x) \wedge A) \\
\forall^{\{\mathrm{st}\}}x.A & \triangleq & \forall x.(\mathrm{st}(x) \to (\{\mathrm{Nat}(x)\} \mapsto A)) & \exists^{\{\mathrm{st}\}}x.A & \triangleq & \forall X.((\forall^{\{\mathrm{st}\}}x.(A \to X)) \to X)
\end{array}$$

With the exception of the get / set instructions, the syntax of terms does not account for states. In fact, only the reduction rule for the set instruction allows to change the state. Nonetheless, states play a crucial role in the reduction system. In particular, one-step reductions are now defined for terms together with a state. We write $t \rhd^{\mathfrak{s}}_{\mathfrak{s}'} t'$ to denote that the term t in state $\mathfrak{s}$ reduces to the term t' in state $\mathfrak{s}'$. The one-step reduction over terms is defined by the following rules:

$$\frac{t \rhd_\beta t'}{t \rhd^{\mathfrak{s}}_{\mathfrak{s}} t'} \qquad \frac{}{\mathsf{get} \rhd^{\mathfrak{s}}_{\mathfrak{s}} \mathfrak{s}} \qquad \frac{\mathfrak{s}'' = \max(\mathfrak{s}, \mathfrak{s}')}{\mathsf{set}\, \bar{\mathfrak{s}}\, t \rhd^{\mathfrak{s}'}_{\mathfrak{s}''} t} \qquad \frac{t \rhd^{\mathfrak{s}}_{\mathfrak{s}'} t'}{C[t] \rhd^{\mathfrak{s}}_{\mathfrak{s}'} C[t']}$$

where $C[\,] ::= \mathsf{rec}\ u_0\, u_1\, [\,] \mid [\,]\, u \mid \pi_i([\,]) \mid \mathsf{case}\, [\,]\, \{\iota_1(x_1) \mapsto t_1 | \iota_2(x_2) \mapsto t_2\} \mid \mathsf{s}\, [\,] \mid \mathsf{set}\, [\,]\, u$.

We write $t\, {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}'}\, t'$ for the reflexive-transitive closure of this relation.

Since we now consider effectful computations, we have to fix an evaluation strategy in order to ensure the confluence of the reduction system[2]. Here we follow a call-by-name evaluation strategy (we substitute unevaluated arguments), while for rec and set one of their arguments must be reduced.

[1] In Proposition 29, we show how these terms define realizers for the value restricted natural numbers.

[2] Observe that our definition for $C[\,]$ ensures that our reduction system has no critical pair. We refer the reader unfamiliar with side effects to Example 47, given in the appendices.

4.2 Stateful realizability interpretation

The fact that our syntax now includes states allows us to interpret formulas as terms-with-states[3]. Truth values are then defined as saturated sets in $\mathcal{P}(\Lambda \times \mathfrak{S})$. Individuals are now individuals with states, so elements of $\mathbb{N}^{\mathfrak{S}}$, and similarly predicates of arity k are elements of the set of functions from $\mathbb{N}^k$ to $\mathcal{P}(\Lambda \times \mathfrak{S})$. This creates a mismatch in the sense that predicates are no longer shaped to be applied to individuals[4]. In order to define our interpretation, we need to deal with this mismatch between the structure of individuals and the one of predicates, by defining a suitable notion of application.

▶ **Definition 13.** *Let $F : \mathbb{N}^k \to \mathcal{P}(\Lambda \times \mathfrak{S})$ be a predicate. We define the application of F to individuals $f_1, \ldots, f_k \in \mathbb{N}^{\mathfrak{S}}$ by $F@(f_1, \ldots, f_k) \triangleq \{(t;\mathfrak{s}) : (t;\mathfrak{s}) \in F(f_1(\mathfrak{s}), \ldots, f_k(\mathfrak{s}))\}$.*

▶ **Definition 14.** *An individual $f \in \mathbb{N}^{\mathfrak{S}}$ is said to be* standard *if it is a constant function, i.e. if there exists $n \in \mathbb{N}$ such that $\forall \mathfrak{s} \in \mathfrak{S}.(f(\mathfrak{s}) = n)$. We then write $f = n^*$.*

▶ **Definition 15.** *We define* saturated sets with respect to the stateful reduction *to be sets $S \in \Lambda \times \mathfrak{S}$ s.t. for any terms $t, t' \in \Lambda$ and any states $\mathfrak{s}, \mathfrak{s}' \in \mathfrak{S}$, if $(t';\mathfrak{s}') \in S$ and $t\,{}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}'}\,t'$ then $(t;\mathfrak{s}) \in S$. With abuse of notation we denote the set of these saturated sets by $\mathbf{SAT}$.*

In the realizability interpretation with slices below, truth values are defined as saturated sets. This allows us to reason by *anti-reduction* (sometimes also called *expansion*) in any fixed state. By anti-reduction, we mean that to show that a term t with a state $\mathfrak{s}$ belongs to such a saturated set S, it is enough to find $\mathfrak{s}'$ and t' such that $t\,{}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}'}\,t'$ and $(t';\mathfrak{s}') \in S$.

We now consider valuations which are functions that associate a function in $\mathbb{N}^{\mathfrak{S}}$ to every first-order variable x and a truth value function from $\mathbb{N}^k$ to $\mathbf{SAT}$ to every second-order variable X of arity k. Again, with abuse of notation we denote such valuation by ρ.

We also extend the usual interpretation of first-order expressions to range over $\mathbb{N}^{\mathfrak{S}}$. To that end, we simply define arithmetical functions pointwise on the domain. For instance, if $f \in \mathbb{N}^{\mathfrak{S}}$, we write $S^*(f)$ for the function $\mathfrak{s} \mapsto (S(f(\mathfrak{s})))$. When it is clear from the context, we abuse the notation by writing 0, S, $[\![\cdot]\!]_\rho$, etc. instead of 0^*, S^*, $[\![\cdot]\!]^*_\rho$.

▶ **Definition 16** (Realizability with slices). *The interpretation of a formula A together with a valuation ρ is the set $|A|^{\mathfrak{S}}_\rho$ defined inductively according to the following clauses:*

$$
\begin{array}{rcl}
|\mathrm{st}(e)|^{\mathfrak{S}}_\rho & \triangleq & \begin{cases} \Lambda \times \mathfrak{S} & \text{if } [\![e]\!]_\rho \text{ is standard} \\ \emptyset & \text{otherwise} \end{cases} \\
|X(e_1, \ldots, e_n)|^{\mathfrak{S}}_\rho & \triangleq & \rho(X)@([\![e_1]\!]_\rho, \ldots, [\![e_n]\!]_\rho) \\
|\{\mathrm{Nat}(e)\} \mapsto A|^{\mathfrak{S}}_\rho & \triangleq & \{(t;\mathfrak{s}) \in \Lambda \times \mathfrak{S} : (t\,\overline{n};\mathfrak{s}) \in |A|^{\mathfrak{S}}_\rho, \text{ where } n = [\![e]\!]_\rho(\mathfrak{s})\} \\
|A \to B|^{\mathfrak{S}}_\rho & \triangleq & \{(t;\mathfrak{s}) \in \Lambda \times \mathfrak{S} : \forall u.\big((u;\mathfrak{s}) \in |A|^{\mathfrak{S}}_\rho \Rightarrow (t\,u;\mathfrak{s}) \in |B|^{\mathfrak{S}}_\rho\big)\} \\
|A_1 \wedge A_2|^{\mathfrak{S}}_\rho & \triangleq & \{(t;\mathfrak{s}) \in \Lambda \times \mathfrak{S} : (\pi_1(t);\mathfrak{s}) \in |A_1|^{\mathfrak{S}}_\rho \wedge (\pi_2(t);\mathfrak{s}) \in |A_2|^{\mathfrak{S}}_\rho\big)\}
\end{array}
$$

$$
|A_1 \vee A_2|^{\mathfrak{S}}_\rho \triangleq \{(t;\mathfrak{s}) \in \Lambda \times \mathfrak{S} : \exists i \in \{1,2\}.(\mathsf{case}\ t\,\{\iota_1(x_1) \mapsto x_1 | \iota_2(x_2) \mapsto x_2\};\mathfrak{s}) \in |A_i|^{\mathfrak{S}}_\rho\}
$$

$$
\begin{array}{rclrcl}
|\forall x.A|^{\mathfrak{S}}_\rho & \triangleq & \bigcap_{f \in \mathbb{N}^{\mathfrak{S}}} |A|^{\mathfrak{S}}_{\rho, x \mapsto f} & |\forall X.A|^{\mathfrak{S}}_\rho & \triangleq & \bigcap_{F : \mathbb{N}^k \to \mathbf{SAT}} |A|^{\mathfrak{S}}_{\rho, X \mapsto F} \\
|\exists x.A|^{\mathfrak{S}}_\rho & \triangleq & \bigcup_{f \in \mathbb{N}^{\mathfrak{S}}} |A|^{\mathfrak{S}}_{\rho, x \mapsto f} & |\exists X.A|^{\mathfrak{S}}_\rho & \triangleq & \bigcup_{F : \mathbb{N}^k \to \mathbf{SAT}} |A|^{\mathfrak{S}}_{\rho, X \mapsto F}
\end{array}
$$

We write $(t;s) \Vdash A$ (resp. $t \Vvdash A$) to denote that $(t;s) \in |A|^{\mathfrak{S}}$ (resp. $\forall \mathfrak{s} \in \mathfrak{S}.(t;\mathfrak{s}) \in |A|^{\mathfrak{S}}$). Realizers of the type $t \Vvdash A$ are called universal.

[3] A realizability interpretation with a similar structure, although with a different notion of state, can be found in [28]. The perspective of the latter is also different in that it aims at proving the normalization of a classical call-by-need calculus.

[4] This phenomenon also occurs in the Lightstone-Robinson construction of an ultrapower [23].

Observe that this stateful interpretation has the structure of a product of the interpretation given by Definition 2. The interpretation corresponding to a given state can thus be seen as a *slice* of this product. However, it is important to keep in mind that the set instruction still allows terms to change the value of the state, therefore the slices are not completely independent. We write $|A|^{\mathfrak{s}}_{\rho}$ to denote the truth value $\{(t;\mathfrak{s}) \in |A|^{\mathfrak{S}}_{\rho}\}$ in the slice induced by $\mathfrak{s}$. We first verify that truth values are indeed saturated sets and that the interpretation validates the congruence rules.

▶ **Proposition 17**. *Let A be a formula and ρ a valuation closing A. Then $|A|^{\mathfrak{S}}_{\rho} \in \mathbf{SAT}$.*

▶ **Proposition 18**. *If A and A' are two formulas of HA2 such that $A \cong A'$, then for all valuations ρ closing both A and A' we have $|A|^{\mathfrak{S}}_{\rho} = |A'|^{\mathfrak{S}}_{\rho}$.*

We need to adapt a few definitions to prove the adequacy theorem in this setting.

▶ **Definition 19.** *Given a context Γ, a state $\mathfrak{s}$ and a valuation ρ closing the formulas in Γ, we say that a substitution σ* realizes $\rho(\Gamma)$ *in the state $\mathfrak{s}$ and write $(\sigma;\mathfrak{s}) \Vdash \rho(\Gamma)$ if $\mathrm{dom}(\rho(\Gamma)) \subseteq \mathrm{dom}(\sigma)$ and $(\sigma(x);\mathfrak{s}) \in |A|^{\mathfrak{S}}_{\rho}$, for every declaration $(x : A) \in \Gamma$.*

▶ **Definition 20.** *We say that a typing judgement $\Gamma \vdash t : A$ is* adequate *w.r.t. a state $\mathfrak{s}$ in the stateful system if for any valuation ρ and any substitution $(\sigma;\mathfrak{s}) \Vdash \rho(\Gamma)$ we have $(\sigma(t);\mathfrak{s}) \in |\rho(A)|$. An inference rule is adequate w.r.t. a state $\mathfrak{s}$ if the adequacy (w.r.t. $\mathfrak{s}$) of all its premises implies the adequacy (w.r.t. $\mathfrak{s}$) of its conclusion.*

We are now able to show that, with the exception of the $(\forall^2_E)/(\exists^2_I)$-rules, our rules are adequate. The $(\forall^2_E)/(\exists^2_I)$-rules are shown to be adequate, for internal formulas only, in Proposition 27.

▶ **Theorem 21** (Adequacy). *The typing rules of Figure 1, except the $(\forall^2_E)/(\exists^2_I)$-rules, are adequate.*

▶ Remark 22. Let us explain why the $(\forall^2_E)$-rule is not adequate in general (the same argument applies to the $(\exists^2_I)$-rule). As emphasized at the beginning of this section, we interpret predicates by functions from $\mathbb{N}^k$ to $\mathbf{SAT}$, while the truth values of formulas may vary in the set of functions from $(\mathbb{N}^{\mathfrak{S}})^k$ to $\mathbf{SAT}$. Theorem 26 will make this more precise: internal formulas correspond to functions from $\mathbb{N}^k$ to $\mathbf{SAT}$ while external formulas correspond to functions from $(\mathbb{N}^{\mathfrak{S}})^k$ to $\mathbf{SAT}$. Therefore, in general we cannot substitute a second-order variable by any formula. Indeed, in the second-order elimination rule (for universal quantifiers) variables can only be instantiated by internal formulas. Moreover, if the formula B that we want to substitute is a proposition (i.e. if its arity k is equal to 0), then the substitution is valid since the interpretations of internal and external formulas coincide. This means that we could have chosen to work with impredicative encodings of the conjunction (or other connectives) as in the Russell-Prawitz translation [34]. Indeed, such an encoding relies on the use of propositions, which are thus compatible with the elimination rule:

$$A \wedge B \triangleq \forall X.(A \to B \to X) \to X \qquad A \vee B \triangleq \forall X.(A \to X) \to (B \to X) \to X$$

We show that rec realizes a formula that emulates its former typing rule by using quantifiers relativized with a value restriction.

▶ **Proposition 23**. *We have* rec $\Vdash \forall X.X(0) \to \forall^{\{\mathbb{N}\}}x.(X(x) \to X(S(x))) \to \forall^{\{\mathbb{N}\}}x.X(x)$.

▶ Remark 24. Regarding the necessity of restricting the relativization of quantifiers to values, the proof of Proposition 23 is enlightening. Indeed, given a state $\mathfrak{s}$, two terms $(u_S;\mathfrak{s}) \Vdash \forall^{\{\mathbb{N}\}}y.(X(y) \to X(S(y)))$ and $(u_0;\mathfrak{s}) \Vdash X(0)$ and an individual $f \in \mathbb{N}^{\mathfrak{S}}$ to instantiate x, if instead of a value we were only given a term reducing to a value witnessing $\mathrm{Nat}(f)$, this term may change the value of the state, say to some $\mathfrak{s}'$, before reducing to the value of $f(\mathfrak{s}')$. This would break the proof since nothing is assumed on u_0 and u_S in this new state $\mathfrak{s}'$.

4.3 Glueing

An important property of our interpretation (which also reflects a similar property in the Lightstone-Robinson construction) is that the interpretation of internal formulas can be decomposed as the product of its slices. In other words, internal formulas can only access information in the current state. In particular, and as expected, this means that it is impossible to express standardness by means of internal formulas. To state this formally, we first define the restriction of formulas and truth values with respect to a slice.

▶ **Definition 25.** *Given an internal formula A, we define $\overline{A}^{\mathfrak{s}}$ as the formula whose individuals are all applied in $\mathfrak{s}$. Formally, it amounts to replacing each individual by the standard individual with which it coincides in the state $\mathfrak{s}$:*

$$\begin{array}{rcl|rcl|rcl}
\overline{F(e_1,...,e_k)}^{\mathfrak{s}} & \triangleq & F((e_1(\mathfrak{s}))^*,\ldots,(e_k(\mathfrak{s}))^*) & \overline{A\wedge B}^{\mathfrak{s}} & \triangleq & \overline{A}^{\mathfrak{s}}\wedge\overline{B}^{\mathfrak{s}} & \overline{\exists x.A}^{\mathfrak{s}} & \triangleq & \exists x.\overline{A}^{\mathfrak{s}} \\
\overline{A\to B}^{\mathfrak{s}} & \triangleq & \overline{A}^{\mathfrak{s}}\to\overline{B}^{\mathfrak{s}} & \overline{A\vee B}^{\mathfrak{s}} & \triangleq & \overline{A}^{\mathfrak{s}}\vee\overline{B}^{\mathfrak{s}} & \overline{\forall X.A}^{\mathfrak{s}} & \triangleq & \forall X.\overline{A}^{\mathfrak{s}} \\
\overline{\{\mathrm{Nat}(e)\}\mapsto B}^{\mathfrak{s}} & \triangleq & \{\mathrm{Nat}((e(\mathfrak{s}))^*)\}\mapsto\overline{B}^{\mathfrak{s}} & \overline{\forall x.A}^{\mathfrak{s}} & \triangleq & \forall x.\overline{A}^{\mathfrak{s}} & \overline{\exists X.A}^{\mathfrak{s}} & \triangleq & \exists X.\overline{A}^{\mathfrak{s}}
\end{array}$$

The next result ensures that truth values of internal formulas can be split into slices.

▶ **Theorem 26** (Glueing). *For any internal formula A and valuation ρ closing A, we have that $(t;\mathfrak{s}) \in |A|_\rho^{\mathfrak{S}} \Leftrightarrow t \in |\overline{A}^{\mathfrak{s}}|_\rho^{\mathfrak{s}}$.*

Let $B(x)$ be a formula whose only free variable is x, and ρ a valuation. In general, the function $\mathcal{F}_B$ that associates to any individual f the truth value $|B(f)|_\rho^{\mathfrak{S}}$ is a function from $\mathbb{N}^{\mathfrak{S}}$ to **SAT**. If B is internal, by the glueing theorem, to determine $\mathcal{F}_B$ it is enough to know its value for standard individuals. This means that we only need to know a function from $\mathbb{N}$ to **SAT**. As such, we can now formally state the intuition developed in Remark 22.

▶ **Proposition 27.** *The elimination rule $(\forall_E^2)$ for the 2nd-order universal quantification and the introduction rule $(\exists_I^2)$ for the 2nd-order existential quantification are adequate for any internal formula B whose only free variables are $(x_1,...,x_k)$.*

▶ Remark 28. Observe that external formulas such as $\mathrm{st}(x)\to\bot$ cannot be defined by glueing. Consider for instance a nonstandard element τ. Then $|\mathrm{st}(\tau)\to\bot|^{\mathfrak{S}} = \Lambda\times\mathfrak{S}$, while for any state $\mathfrak{s}\in\mathfrak{S}$ we have $|\overline{\mathrm{st}(\tau)\to\bot}^{\mathfrak{s}}|_{\mathfrak{s}} = |\mathrm{st}(\tau(\mathfrak{s})^*)\to\bot|_{\mathfrak{s}} = |\top\to\bot|_{\mathfrak{s}} = \emptyset$.

It is well-known that the comprehension scheme $\mathrm{CA}_B \triangleq \exists X.\forall x.(X(x)\Leftrightarrow B)$ is a logical consequence of the elimination principle $\mathrm{Elim}_A^B \triangleq (\forall X.A) \Rightarrow A[X(x) := B]$ (by taking $A = \exists Y.\forall x.(Y(x)\Leftrightarrow X(x))$). Since we have the $(\forall_E^2)$-rule restricted to internal formulas B, the comprehension scheme is also valid for these formulas. In particular, this implies standardization for internal formulas, i.e. $\forall^{\mathrm{st}}X.\exists^{\mathrm{st}}Y.\forall^{\mathrm{st}}z.(Y(z)\Leftrightarrow X(z)\wedge B(z))$ holds for B an internal formula. The usual standardization scheme, formulated for all formulas, requires further investigation and is left for future work. Of course, the comprehension scheme does not hold for external formulas, so the relativization on the quantifiers in standardization is in this sense necessary.

5 Nonstandard principles in realizability with slices

5.1 Natural numbers

Observe that the language of HA2 does not express the existence of specific nonstandard elements, e.g. δ is not in the language. However, to refer to some nonstandard element τ, we can always consider a valuation that maps a variable x to τ. With abuse of notation, in the remainder of this paper, we will write nonstandard elements directly in formulas as if they were in the language. Also, we will use the notation † to refer to an arbitrary λ-term with no further assumption.

In the stateful interpretation (Definition 16), we considered a value restriction to natural numbers. Nonetheless, we can assert that an expression is a natural number through the formula $\mathrm{Nat}'(e) \quad \triangleq \quad \forall X.(\{\mathrm{Nat}(e)\} \mapsto X) \to X$. It is easy to see, by an argument similar to Proposition 10, that for any individual $f \in \mathbb{N}^{\mathfrak{S}}$, if t is a term such that $(t; \mathfrak{s}) \in |\mathrm{Nat}'(f)|^{\mathfrak{S}}$, then $t\,\lambda x.x\ {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\ \overline{n}$ where $n = f(\mathfrak{s})$. In other words, t is an effect-free term producing $\overline{n}$. This is to be compared with $\mathrm{Nat}(f)$, for which the requirement for its truth value to be saturated, would have entailed its interpretation to reduce to a natural number $f(\mathfrak{s}')$ in a possibly different state. We show that (by-value) natural numbers, i.e. Nat', contain 0, and are closed under the successor and recursion for internal formulas.

▶ **Proposition 29**. *Let A be an internal formula. We have*

1. $\lambda x.x\,0 \Vvdash \mathrm{Nat}'(0)$
2. $\lambda xy.y\,(\mathsf{s}\,x) \Vvdash \forall^{\{\mathbb{N}\}}x.\mathrm{Nat}'(S(x))$
3. $\mathsf{rec} \Vvdash A(0) \to \big(\forall^{\{\mathbb{N}\}}y.(A(y) \to A(S\,y))\big) \to \forall^{\{\mathbb{N}\}}x.A(x))$

The interpretation now witnesses the existence of new elements. The canonical example is the *diagonal*, i.e. the function $\delta : n \mapsto n$. Indeed, the diagonal is a nonstandard natural number which is realized by the get instruction.

▶ **Proposition 30**. *We have that*

1. $\dagger \Vvdash \neg\mathrm{st}(\delta)$
2. $\dagger \Vvdash \exists x.\neg\mathrm{st}(x)$
3. $\lambda x.T\,x\ \mathsf{get} \Vvdash \mathrm{Nat}'(\delta)$
4. $\lambda x.T\,x\ \mathsf{get}\ \dagger \Vvdash \exists^{\{\mathbb{N}\}}x.\neg\mathrm{st}(x)$

Part 2 in Proposition 30 is sometimes referred to as the ENS_0 (existence of nonstandard elements) principle [6]. As a consequence of Proposition 27, Leibniz equality is only compatible with the $(\forall^2_E)$-rule restricted to internal formulas. In our setting, this encoding only reflects equality in the current state, i.e. a local knowledge of individuals (slice by slice), while the usual notion of equality (for $\mathbb{N}^{\mathfrak{S}}$) requires a global knowledge (on all the slices). If $A(x)$ is an external formula, we cannot hope to have an internal definition of equality such that its elimination principle $x = y \to A(x) \to A(y)$ is valid.

▶ **Example 31.** Consider an individual f, equal to 1 everywhere except for some state $\mathfrak{s}_0$ where it is equal to 0. Then by considering the formula $A(x) \triangleq (\mathrm{st}(x) \to \bot) \to \bot$, it is easy to get a realizer of $\bot$ out of any realizer of $(\forall Z.(Z(1^*) \to Z(f))) \to A(1^*) \to A(f)$.

Nonetheless, the elimination of Leibniz equality is realizable for standard individuals or for internal formulas.

▶ **Proposition 32**. *Let f and g be individuals in $\mathbb{N}^{\mathfrak{S}}$, then*

1. *For any formula $A(x)$, $\lambda x.x \Vvdash \mathrm{st}(f) \to \mathrm{st}(g) \to (\forall Z.(Z(f) \to Z(g))) \to A(f) \to A(g)$*
2. *If $A(x)$ is an internal formula, then $\lambda x.x \Vvdash (\forall Z.(Z(f) \to Z(g))) \to A(f) \to A(g)$*

5.2 Nonstandard reasoning principles

In this section, we prove some properties which are usual in frameworks that use nonstandard analysis: transfer, overspill, external induction, idealization, etc.

Theorem 33 below indicates that the transfer property (for internal formulas) is devoid of computational content. This is a somewhat reassuring fact: properties that are true for standard individuals are automatically true for all individuals.

▶ **Theorem 33** (Transfer). *For any internal formula A we have:*

1. $\bigcap_{f\in\mathbb{N}^{\mathfrak{S}}} |A|^{\mathfrak{S}}_{x\mapsto f} = \bigcap_{n\in\mathbb{N}} |A|^{\mathfrak{S}}_{x\mapsto n^*}$
2. $\lambda xy.x \Vdash \forall x.A(x) \to \forall^{\mathrm{st}} x.A(x)$
3. $\lambda x.x\,\dagger \Vdash \forall^{\mathrm{st}} x.A(x) \to \forall x.A(x)$
4. $\bigcup_{f\in\mathbb{N}^{\mathfrak{S}}} |A|^{\mathfrak{S}}_{x\mapsto f} = \bigcup_{n\in\mathbb{N}} |A|^{\mathfrak{S}}_{x\mapsto n^*}$
5. $\lambda x.(\dagger, x) \Vdash \exists x.A(x) \to \exists^{\mathrm{st}} x.A(x)$
6. $\lambda x.\pi_2(x) \Vdash \exists^{\mathrm{st}} x.A(x) \to \exists x.A(x)$

As expected, transfer does not hold for all formulas. A counter-example is given in the next proposition by the external formula stating that all individuals are (not not) standard.

▶ **Proposition 34.** *Let $A(x)$ be the formula $\neg\mathrm{st}(x)$. The formulas $\forall^{\mathrm{st}} x.\neg A(x) \to \forall x.\neg A(x)$ and $\exists x.A(x) \to \exists^{\mathrm{st}} x.A(x)$ have no realizer.*

The principle of external induction [32] allows to prove that a certain property is valid for all standard natural numbers, for instance, that every nonstandard element is larger than all standard natural numbers[5]. We show that in our context, this principle can be realized using the rec instruction.

▶ **Proposition 35** (External induction). *For any formula $A(x)$ whose only free variable is x*

$$\mathsf{rec} \Vdash A(0^*) \to \forall^{\{\mathrm{st}\}} x.(A(x) \to A(S(x))) \to \forall^{\{\mathrm{st}\}} x.A(x).$$

The next two propositions, show that one cannot separate standard natural numbers from nonstandard natural numbers using an internal formula [36]. We first show that overspill can be *realized* by combining the realizers for ENS_0 and for the transfer principle.

▶ **Proposition 36** (Overspill). *For any internal formula A, we have*

$$\lambda x.(x\,\dagger, \dagger) \Vdash \forall^{\mathrm{st}} x.A(x) \to \exists x.(\neg\mathrm{st}(x) \wedge A(x)).$$

The usual proof of underspill is by contradiction, hence using classical logic, which we do not have here. Nevertheless, we can obtain the following version in which a double-negation occurs.

▶ **Proposition 37** (Underspill). *For any internal formula A, we have*

$$\lambda xy.(\lambda z.y\,(\dagger, z))(x\,\dagger) \Vdash (\forall x.\neg\mathrm{st}(x) \to A(x)) \to \neg\neg\exists^{\mathrm{st}} x.A(x).$$

[5] Actually, this requires to consider a quotiented definition of the standardness predicate, see Proposition 42.

5.3 Idealization

We first extend the realizability interpretation to take into account relations $R : \mathbb{N}^2 \to \mathbb{N}$ on the natural numbers:

$$|R(e_1, e_2)|^{\mathfrak{S}}_{\rho} \quad \triangleq \quad \{(t; \mathfrak{s}) : R([\![e_1]\!]_{\rho}(\mathfrak{s}), [\![e_2]\!]_{\rho}(\mathfrak{s})) \text{ holds}\}$$

This coincides with the interpretation of the relation R through a second-order variable and the corresponding semantic relation from $\mathbb{N}^2$ to **SAT** in the interpretation.

Let us now briefly illustrate the main idea behind the proof of idealization by showing that there exists a (nonstandard) natural number greater than or equal to any standard number. The usual proof relies on the fact that δ is such a number, since for any standard number n, in any slice greater than or equal to n, the relation $n \leq \delta$ holds. In our setting, we use the set instruction to reach such a state.

▶ **Proposition 38** (Diagonalization). *We have* $\lambda z.T\,z\,\mathsf{get}\,(\lambda xy.\,\mathsf{set}\,y\,\dagger) \Vvdash \exists^{\{\mathbb{N}\}}x.\forall^{\{\mathrm{st}\}}y.y \leq x$.

▶ Remark 39. Consider a term loop^+ such that for any state $\mathfrak{s} \in \mathfrak{S}$, $\mathsf{loop}^+\ {}^{\mathfrak{s}}\!\!\downarrow^{\mathfrak{s}}\ \mathsf{incr}\ \mathsf{loop}^+$ where $\mathsf{incr} \triangleq \lambda x.\,\mathsf{set}\,(\mathsf{s}\ \mathsf{get})\,x$. Observe that $\mathsf{loop}^+ \Vvdash \forall^{\mathrm{st}}x.x < \delta$ where the quantifier does not need to be relativized since the value of x is not required. Yet, the computation never terminates and we do not even know when the computation reaches a correct state.

As mentioned above, the idea to prove the general case of idealization is very similar. If for any $n \in \mathbb{N}$ there exists $\tau_n \in \mathbb{N}$ such that for any $m \leq n$, $R(\tau_n, m)$ holds, we can consider the nonstandard natural number $\tau \triangleq (\tau_{\mathfrak{s}})_{\mathfrak{s} \in \mathfrak{S}} \in \mathbb{N}^{\mathfrak{S}}$. As shown by the following lemma, we can compute τ from any realizer of $\forall^{\{\mathrm{st}\}}n.\exists^{\{\mathrm{st}\}}x.\forall^{\{\mathrm{st}\}}y.(y \leq n \to R(x, y))$.

▶ **Lemma 40.** *For any formula A, any valuation ρ, any state $\mathfrak{s}$ and any term t such that $(t; \mathfrak{s}) \in |\exists^{\{\mathrm{st}\}}x.A|^{\mathfrak{S}}_{\rho}$, there exists a natural number $n \in \mathbb{N}$ and a term u such that $(u; \mathfrak{s}) \in |A|^{\mathfrak{S}}_{\rho, x \mapsto n}$ and $t\,(\lambda xyz.(y, z))\ {}^{\mathfrak{s}}\!\!\downarrow^{\mathfrak{s}}\ (\overline{n}, u)$.*

The term $\mathsf{ideal} \triangleq \lambda x.\lambda y.T\,y\,(\pi_1(T\,(x\,\dagger)\,\mathsf{get}\,(\lambda wyz.(y, z))))\,(\lambda yz.\,\mathsf{set}\,z\,y)$ is a realizer for the idealization principle. Indeed, in any state $\mathfrak{s}$ the first component of ideal computes $\tau(\mathfrak{s})$, using Lemma 40, while the second component increases the state to ensure the validity of the relation (as in Proposition 38).

▶ **Theorem 41** (Idealization). $\mathsf{ideal} \Vvdash \forall^{\{\mathrm{st}\}}n.\exists^{\{\mathrm{st}\}}x.\forall^{\{\mathrm{st}\}}y.(y \leq n \to R(x, y)) \to \exists^{\{\mathbb{N}\}}x.\forall^{\{\mathrm{st}\}}y.R(x, y)$

6 Conclusion and future work

6.1 Towards a quotient

In order to fully mimic Lightstone and Robinson's construction, an extra step would be required where one would take a quotient of the interpretation with slices. The study of such an interpretation is outside the scope of this paper[6]. Let us nevertheless comment on a possibility. Fix a free ultrafilter $\mathcal{U}$ over the set of states. Given any set V, let us denote by $\cong$ the equivalence relation over $V^{\mathfrak{S}}$ defined by $f \cong g \triangleq \{\mathfrak{s} \in \mathfrak{S} : f(\mathfrak{s}) = g(\mathfrak{s})\} \in \mathcal{U}$.

First, we can, within the realizability with slices, change the way $\mathrm{st}(f)$ is interpreted to consider standardness up to the ultrafilter. In this way, $f \in \mathbb{N}^{\mathfrak{S}}$ is said to be standard if and only if there exists $n \in \mathbb{N}$ s.t. $f \cong n^*$. As a consequence, we for instance get that:

[6] We leave it for future work, but more details sketching this construction are given in Appendix C.

▶ **Proposition 42.** $\lambda xy.\mathsf{loop}^+ \Vdash \forall x, y.\neg\mathrm{st}(x) \to \mathrm{st}(y) \to y < x$

We then need to define a new notion of realizability in which realizers are also considered up to the equivalence relations induced by $\mathcal{U}$. To that end, a natural attempt consists in considering Łoś' theorem as a guideline. For the sake of clarity, let us denote by $|A|^*$ the truth values in this interpretation, which we shall call *realizability up to* $\mathcal{U}$.

▶ **Definition 43.** *We say that a formula A is* Łoś-reducible *if for any valuation ρ closing A, $t \in |A|^*$ if and only if $\{\mathfrak{s} \in \mathfrak{S} : (t;\mathfrak{s}) \in |A|^{\mathfrak{S}}_{\rho}\} \in \mathcal{U}$.*

We actually define the interpretation of connectives by this equivalence (*e.g.*, we define $|A \to B|^*_\rho \triangleq \{t \in \Lambda : \{\mathfrak{s} \in \mathfrak{S} : (t;\mathfrak{s}) \in |A \to B|^{\mathfrak{S}}_{\rho}\} \in \mathcal{U}, \})$ while the interpretation of the quantifiers is still defined via intersections (resp. unions) over the same domain as in the interpretation with slices (*e.g.*, $|\forall x.A|^*_\rho \triangleq \bigcap_{f\in\mathbb{N}^{\mathfrak{S}}} |A|^*_{\rho,x\mapsto f}$). As shown in the following theorem, first-order quantifiers behave well with respect to the ultrafilter.

▶ **Theorem 44** (Łoś' theorem). *First-order internal formulas as well as arbitrary disjunctions, conjunctions and implications are Łoś-reducible.*

Theorem 44 implies that if a term t is a realizer of a first-order internal formula A "often enough" in the interpretation with slices, then t is still a realizer in the interpretation up to $\mathcal{U}$. Since all the realizers in Section 5 were universal, they are still realizers in this new setting, meaning that all the results from that section remain valid in the interpretation up to $\mathcal{U}$. In particular, Theorem 44 applies to transfer, idealization, overspill or underspill.

A simple example illustrating this new interpretation is the formula $\forall^{\mathrm{st}}x.x < \delta$, which was realized by loop^+ in the interpretation with slices and is now realized by any term (because for any $n \in \mathbb{N}$, the set of states such that $n < \delta$ is equal to $[n; +\infty[$ which belongs to $\mathcal{U}$). Similarly, loop^+ can be replaced by † in Proposition 42.

However, this construction is still prospective and it raises several questions. On the one hand, such a definition is not as compositional as one usually expects in realizability. Indeed, while we have that $|A \to B|^*_\rho \subseteq \{t : \forall u \in |A|^*_\rho.t\,u \in |B|^*_\rho\}$ for any internal formulas A and B and any valuation ρ, this inclusion is strict in general (see Remark 52) . In other words, we can compose a realizer $t \in |A \to B|^*_\rho$ with a realizer in $u \in |A|^*_\rho$ to get $t\,u \in |B|^*_\rho$, but the $(\to_I)$-rule is not adequate when considering substitutions of variables by realizers in the quotiented truth values. More generally, such a definition does not exactly match the intuition of the quotient in the Lightstone-Robinson construction, just like the interpretation with slices does not exactly define a product due to the ability to change the state via set.

On the other hand, the interpretation up to $\mathcal{U}$ is indeed a new and more flexible interpretation in that it allows us to get realizers for principles that were inaccessible in the interpretation with slices (e.g., $\forall x, y.\neg\mathrm{st}(x) \to \mathrm{st}(y) \to y < x$). We would like to determine whether it allows us to realize other, more involved, nonstandard reasoning principles such as standardization but *prima facie* this principle does not seem to be realizable with the current definitions.

6.2 Related and future work

Some related works concern notions of realizability for nonstandard arithmetic which are variants of Kreisel's modified realizability [6, 9]. These notions of realizability are more inspired by Nelson's syntactical approach to nonstandard analysis. In particular, they rely on translations of formulas inducing conservative extensions of Heyting arithmetic. An important difference with our work is that we are able to give non-trivial computational content to idealization. It could be interesting to better understand the relation between this approach and the approaches based on Kreisel's realizability. In particular, we would like to know whether we can obtain a preservation result for some class of formulas (*e.g.* internal, quantifier-free, ∃-free formulas).

It seems that our interpretation with slices can be adapted without difficulty to Krivine's classical realizability. In particular, a similar interpretation (but with a very different purpose) for a classical calculus with a global environment is given in [28]. This setting could possibly allow to validate new principles by taking advantage of the computational power brought by control operators.

Finally, similar ideas have been adressed by Aschieri. In [1] the author uses a notion of state which allows to construct a forcing model. In particular, natural numbers are interpreted as functions from states to ℕ. Yet, his work does not pay attention to the nonstandard principles that can be obtained in his setting but rather to forcing. It would be natural to investigate whether our setting also allows for forcing techniques. This connection with forcing is reinforced by the fact that in the realm of Krivine's realizability, which generalizes Cohen's forcing, the latter is given a computational content via the addition of a monotone memory cell to the abstract machine in order to store forcing conditions [21, 26].

References

1 Federico Aschieri. Constructive forcing, CPS translations and witness extraction in interactive realizability. *Mathematical Structures in Computer Science*, 27(6):993–1031, 2017. doi:10.1017/S0960129515000468.

2 Jeremy Avigad. Weak theories of nonstandard arithmetic and analysis. In *Reverse mathematics 2001*, volume 21 of *Lect. Notes Log.*, pages 19–46. Assoc. Symbol. Logic, La Jolla, CA, 2005.

3 Jacques Bair, Piotr Błaszczyk, Elías Guillén, Peter Heinig, Vladimir Kanovei, and Mikhail G. Katz. Continuity between Cauchy and Bolzano: issues of antecedents and priority. *British Journal for the History of Mathematics*, pages 1–18, 2020. doi:10.1080/26375451.2020.1770015.

4 Jacques Bair, Piotr Błaszczyk, Robert Ely, Peter Heinig, and Mikhail Katz. Leibniz's well-founded fictions and their interpetations. *Mat. Stud.*, 49(2):186–224, 2018.

5 Henk Barendregt. Lambda calculi with types. In S. Abramsky, Dov M. Gabbay, and S. E. Maibaum, editors, *Handbook of Logic in Computer Science (Vol. 2)*, pages 117–309. Oxford University Press, Inc., New York, NY, USA, 1992.

6 Benno van den Berg, Eyvind Briseid, and Pavol Safarik. A functional interpretation for nonstandard arithmetic. *Ann. Pure Appl. Logic*, 163(12):1962–1994, 2012.

7 Jean-Louis Callot. Trois leçons d'analyse infinitésimale. In J.M. Salanskis and H. Sinaceur, editors, *Le labyrinthe du continu*, pages 369–381. Springer-Verlag, Paris, 1992.

8 Bruno Dinis and Fernando Ferreira. Interpreting weak Kőnig's lemma in theories of nonstandard arithmetic. *Mathematical Logic Quarterly*, 63(1-2):114–123, 2017. doi:10.1002/malq.201600066.

9 Bruno Dinis and Jaime Gaspar. Intuitionistic nonstandard bounded modified realisability and functional interpretation. *Ann. Pure Appl. Logic*, 169(5):392–412, 2018. doi:10.1016/j.apal.2017.12.004.

10 Bruno Dinis and Imme van den Berg. *Neutrices and external numbers: A flexible number system.* Monographs and Research Notes in Mathematics. CRC Press, Boca Raton, FL, 2019. With a foreword by Claude Lobry. doi:10.1201/9780429291456.

11 Kurt Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12(3-4):280–287, 1958. doi:10.1111/j.1746-8361.1958.tb01464.x.

12 Timothy Griffin. A formulae-as-type notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, pages 47–58, New York, NY, USA, 1990. ACM. doi:10.1145/96709.96714.

13 Amar Hadzihasanovic and Benno van den Berg. Nonstandard functional interpretations and categorical models. *ND Journal of Formal Logic*, 58(3), 2017.

14 Arend Heyting. *Mathematische Grundlagenforschung. Intuitionismus. Beweistheorie.* Springer-Verlag, Berlin, 1934. doi:10.1007/978-3-642-65617-0.

15 Mikhail Katz and David Sherry. Leibniz's infinitesimals: their fictionality, their modern implementations, and their foes from Berkeley to Russell and beyond. *Erkenntnis*, 78(3):571–625, 2013. doi:10.1007/s10670-012-9370-y.

16 Stephen Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109–124, 1945.

17 Andrey Kolmogorov. Zur Deutung der intuitionistischen Logik. *Mathematische Zeitschrift*, 35(1):58–65, 1932. doi:10.1007/BF01186549.

18 Georg Kreisel. On the interpretation of non-finitist proofs, I. *J. Symb. Log.*, 16:241–267, 1951.

19 Jean-Louis Krivine. Typed lambda-calculus in classical Zermelo-Fraenkel set theory. *Arch. Math. Log.*, 40(3):189–205, 2001.

20 Jean-Louis Krivine. Realizability in classical logic. In Interactive models of computation and program behaviour. *Panoramas et synthèses*, 27, 2009.

21 Jean-Louis Krivine. Realizability algebras: a program to well order $\mathbb{R}$. *Logical Methods in Computer Science*, 7(3), 2011. doi:10.2168/LMCS-7(3:2)2011.

22 Jean-Louis Krivine. Bar Recursion in Classical Realisability: Dependent Choice and Continuum Hypothesis. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:11, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2016.25.

23 Albert Lightstone and Abraham Robinson. *Nonarchimedean Fields and Asymptotic Expansions.* North-Holland mathematical library. North-Holland, 1975.

24 Robert Lutz. Rêveries infinitésimales. *Gazette des mathématiciens*, 34:79–87, 1987.

25 Alexandre Miquel. Existential witness extraction in classical realizability and via a negative translation. *Logical Methods in Computer Science*, 7(2):188–202, 2011. doi:10.2168/LMCS-7(2:2)2011.

26 Alexandre Miquel. Forcing as a program transformation. In *Proceedings of the 2011 IEEE 26th Annual Symposium on Logic in Computer Science*, LICS '11, page 197–206, USA, 2011. IEEE Computer Society. doi:10.1109/LICS.2011.47.

27 Alexandre Miquel. Implicative algebras: A new foundation for realizability and forcing. *ArXiv e-prints*, 2020. arXiv:1802.00528.

28 Étienne Miquey and Hugo Herbelin. Realizability interpretation and normalization of typed call-by-need λ-calculus with control. In *Foundations of software science and computation structures*, volume 10803 of *Lecture Notes in Comput. Sci.*, pages 276–292. Springer, Cham, 2018. doi:10.1007/978-3-319-89366-2_1.

29 Ieke Moerdijk. A model for intuitionistic non-standard arithmetic. *Ann. Pure Appl. Logic*, 73(1):37–51, 1995. A tribute to Dirk van Dalen. doi:10.1016/0168-0072(93)E0071-U.

30 Ieke Moerdijk and Erik Palmgren. Minimal models of Heyting arithmetic. *J. Symbolic Logic*, 62(4):1448–1460, 1997. doi:10.2307/2275651.

31 Edward Nelson. Internal set theory: A new approach to nonstandard analysis. *Bull. Amer. Math. Soc*, 1977.

32 Edward Nelson. *Radically Elementary Probability Theory.* Annals of Mathematical Studies, vol. 117. Princeton University Press, Princeton, N. J., 1987.

33 Jerzy Łoś. Quelques remarques, théorèmes et problèmes sur les classes définissables d'algèbres. *Journal of Symbolic Logic*, 25(2):168–168, 1960. doi:10.2307/2964232.

34 Dag Prawitz. *Natural deduction. A proof-theoretical study.* Acta Universitatis Stockholmiensis. Stockholm Studies in Philosophy, No. 3. Almqvist & Wiksell, Stockholm, 1965.

35 Abraham Robinson. Non-standard analysis. *Proc. Roy. Acad. Sci.*, 1961.

36 Abraham Robinson. *Non-standard analysis.* North-Holland Publishing Co., Amsterdam, 1966.

37 Alfred Tarski. Une contribution à la théorie de la mesure. *Fundamenta Mathematicae*, 15(1):42–50, 1930. URL: `http://eudml.org/doc/212372`.

38 Jaap van Oosten. *Realizability: an introduction to its categorical side*, volume 152 of *Studies in Logic and the Foundations of Mathematics.* Elsevier B. V., Amsterdam, 2008.

A Proofs of Section 3

A.1 Realizability interpretation

Proof of Proposition 3. By induction on $A \cong A'$. The interesting case is for proving the equality $|(\exists x.A) \to B|^{\mathfrak{S}}_\rho = |\forall x.(A \to B)|^{\mathfrak{S}}_\rho$. ◀

Proof of Theorem 7 (Adequacy). The proof is standard, by case analysis. ◀

Proof of Corollary 9 (Consistency). If $\vdash t : \bot$, then by Theorem 7 one has $t \in |\bot| = |\forall X.X| = \bigcap_{S \in \mathbf{SAT}} S = \emptyset$. To see that this intersection is indeed empty, one may take for example $S_0 = \{t \in \Lambda : t \to 0\} \in \mathbf{SAT}$ and $S_1 = \{t \in \Lambda : t \to \mathsf{s}0\} \in \mathbf{SAT}$, and clearly $S_0 \cap S_1 = \emptyset$. ◀

A.2 Introducing value restrictions

▶ **Proposition 45.** *The following typing rules are adequate:*

$$\frac{\Gamma \vdash t : A \to B}{\Gamma \vdash t : \{A\} \mapsto B}\ (\mapsto_I) \qquad\qquad \frac{\Gamma \vdash t : \{A\} \mapsto B \quad \Gamma \vdash V : A}{\Gamma \vdash t\,V : B}\ (\mapsto_E)$$

Proof. For the first rule it suffices to see that for any valuation ρ, we have:

$$\{t \in \Lambda : \forall u \in |A|^{\mathfrak{S}}_\rho.(t\,u \in |B|^{\mathfrak{S}}_\rho)\} \subseteq \{t \in \Lambda : \forall V \in |A|^{\mathfrak{S}}_\rho.(t\,V \in |B|^{\mathfrak{S}}_\rho)\}$$

For the second one, the proof is analogous to the adequacy of the $(\to_E)$-rule. ◀

▶ **Proposition 46.** *For any formulas A and B, we have*

1. $|\{\exists x.A\} \mapsto B|^{\mathfrak{S}}_\rho = |\forall x.\{A\} \mapsto B|^{\mathfrak{S}}_\rho$
2. $|\{\exists X.A\} \mapsto B|^{\mathfrak{S}}_\rho = |\forall X.\{A\} \mapsto B|^{\mathfrak{S}}_\rho$

Proof. The proof is analogous to the proof of Proposition 3. ◀

Proof of Proposition 10. Let t be a term in $|\exists^{\{\mathbb{N}\}}x.A|^{\mathfrak{S}}_\rho$. For any $\mathbb{X} \in \mathbf{SAT}$ and any $v \in |\forall^{\{\mathbb{N}\}}x.(A \to X)|_{\rho, X \mapsto \mathbb{X}}$, we have that $t\,v \in \mathbb{X}$. Let us define the set $\mathbb{X} = \{w \in \Lambda : \exists n, u.w \to_\beta (\overline{n}, u) \wedge u \in |A[x := n]|^{\mathfrak{S}}_\rho\}$, which is obviously saturated. It is clear that $\lambda xy.(x, y) \in |\forall^{\{\mathbb{N}\}}x.(A \to X)|_{\rho, X \mapsto \mathbb{X}}$ since for any $n \in \mathbb{N}$ and any $u \in |A[x := n]|^{\mathfrak{S}}_\rho\}$, it holds that $(\lambda xy.(x, y))\,\overline{n}\,u \to_\beta (\overline{n}, u) \in \mathbb{X}$. We conclude that $t\,(\lambda xy.(x, y)) \to_\beta (\overline{n}, u)$. ◀

Proof of Proposition 12. Easy realizability proofs by anti-reduction. ◀

B Proofs of Section 4

B.1 Stateful computations

We illustrate the need for an evaluation strategy to ensure confluence in the presence of states by giving a simple example of stateful computation whose result is not the same using call-by-name and call-by-value strategies.

▶ **Example 47.** Let us write $x + y$ for a term that computes the addition of x and y (such term is easily definable via rec). Let us define $\mathsf{incr}_0 \triangleq \mathsf{set}\,(\mathsf{s}\;\mathsf{get})\,0$ (which increases the state and reduces to 0) and $t \triangleq (\lambda x.(\mathsf{get} + x) + x)\;\mathsf{incr}_0$. If we reduce the argument of the functions first (call-by-value) we obtain $t\;{}^0{\downarrow}^1\;(\lambda x.(\mathsf{get} + x) + x))\,0\;{}^1{\downarrow}^1\;(\mathsf{get} + 0) + 0\;{}^1{\downarrow}^1\;1$. In turn, if we perform the β-reduction without reducing the argument (call-by-name), we get $t\;{}^0{\downarrow}^0\;(\mathsf{get} + \mathsf{incr}_0) + \mathsf{incr}_0\;{}^0{\downarrow}^1\;(\mathsf{get} + \mathsf{incr}_0) + 0\;{}^1{\downarrow}^2\;\mathsf{get} + 0\;{}^2{\downarrow}^2\;2$. In the absence of an evaluation strategy, the system would thus have admitted unsolvable critical pairs.

B.2 Realizability interpretation

Proof of Proposition 17. By a straighforward induction on the structure of A. Observe for instance that the case $\mathrm{st}(f)$ follows from the definition and that the case $X(e_1, \ldots, e_n)$ follows from the fact that, by definition, $\rho(X)$ takes values in $\mathbf{SAT}$. ◀

Proof of Proposition 18. The proof, by induction on $A \cong A'$, is similar to the proof of Proposition 3. ◀

Proof of Theorem 21 (Adequacy). The proof, by case analysis, is essentially the same as the usual adequacy proof for HA2, since none of the instructions involved in the typing rules allows to change the value of the state. Let us prove the case $(\rightarrow_I)$. Writing Γ for the typing context, ρ for a valuation closing all the considered formulas, $\mathfrak{s}$ for the considered state and σ for a substitution such that $(\sigma;\mathfrak{s}) \Vdash \rho(\Gamma)$, by assumption, for any substitution σ' such that $(\sigma';\mathfrak{s}) \Vdash \rho(\Gamma), x : \rho(A)$, we have that $(\sigma(t);\mathfrak{s}) \in |B|^{\mathfrak{S}}_{\rho}$. We have to prove that $(\lambda x.\sigma(t);\mathfrak{s}) \in |A \to B|^{\mathfrak{S}}_{\rho}$. Let then u be a term such that $(u;\mathfrak{s}) \in |A|^{\mathfrak{S}}_{\rho}$. By definition, we have $\lambda x.\sigma(t)\,u\;{}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\;\sigma(t)[u/x]$. Since $\sigma(t)[u/x] = (\sigma, x := u)(t)$ and $(\sigma, x := u;\mathfrak{s}) \Vdash \rho(\Gamma, x : A)$, we obtain $(\sigma(t)[u/x];\mathfrak{s}) \in |B|^{\mathfrak{S}}_{\rho}$. We conclude that $(\lambda x.\sigma(t)\,u;\mathfrak{s}) \in |B|^{\mathfrak{S}}_{\rho}$ by anti-reduction. ◀

Proof of Proposition 23. Let $\mathbb{X} : \mathbb{N} \to \mathbf{SAT}$ be a predicate, $\mathfrak{s} \in \mathfrak{S}$, $f \in \mathbb{N}^{\mathfrak{S}}$, u_0 and u_S be terms, and V be a value such that $(u_0;\mathfrak{s}) \in \mathbb{X}(0)$, $(u_S;\mathfrak{s}) \in |\forall^{\mathbb{N}} y.(X(y) \to X(S(y)))|^{\mathfrak{S}}_{X \mapsto \mathbb{X}}$ and $(\overline{n};\mathfrak{s}) \in |\mathrm{Nat}(f)|^{\mathfrak{S}}$. The latter implies that $n = f(\mathfrak{s})$. The result follows from the fact that $\mathsf{rec}\;u_0\,u_S\,\overline{n} \in \mathbb{X}(n)$, which is proved by induction on n. ◀

B.3 Glueing

Proof of Theorem 26 (Glueing). The proof is by induction on the structure of A. ◀

Proof of Proposition 27. We consider $\mathcal{F} : (n_1, ..., n_k) \mapsto |B[x_1 := n_1^*, ..., x_k := n_k^*]|^{\mathfrak{S}}_{\rho}$ which defines a function from $\mathbb{N}^k$ to $\mathbf{SAT}$. By an easy induction on A, we show using the glueing theorem and Definition 13 that $|A|^{\mathfrak{S}}_{\rho, X \mapsto \mathcal{F}} = |A[X(x_1, ..., x_k) := B]|^{\mathfrak{S}}_{\rho}$. ◀

B.4 Natural numbers

▶ **Proposition 48.** *Let $f \in \mathbb{N}^{\mathfrak{S}}$ and $\mathfrak{s} \in \mathfrak{S}$. If t is a term such that $(t;\mathfrak{s}) \in |\mathrm{Nat}'(f)|^{\mathfrak{S}}$, then $t\,\lambda x.x \;{}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\; \overline{n}$, where $n = f(\mathfrak{s})$.*

Proof. Let us define $\mathbb{X} \triangleq \{(t;\mathfrak{s}') : t \;{}^{\mathfrak{s}}{\downarrow}^{'}\; \mathfrak{s}\overline{n}\}$. This set is clearly saturated, and it is easy to see that $(\lambda x.x;\mathfrak{s}) \in |\{\mathrm{Nat}(f)\} \mapsto \mathbb{X}|^{\mathfrak{S}}$ (since $\lambda x.x\,\overline{n} \;{}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\; \overline{n}$). Therefore, we have that $t \in |(\{\mathrm{Nat}(f)\} \mapsto \mathbb{X}) \to \mathbb{X}|^{\mathfrak{S}}$ and then $(t\,\lambda x.x;\mathfrak{s}) \in \mathbb{X}$, that is $t\,\lambda x.x \;{}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\; \overline{n}$. ◀

Proof of Proposition 29. Easy realizability proofs by anti-reduction. ◀

▶ **Lemma 49.** *Let $\mathfrak{s} \in \mathfrak{S}$ and t, u be terms.*

1. *For any $n \in \mathbb{N}$, if $u \;{}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\; \overline{n}$, then $T\,t\,u \;{}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\; t\,\overline{n}$.*
2. *For any $f \in \mathbb{N}^{\mathfrak{S}}$, if $u \;{}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\; \overline{f(\mathfrak{s})}$ and $(t;\mathfrak{s}) \in |\forall^{\{\mathbb{N}\}}x.A(x)|^{\mathfrak{S}}$, then $T\,t\,u \in |A(f)|^{\mathfrak{S}}$.*

Proof. The first part is an easy induction on n, and the second part follows from the first by anti-reduction. ◀

Proof of Proposition 30. The first three parts are easy, and the fourth one is similar to Proposition 12 using Lemma 49. ◀

Proof of Proposition 32. 1. If either f or g is not standard, the result is trivial. Assume that f and g are standard. If $f \neq g$, we have $|(\forall Z.(Z(f) \mapsto Z(g))|^{\mathfrak{S}} = |\top \mapsto \bot|^{\mathfrak{S}}$, while the case $f = g$ is trivial.
2. The result easily follows from Proposition 27. ◀

B.5 Nonstandard reasoning principles

Proof of Theorem 33 (Transfer). Parts 1 and 4 follow from the glueing theorem, while parts 2 and 3 (resp. 5, 6) are direct consequences of the first (resp. fourth) part. ◀

Proof of Proposition 34. Both statements follow by unfolding the definitions. ◀

Proof of Proposition 35. Let $\mathfrak{s}$ be a state, $n \in \mathbb{N}$ and u_0, u_S be terms and V be a value such that $(u_0;\mathfrak{s}) \in |A(0^*)|^{\mathfrak{S}}$, $(u_S;\mathfrak{s}) \in |\forall^{\mathrm{st}}y.(A(y) \to A(S(y))|^{\mathfrak{S}}$ and $(V;\mathfrak{s}) \in |\mathrm{Nat}(n^*)|^{\mathfrak{S}}$. The latter implies that $V = \overline{n}$. The result follows from the fact that $\mathsf{rec}\; u_0\, u_S\, \overline{n} \in |A(n^*)|^{\mathfrak{S}}$, which is proved by induction on n. ◀

Proof of Proposition 36 (Overspill). We show that $((\lambda x.(x\,t, t))u;\mathfrak{s}) \Vdash \exists x.(\neg\mathrm{st}(x) \wedge A(x))$ where $(u;\mathfrak{s}) \Vdash \forall^{\mathrm{st}}x.A(x)$. Following the proof of part 3 in Theorem 33, we obtain that $(u\,t;\mathfrak{s}) \Vdash \forall x.A(x)$ and consequently $(u\,t;\mathfrak{s}) \Vdash A(\delta)$. By ENS_0 (Proposition 30), we have $(t;\mathfrak{s}) \Vdash \neg\mathrm{st}(\delta)$. Then $((u\,t, t);\mathfrak{s}) \Vdash \exists x.(\neg\mathrm{st}(x) \wedge A(x))$ and we conclude by anti-reduction. ◀

Proof of Proposition 37 (Underspill). Let u and v be terms s.t. $(u;\mathfrak{s}) \Vdash \forall x.\neg\mathrm{st}(x) \to A(x)$ and $(v;\mathfrak{s}) \Vdash \neg\exists^{\mathrm{st}}x.A(x)$. Using Proposition 18, we get that $(v;\mathfrak{s}) \Vdash \forall x.((\mathrm{st}(x) \wedge A(x)) \to \bot)$, and by currying $(\lambda wz.v\,(w,z);\mathfrak{s}) \Vdash \forall^{\mathrm{st}}x.A(x) \to \bot$.

Since A is internal, by transfer, we obtain that$(\lambda z.v\,(t,z);\mathfrak{s}) \Vdash \forall x.A(x) \to \bot$. By the hypothesis on u and ENS_0, we have $(u\,t;\mathfrak{s}) \Vdash A(\delta)$, hence $(\lambda z.v\,(t,z))(u\,t);\mathfrak{s}) \Vdash \bot$, and we can conclude by anti-reduction. ◀

B.6 Idealization

Proof of Proposition 38 (Diagonalization). Let $\mathfrak{s}$ be an arbitrary state. Following the proof of part 2 of Lemma 49, it is clearly enough to prove that $(\lambda xy.\,\mathsf{set}\ y\,\dagger;\mathfrak{s}) \Vdash \forall^{\{\mathrm{st}\}}y.y \leq \delta$ (the rest of the proof is exactly the same replacing $\neg\mathrm{st}(\delta)$ by $\forall^{\{\mathrm{st}\}}y.y \leq \delta$). Let $n \in \mathbb{N}$ and t an arbitrary term. Then

$$(\lambda xy.\,\mathsf{set}\ y\,t)\,t\,\overline{n}\ \ {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\ \ \mathsf{set}\ \overline{n}\,t\ \ {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}'}\ \ t$$

where $\mathfrak{s}' = \max(n,\mathfrak{s})$. In particular, $n \leq \delta(\mathfrak{s}')$ holds, hence $(t;\mathfrak{s}') \in |n \leq \delta|^{\mathfrak{S}}$ and we can conclude by anti-reduction. ◀

Proof of Lemma 40. The proof is analogous to the proof of Proposition 10. ◀

Recall that we define $\mathsf{ideal} \triangleq \lambda x.\lambda y.T\,y\,(\pi_1(T\,(x\,\dagger)\ \mathsf{get}\,(\lambda wyz.(y,z)))\,(\lambda yz.\,\mathsf{set}\ z\,y)$.

Proof of Theorem 41 (Idealization). Let $\mathfrak{s}$ be any state and let u be a term such that $(u;\mathfrak{s}) \in |\forall^{\{\mathrm{st}\}}n.\exists^{\{\mathrm{st}\}}x.\forall^{\{\mathrm{st}\}}y.(y \leq n \to R(x,y))|^{\mathfrak{S}}$. By part 2 of Lemma 49, this entails that

$$(T\,(u\,\dagger)\ \mathsf{get};\mathfrak{s}) \in |\exists^{\{\mathrm{st}\}}x.\forall^{\{\mathrm{st}\}}y.(y \leq \mathfrak{s} \to R(x,y))|^{\mathfrak{S}}_{\rho}.$$

By Lemma 40, we know that there exists a natural number $p_{\mathfrak{s}} \in \mathbb{N}$ and a term $v_{\mathfrak{s}} \in \Lambda$ such that $T\,(u\,\dagger)\ \mathsf{get}\,(\lambda zxy.(x,y))\ {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\ (\overline{p_{\mathfrak{s}}},v_{\mathfrak{s}})$ and $(v_{\mathfrak{s}};\mathfrak{s}) \in |\forall^{\{\mathrm{st}\}}y.(y \leq \mathfrak{s} \to R(p_{\mathfrak{s}},y))|^{\mathfrak{S}}$. The latter implies that for any $m \in \mathbb{N}$ such that $m \leq \mathfrak{s}$ and any term t, it holds that $(v_{\mathfrak{s}}\,t\,\overline{m}\,t;\mathfrak{s}) \in |R(p_{\mathfrak{s}},m)|^{\mathfrak{S}}$ and hence $R(p_{\mathfrak{s}},m)$ holds (otherwise $|R(p_{\mathfrak{s}},m)|_{\mathfrak{s}} = \emptyset$).

Consider the (nonstandard) individual $\tau \in \mathbb{N}^{\mathfrak{S}}$ defined by $\tau(\mathfrak{s}) = p_{\mathfrak{s}}$. We have

$$\mathsf{ideal}\ u\ \ {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\ \ \lambda y.T\,y\,(\pi_1(T\,(u\,\dagger)\ \mathsf{get}\,(\lambda wyz.(y,z))))\,(\lambda yz.\,\mathsf{set}\ z\,y)$$

hence, by part 2 of Lemma 49, to conclude by anti-reduction it suffices to prove that

1. $\underline{\pi_1(T\,(u\,\dagger)\ \mathsf{get}\,(\lambda wyz.(y,z)))\ {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\ \overline{\tau(\mathfrak{s})}}$. Indeed, we know that this term reduces as follows:

   $$\pi_1(T\,(u\,\dagger)\ \mathsf{get}\,(\lambda wyz.(y,z)))\ {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\ \pi_1(\overline{p_{\mathfrak{s}}},v_{\mathfrak{s}})\ {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\ \overline{p_{\mathfrak{s}}}$$

   and by definition $\tau(\mathfrak{s}) = p_{\mathfrak{s}}$.
2. $\underline{(\lambda yz.\,\mathsf{set}\ z\,y;\mathfrak{s}) \Vdash \forall^{\{\mathrm{st}\}}y.R(\tau,y)}$. To prove this, it suffices to show that for any $m \in \mathbb{N}$ and any $t \in \Lambda$, we have $((\lambda yz.\,\mathsf{set}\ z\,y)\,t\,\overline{m};\mathfrak{s}) \Vdash R(\tau,m^*)$. With $\mathfrak{s}' \triangleq \max(\mathfrak{s},m)$, we have that $(\lambda yz.\,\mathsf{set}\ z\,y)\,t\,\overline{m}\ {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\ \mathsf{set}\ \overline{m}\,t\ {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}'}\ t$. By construction, since $m \leq \mathfrak{s}'$, we know that $R(\tau(\mathfrak{s}'),m)$ holds, hence $(t;\mathfrak{s}') \in |R(\tau(\mathfrak{s}'),m)|^{\mathfrak{S}}_{\rho}$ and we conclude by anti-reduction. ◀

C Realizability up to $\mathcal{U}$

Proof of Proposition 42. Follows from the fact that for any nonstandard $f \in \mathbb{N}^{\mathfrak{S}}$, any $n \in \mathbb{N}$ and any $\mathfrak{s} \in \mathfrak{S}$, there exists $\mathfrak{s}' \in \mathfrak{S}$ such that $\mathfrak{s}' > \mathfrak{s}$ and $n < f(\mathfrak{s}')$. The result then follows by anti-reduction from the fact that $\mathsf{loop}^{+}\ {}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}'}\ \mathsf{loop}^{+}$. ◀

We give here the quotiented interpretation referred to in Section 6.1.

▶ **Definition 50** (Realizability up to $\mathcal{U}$). *The interpretation of a formula A together with a valuation ρ is the set $|A|^{*}_{\rho}$ defined inductively according to the following clauses:*

$$\begin{array}{rcl}
|\mathrm{st}(f)|^*_\rho & \triangleq & \begin{cases} \Lambda & \textit{if } f \cong n^*, \textit{ for some } n \in \mathbb{N} \\ \emptyset & \textit{otherwise} \end{cases} \\
|X(e_1,\dots,e_n)|^*_\rho & \triangleq & \{t \in \Lambda : \{\mathfrak{s} \in \mathfrak{S} : (t;\mathfrak{s}) \in \rho(X)@([\![e_1]\!]_\rho,\dots,[\![e_n]\!]_\rho)\} \in \mathcal{U}\} \\
|\{\mathrm{Nat}(e)\} \mapsto A|^*_\rho & \triangleq & \{t \in \Lambda : \{\mathfrak{s} \in \mathfrak{S} : (t;\mathfrak{s}) \in |\{\mathrm{Nat}(e)\} \mapsto A|^{\mathfrak{S}}_\rho\} \in \mathcal{U}\}\} \\
|A \to B|^*_\rho & \triangleq & \{t \in \Lambda : \{\mathfrak{s} \in \mathfrak{S} : (t;\mathfrak{s}) \in |A \to B|^{\mathfrak{S}}_\rho\} \in \mathcal{U}\} \\
|A_1 \wedge A_2|^*_\rho & \triangleq & \{t \in \Lambda : \{\mathfrak{s} \in \mathfrak{S} : (\pi_1(t);\mathfrak{s}) \in |A_1|^{\mathfrak{S}}_\rho \wedge (\pi_2(t);\mathfrak{s}) \in |A_2|^{\mathfrak{S}}_\rho\} \in \mathcal{U}\} \\
|A_1 \vee A_2|^*_\rho & \triangleq & \{t \in \Lambda : \{\mathfrak{s} \in \mathfrak{S} : \exists i \in \{1,2\}.\,\mathsf{case}\ t\,\{\iota_1(x_1) \mapsto x_1 | \iota_2(x_2) \mapsto x_2\} \in |A_i|^{\mathfrak{S}}_\rho\} \in \mathcal{U}\} \\
|\forall x.A|^*_\rho & \triangleq & \bigcap_{f \in \mathbb{N}^{\mathfrak{S}}} |A|^*_{\rho,x\mapsto f} \qquad |\forall X.A|^*_\rho \triangleq \bigcap_{F:\mathbb{N}^k \to \mathbf{SAT}} |A|^*_{\rho,X\mapsto F} \\
|\exists x.A|^*_\rho & \triangleq & \bigcup_{f \in \mathbb{N}^{\mathfrak{S}}} |A|^*_{\rho,x\mapsto f} \qquad |\exists X.A|^*_\rho \triangleq \bigcup_{F:\mathbb{N}^k \to \mathbf{SAT}} |A|^*_{\rho,X\mapsto F}
\end{array}$$

We write $t \Vdash^* A$ *if* $t \in |A|^*$.

As explained in Section 6.1, this definition is meant to satisfy a counterpart of Łoś' theorem in our setting.

Proof of Theorem 44 (Łoś' theorem). The proof goes by induction on the structure of A. In the cases $\{\mathrm{Nat}(e)\} \mapsto A$, $X(e_1,\dots,e_n)$, $A \to B$, $A \vee B$ and $A \wedge B$, the result follows directly from the definitions. The proof for quantifiers is similar to the usual proof of Łoś' theorem, we only give here the case of the existential quantifier.

Case $\exists x.A$ By the induction hypothesis, we have that for any $f \in \mathbb{N}^{\mathfrak{S}}$,

$$|A|^*_{\rho,x\mapsto f} = \{t : \{\mathfrak{s} \in \mathfrak{S} : t;\mathfrak{s} \in |A|^{\mathfrak{S}}_{\rho,x\mapsto f}\} \in \mathcal{U}\}$$

By glueing, we have that $|A|^{\mathfrak{S}}_{\rho,x\mapsto f} = |\overline{A}^{\mathfrak{s}}|^{\mathfrak{s}}_{\rho,x\mapsto f} = |\overline{A}^{\mathfrak{s}}|^{\mathfrak{s}}_{\rho,x\mapsto(f(\mathfrak{s}))^*}$. We want to prove that for any $t \in \Lambda$

$$\exists f \in \mathbb{N}^{\mathfrak{S}}.t \in |A|^*_{\rho,x\mapsto f} \text{ iff } \{\mathfrak{s} \in \mathfrak{S} : t;\mathfrak{s} \in |\exists x.A|^{\mathfrak{S}}_\rho\} \in \mathcal{U}$$

Observe that, by glueing, the right-hand side is equivalent to $\{\mathfrak{s} \in \mathfrak{S} : \exists n \in \mathbb{N}.t \in |A|^{\mathfrak{s}}_{\rho,x\mapsto n^*}\} \in \mathcal{U}$.

$\underline{\Rightarrow|}$ If there exists $f \in \mathbb{N}^{\mathfrak{S}}$ such that $t \in |A|^*_{\rho,x\mapsto f}$. We easily see that

$$\{\mathfrak{s} \in \mathfrak{S} : t \in |A|^{\mathfrak{s}}_{\rho,x\mapsto(f(\mathfrak{s}))^*}\} \subseteq \{\mathfrak{s} \in \mathfrak{S} : \exists n \in \mathbb{N}.t \in |A|^{\mathfrak{s}}_{\rho,x\mapsto n^*}\}$$

hence we can conclude by upwards closure of the ultrafilter.

$\underline{\Leftarrow|}$ Assume that $E \triangleq \{\mathfrak{s} \in \mathfrak{S} : \exists n \in \mathbb{N}.t \in |A|^{\mathfrak{s}}_{\rho,x\mapsto n^*}\} \in \mathcal{U}$

For any $\mathfrak{s} \in E$, using countable choice we can pick an integer $n_{\mathfrak{s}}$ such that $t \in |A|^{\mathfrak{s}}_{\rho,x\mapsto n^*_{\mathfrak{s}}}$. We may then define the function $g \in \mathbb{N}^{\mathfrak{S}}$ by $g(\mathfrak{s}) \triangleq n_{\mathfrak{s}}$ if $\mathfrak{s} \in E$, 0 otherwise. By definition, $E \subseteq \{\mathfrak{s} \in \mathfrak{S} : t \in |A|^{\mathfrak{s}}_{\rho,x\mapsto(g(\mathfrak{s}))^*}\}$, hence this set belongs to $\mathcal{U}$ by upwards closure. Therefore we can conclude by induction hypothesis that $t \in |A|^*_{\rho,x\mapsto f}$. ◀

▶ **Proposition 51.** *For any internal formulas A and B, and any valuation ρ closing both A and B, we have $|A \to B|^*_\rho \subseteq \{t : \forall u \in |A|^*_\rho.t\,u \in |B|^*_\rho\}$.*

Proof. For any term t and any formula A, let us denote by S^A_t the set $\{\mathfrak{s} \in \mathfrak{S} : (t;\mathfrak{s}) \in |A|^{\mathfrak{S}}_\rho\}$. Let $t \in \Lambda$ be such that $S^{A\to B}_t \in \mathcal{U}$ and $u \in |A|^*_\rho$. By hypothesis, $S^A_u \in \mathcal{U}$. We need to show that $tu \in |B|^*_\rho$. Again, for any $\mathfrak{s} \in S^{A\to B}_t \cap S^A_u \in \mathcal{U}$, we have $tu;\mathfrak{s} \in |B|^{\mathfrak{S}}_\rho$. By upwards closure, we deduce that $\{\mathfrak{s} : (tu;\mathfrak{s}) \in |B|^{\mathfrak{S}}_\rho\} \in \mathcal{U}$, hence $tu \in |B|^*_\rho$, and the result follows from Theorem 44. ◀

▶ Remark 52. One could have been tempted to define the truth value $|A \to B|_\rho^*$ as the set of terms t such that for any $u \in |A|_\rho^*$, $t\,u \in |B|_\rho^*$, as is usual in realizability. Unfortunately, such a definition is incompatible with Theorem 44, as the other inclusion in Proposition 51 does not hold. To see this, let $A \triangleq \mathrm{Nat}'(\tau)$ and $B \triangleq \bot$ where τ is a non-computable function[7] $\tau : \mathfrak{S} \to \mathbb{N}$ for which there is no term u such that $\forall \mathfrak{s}. u \,{}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\, \tau(\mathfrak{s})$. By construction, we have that $|A|^* = \emptyset$, so that obviously for any $u \in |\mathrm{Nat}'(\tau)|^*$, the function $(\lambda x.x)\,u \in |\bot|^*$. Yet, for each state $\mathfrak{s}$ the truth value $|\mathrm{Nat}'(\tau)|_\rho^{\mathfrak{S}}$ is not empty (it contains at least $(\overline{n}, \mathfrak{s})$, for $n = \tau(\mathfrak{s})$) and therefore $(\lambda x.x; \mathfrak{s}) \notin |\mathrm{Nat}'(\tau) \to \bot|_\rho^{\mathfrak{S}}$ (since for any $(u;\mathfrak{s}) \in |\mathrm{Nat}'(\tau)|^{\mathfrak{S}}$, $(\lambda x.x\,u;\mathfrak{s}) \notin |\bot|^*$).

We want to point out that Remark 52 highlights the “counter-intuitive” peculiarities of the interpretation up to $\mathcal{U}$ with respect to the quotient in the Lightstone-Robinson construction. The latter indeed appears to be more regular, seemingly for two main reasons. First, as we highlight in Section 4, in the stateful interpretation the set instruction allows terms to change the value of the states during computations, and thus of the slices. This phenomenon does not occur in the Lightstone-Robinson construction where slices of the product are complety isolated between them. Second, while the Lightstone-Robinson construction is based on Boolean-valued models, realizability interpretations associate to each formula a set of realizers (instead of one unique Boolean). Besides, the use of relativized quantifiers (for instance in the statement for idealization) forces us to use only computable functions[8].

[7] To that end, one can for instance consider the function τ which to each $\mathfrak{s} \in \mathfrak{S}$ associates the smallest natural number $n \in \mathbb{N}$ such that there is no term of size smaller than or equal to $\mathfrak{s}$ that computes n the state $\mathfrak{s}$: $\tau(\mathfrak{s}) \triangleq \inf\{n \in \mathbb{N} : \neg\exists t. |t| \leq \mathfrak{s} \wedge t \,{}^{\mathfrak{s}}{\downarrow}^{\mathfrak{s}}\, \overline{n}\}$.

[8] This is the reason why, for instance, the premise of idealization needs to be restricted to the existence of a *standard* natural number x, instead of any natural number as is usually the case.

Decidable Entailments in Separation Logic with Inductive Definitions: Beyond Establishment

Mnacho Echenim
Université Grenoble Alpes, CNRS, LIG, F-38000 Grenoble, France

Radu Iosif
Université Grenoble Alpes, CNRS, VERIMAG, F-38000 Grenoble, France

Nicolas Peltier
Université Grenoble Alpes, CNRS, LIG, F-38000 Grenoble, France

Abstract

We define a class of Separation Logic [10, 16] formulæ, whose entailment problem *given formulæ $\phi, \psi_1, \ldots, \psi_n$, is every model of ϕ a model of some ψ_i?* is 2-EXPTIME-complete. The formulæ in this class are existentially quantified separating conjunctions involving predicate atoms, interpreted by the least sets of store-heap structures that satisfy a set of inductive rules, which is also part of the input to the entailment problem. Previous work [8, 12, 15] consider *established* sets of rules, meaning that every existentially quantified variable in a rule must eventually be bound to an *allocated* location, i.e. from the domain of the heap. In particular, this guarantees that each structure has treewidth bounded by the size of the largest rule in the set. In contrast, here we show that establishment, although sufficient for decidability (alongside two other natural conditions), is not necessary, by providing a condition, called *equational restrictedness*, which applies syntactically to (dis-)equalities. The entailment problem is more general in this case, because equationally restricted rules define richer classes of structures, of unbounded treewidth. In this paper we show that

(1) every established set of rules can be converted into an equationally restricted one and

(2) the entailment problem is 2-EXPTIME-complete in the latter case, thus matching the complexity of entailments for established sets of rules [12, 15].

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Separation logic, Induction definitions, Inductive theorem proving, Entailments, Complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.20

Related Version A full version of the paper is available at `https://arxiv.org/abs/2007.00502`.

1 Introduction

Separation Logic (SL) [10, 16] is widely used to reason about programs manipulating recursively linked data structures, being at the core of several industrial-scale static program analysis techniques [3, 2, 5]. Given an integer $\mathfrak{K} \geq 1$, denoting the number of fields in a record datatype, and an infinite set $\mathbb{L}$ of memory locations (addresses), the assertions in this logic describe *heaps*, that are finite partial functions mapping locations to records, i.e., $\mathfrak{K}$-tuples of locations. A location ℓ in the domain of the heap is said to be *allocated* and the *points-to* atom $x \mapsto (y_1, \ldots, y_{\mathfrak{K}})$ states that the location associated with x refers to the tuple of locations associated with $(y_1, \ldots, y_{\mathfrak{K}})$. The *separating conjunction* $\phi * \psi$ states that the formulæ ϕ and ψ hold in non-overlapping parts of the heap, that have disjoint domains. This connective allows for modular program analyses, because the formulæ specifying the behaviour of a program statement refer only to the small (local) set of locations that are manipulated by that statement, with no concern for the rest of the program's state.

Formulæ consisting of points-to atoms connected with separating conjunctions describe heaps of bounded size only. To reason about recursive data structures of unbounded sizes (lists, trees, etc.), the base logic is enriched by predicate symbols, with a semantics specified

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 20; pp. 20:1–20:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

by user-defined inductive rules. For instance, the rules: $\mathsf{excls}(x, y) \Leftarrow \exists z \,.\, x \mapsto (z, y) * z \not\simeq \mathsf{c}$ and $\mathsf{excls}(x, y) \Leftarrow \exists z \exists v \,.\, x \mapsto (z, v) * \mathsf{excls}(v, y) * z \not\simeq \mathsf{c}$ describe a non-empty list segment, whose elements are records with two fields: the first is a data field, that keeps a list of locations, which excludes the location assigned to the global constant c, and the second is used to link the records in a list whose head and tail are pointed to by x and y, respectively.

An important problem in program verification, arising during construction of Hoare-style correctness proofs, is the discharge of verification conditions, that are entailments of the form $\phi \vdash \psi_1, \ldots, \psi_n$, where ϕ and $\psi_1, \ldots, \psi_n$ are separating conjunctions of points-to, predicates and (dis-)equalities, also known as *symbolic heaps*. The *entailment problem* then asks if *every model of ϕ is a model of some ψ_i?* In general, the entailment problem is undecidable and becomes decidable when the inductive rules used to interpret the predicates satisfy three restrictions [8]:

(1) *progress*, stating that each rule allocates *exactly* one memory cell,
(2) *connectivity*, ensuring that the allocated memory cells form a tree-shaped structure, and
(3) *establishment*, stating that all existentially quantified variables introduced by an inductive rule must be assigned to some allocated memory cell, in every structure defined by that rule.

For instance, the above rules are progressing and connected but not established, because the $\exists z$ variables are not explicitly assigned an allocated location, unlike the $\exists v$ variables, passed as first parameter of the $\mathsf{excls}(x, y)$ predicate, and thus always allocated by the points-to atoms $x \mapsto (z, y)$ or $x \mapsto (z, v)$, from the first and second rule defining $\mathsf{excls}(x, y)$, respectively.

The argument behind the decidability of a progressing, connected and established entailment problem is that every model of the left-hand side is encoded by a graph whose treewidth[1] is bounded by the size of the largest symbolic heap that occurs in the problem [8]. Moreover, the progress and connectivity conditions ensure that the set of models of a symbolic heap can be represented by a Monadic Second Order (MSO) logic formula interpreted over graphs, that can be effectively built from the symbolic heap and the set of rules of the problem. The decidability of entailments follows then from the decidability of the satisfiability problem for MSO over graphs of bounded treewidth (Courcelle's Theorem) [4]. Initially, no upper bound better than elementary recursive was known to exist. Recently, a 2-EXPTIME algorithm was proposed [12, 14] for sets of rules satisfying these three conditions, and, moreover, this bound was shown to be tight [6].

Several natural questions arise: are the progress, connectivity and establishment conditions really necessary for the decidability of entailments? How much can these restriction be relaxed, without jeopardizing the complexity of the problem? Can one decide entailments that involve sets of heaps of unbounded treewidth? In this paper, we answer these questions by showing that entailments are still 2-EXPTIME-complete when the establishment condition is replaced by a condition on the (dis-)equations occurring in the symbolic heaps of the problem. Informally, such (dis-)equations must be of the form $x \simeq \mathsf{c}$ ($x \not\simeq \mathsf{c}$), where c ranges over some finite and fixed set of globally visible constants (including special symbols such as nil, that denotes a non-allocated address, but also any free variable occurring on the left-hand side of the entailment). We also relax slightly the progress and connectivity conditions, by allowing forest-like heap structures (instead of just trees), provided that every root is mapped to a constant symbol. These entailment problems are called *equationally restricted* (*e-restricted*, for short). For instance, the entailment problem $\mathsf{excls}(x, y) * \mathsf{excls}(y, z) \vdash \mathsf{excls}(x, z)$, with the above rules, falls in this category.

[1] The treewidth of a graph is a parameter measuring how close the graph is to a tree, see [7, Ch. 11] for a definition.

We prove that the e-restricted condition loses no generality compared to establishment, because any established entailment problem can be transformed into an equivalent e-restricted entailment problem. E-restricted problems allow reasoning about structures that contain dangling pointers, which frequently occur in practice, especially in the context of modular program analysis. Moreover, the set of structures considered in an e-restricted entailment problem may contain infinite sequences of heaps of strictly increasing treewidths, that are out of the scope of established problems [8].

The decision procedure for e-restricted problems proposed in this paper is based on a similar idea as the one given, for established problems, in [14, 15]. We build a suitable abstraction of the set of structures satisfying the left-hand side of the entailment bottom-up, starting from points-to and predicate atoms, using abstract operators to compose disjoint structures, to add and remove variables, and to unfold the inductive rules associated with the predicates. The abstraction is precise enough to allow checking that all the models of the left-hand side fulfill the right-hand side of the entailment and also general enough to ensure termination of the entailment checking algorithm.

Although both procedures are similar, there are essential differences between our work and [14, 15]. First, we show that instead of using a specific language for describing those abstractions, the considered set of structures can themselves be defined in SL, by means of formulæ of some specific pattern called *core formulæ*. Second, the fact that the systems are not established makes the definition of the procedure much more difficult, due to the fact that the considered structures can have an unbounded treewidth. This is problematic because, informally, this boundedness property is essential to ensure that the abstractions can be described using a finite set of variables, denoting the *frontier* of the considered structures, namely the locations that can be shared with other structures. In particular, the fact that disjoint heaps may share unallocated (or "unnamed") locations complexifies the definition of the composition operator. This problem is overcome by considering a specific class of structures, called *normal structures*, of bounded treewidth, and proving that the validity of an entailment can be decided by considering only normal structures.

In terms of complexity, we show that the running time of our algorithm is doubly exponential w.r.t. the maximal size among the symbolic heaps occurring in the input entailment problem (including those in the rules) and simply exponential w.r.t. the number of such symbolic heaps (hence w.r.t. the number of rules). This means that the 2-EXPTIME upper bound is preserved by any reduction increasing exponentially the number of rules, but increasing only polynomially the size of the rules. On the other hand, the 2-EXPTIME-hard lower bound is proved by a reduction from the membership problem for exponential-space bounded Alternating Turing Machines [6].

2 Separation Logic with Inductive Definitions

Let $\mathbb{N}$ denote the set of natural numbers. For a countable set S, we denote by $||S|| \in \mathbb{N} \cup \{\infty\}$ its cardinality. For a partial mapping $f : A \rightharpoonup B$, let $\mathrm{dom}(f) \stackrel{\mathsf{def}}{=} \{x \in A \mid f(x) \in B\}$ and $\mathrm{rng}(f) \stackrel{\mathsf{def}}{=} \{f(x) \mid x \in \mathrm{dom}(f)\}$ be its domain and range, respectively. We say that f is *total* if $\mathrm{dom}(f) = A$, written $f : A \to B$ and *finite*, written $f : A \rightharpoonup_{\mathit{fin}} B$ if $||\mathrm{dom}(f)|| < \infty$. Given integers n and m, we denote by $[\![n \mathinner{..} m]\!]$ the set $\{n, n+1, \ldots, m\}$, so that $[\![n \mathinner{..} m]\!] = \emptyset$ if $n > m$. For a relation $\lhd \subseteq A \times A$, we denote by $\lhd^*$ its reflexive and transitive closure.

For an integer $n \geq 0$, let A^n be the set of n-tuples with elements from A. Given a tuple $\mathbf{a} = (a_1, \ldots, a_n)$ and $i \in [\![1 \mathinner{..} n]\!]$, we denote by $\mathbf{a}_i$ the i-th element of $\mathbf{a}$ and by $|\mathbf{a}| \stackrel{\mathsf{def}}{=} n$ its length. By $f(\mathbf{a})$ we denote the tuple obtained by the pointwise application of f to the elements of $\mathbf{a}$. If multiplicity and order of the elements are not important, we blur the distinction between tuples and sets, using the set-theoretic notations $x \in \mathbf{a}$, $\mathbf{a} \cup \mathbf{b}$, $\mathbf{a} \cap \mathbf{b}$ and $\mathbf{a} \setminus \mathbf{b}$.

Let $\mathbb{V} = \{x, y, \ldots\}$ be an infinite countable set of logical first-order variables and $\mathbb{P} = \{p, q, \ldots\}$ be an infinite countable set (disjoint from $\mathbb{V}$) of relation symbols, called *predicates*, where each predicate p has arity $\#p \geq 0$. We also consider a finite set $\mathbb{C}$ of *constants*, of known bounded cardinality, disjoint from both $\mathbb{V}$ and $\mathbb{P}$. Constants will play a special rôle in the upcoming developments and the fact that $\mathbb{C}$ is bounded is of a particular importance. A *term* is either a variable or a constant and we denote by $\mathbb{T} \overset{\mathsf{def}}{=} \mathbb{V} \cup \mathbb{C}$ the set of terms.

Throughout this paper we consider an integer $\mathfrak{K} \geq 1$ that, intuitively, denotes the number of fields in a record datatype. Although we do not assume $\mathfrak{K}$ to be a constant in any of the algorithms presented in the following, considering that every datatype has exactly $\mathfrak{K}$ records simplifies the definition. The logic $\mathsf{SL}^{\mathfrak{K}}$ is the set of formulæ generated inductively by the syntax:

$$\phi \quad := \quad \mathsf{emp} \mid t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}) \mid p(t_1, \ldots, t_{\#p}) \mid t_1 \approx t_2 \mid \phi_1 * \phi_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \exists x \;.\; \phi_1$$

where $p \in \mathbb{P}$, $t_i \in \mathbb{T}$ and $x \in \mathbb{V}$. Atomic propositions of the form $t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})$ are called *points-to atoms* and those of the form $p(t_1, \ldots, t_{\#p})$ are *predicate atoms*. If $\mathfrak{K} = 1$, we write $t_0 \mapsto t_1$ for $t_0 \mapsto (t_1)$.

The connective $*$ is called *separating conjunction*, in contrast with the classical conjunction $\wedge$. The *size* of a formula ϕ, denoted by $\mathrm{size}(\phi)$, is the number of occurrences of symbols in it. We write $\mathsf{fv}(\phi)$ for the set of *free* variables in ϕ and $\mathsf{trm}(\phi) \overset{\mathsf{def}}{=} \mathsf{fv}(\phi) \cup \mathbb{C}$. A formula is *predicate-free* if it has no predicate atoms. As usual, $\phi_1 \vee \phi_2 \overset{\mathsf{def}}{=} \neg(\neg\phi_1 \wedge \neg\phi_2)$ and $\forall x \;.\; \phi \overset{\mathsf{def}}{=} \neg\exists x \;.\; \neg\phi$. For a set of variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ and a quantifier $Q \in \{\exists, \forall\}$, we write $Q\mathbf{x} \;.\; \phi \overset{\mathsf{def}}{=} Qx_1 \ldots Qx_n \;.\; \phi$. By writing $t_1 = t_2$ ($\phi_1 = \phi_2$) we mean that the terms (formulæ) t_1 and t_2 (ϕ_1 and ϕ_2) are syntactically the same.

A *substitution* is a partial mapping $\sigma : \mathbb{V} \rightharpoonup \mathbb{T}$ that maps variables to terms. We denote by $[t_1/x_1, \ldots, t_n/x_n]$ the substitution that maps the variable x_i to t_i, for each $i \in [\![1 \mathinner{..} n]\!]$ and is undefined elsewhere. By $\phi\sigma$ we denote the formula obtained from ϕ by substituting each variable $x \in \mathsf{fv}(\phi)$ by $\sigma(x)$ (we assume that bound variables are renamed to avoid collisions if needed). By abuse of notation, we sometimes write $\sigma(x)$ for x, when $x \notin \mathrm{dom}(\sigma)$.

To interpret $\mathsf{SL}^{\mathfrak{K}}$ formulæ, we consider an infinite countable set $\mathbb{L}$ of *locations*. The semantics of $\mathsf{SL}^{\mathfrak{K}}$ formulæ is defined in terms of *structures* $(\mathfrak{s}, \mathfrak{h})$, where:

- $\mathfrak{s} : \mathbb{T} \rightharpoonup \mathbb{L}$ is a partial mapping of terms into locations, called a *store*, that interprets at least all the constants, i.e. $\mathbb{C} \subseteq \mathrm{dom}(\mathfrak{s})$ for every store $\mathfrak{s}$, and
- $\mathfrak{h} : \mathbb{L} \rightharpoonup_{fin} \mathbb{L}^{\mathfrak{K}}$ is a finite partial mapping of locations into $\mathfrak{K}$-tuples of locations, called a *heap*.

Given a heap $\mathfrak{h}$, let $\mathrm{loc}(\mathfrak{h}) \overset{\mathsf{def}}{=} \{\ell_0, \ldots, \ell_{\mathfrak{K}} \mid \ell_0 \in \mathrm{dom}(\mathfrak{h}),\ \mathfrak{h}(\ell_0) = (\ell_1, \ldots, \ell_{\mathfrak{K}})\}$ be the set of locations that occur in the heap $\mathfrak{h}$. Two heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are *disjoint* iff $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) = \emptyset$, in which case their *disjoint union* is denoted by $\mathfrak{h}_1 \uplus \mathfrak{h}_2$, otherwise undefined. The *frontier between* $\mathfrak{h}_1$ *and* $\mathfrak{h}_2$ is the set of common locations $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) \overset{\mathsf{def}}{=} \mathrm{loc}(\mathfrak{h}_1) \cap \mathrm{loc}(\mathfrak{h}_2)$. Note that disjoint heaps may have nonempty frontier. The *satisfaction relation* $\models$ between structures $(\mathfrak{s}, \mathfrak{h})$ and predicate-free $\mathsf{SL}^{\mathfrak{K}}$ formulæ ϕ is defined recursively on the structure of formulæ:

$$\begin{array}{ll}
(\mathfrak{s}, \mathfrak{h}) \models t_1 \approx t_2 & \Leftrightarrow t_1, t_2 \in \mathrm{dom}(\mathfrak{s}) \text{ and } \mathfrak{s}(t_1) = \mathfrak{s}(t_2) \\
(\mathfrak{s}, \mathfrak{h}) \models \mathsf{emp} & \Leftrightarrow \mathfrak{h} = \emptyset \\
(\mathfrak{s}, \mathfrak{h}) \models t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}) & \Leftrightarrow t_0, \ldots, t_{\mathfrak{K}} \in \mathrm{dom}(\mathfrak{s}),\ \mathrm{dom}(\mathfrak{h}) = \{\mathfrak{s}(t_0)\} \text{ and } \mathfrak{h}(\mathfrak{s}(t_0)) = (\mathfrak{s}(t_1), \ldots, \mathfrak{s}(t_{\mathfrak{K}})) \\
(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \wedge \phi_2 & \Leftrightarrow (\mathfrak{s}, \mathfrak{h}) \models \phi_i,\ i = 1, 2 \\
(\mathfrak{s}, \mathfrak{h}) \models \neg\phi_1 & \Leftrightarrow \mathsf{fv}(\phi_1) \subseteq \mathrm{dom}(\mathfrak{s}) \text{ and } (\mathfrak{s}, \mathfrak{h}) \not\models \phi_1 \\
(\mathfrak{s}, \mathfrak{h}) \models \phi_1 * \phi_2 & \Leftrightarrow \text{there exist heaps } \mathfrak{h}_1, \mathfrak{h}_2 \text{ such that } \mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2 \text{ and } (\mathfrak{s}, \mathfrak{h}_i) \models \phi_i,\ i = 1, 2 \\
(\mathfrak{s}, \mathfrak{h}) \models \exists x \;.\; \phi & \Leftrightarrow (\mathfrak{s}[x \leftarrow \ell], \mathfrak{h}) \models \phi,\ \text{ for some location } \ell \in \mathbb{L}
\end{array}$$

where $\mathfrak{s}[x \leftarrow \ell]$ is the store, with domain $\mathrm{dom}(\mathfrak{s}) \cup \{x\}$, that maps x to ℓ and behaves like $\mathfrak{s}$ over $\mathrm{dom}(\mathfrak{s}) \setminus \{x\}$. For a tuple of variables $\mathbf{x} = (x_1, \ldots, x_n)$ and locations $\mathbf{l} = (l_1, \ldots, l_n)$, we call the store $\mathfrak{s}[\mathbf{x} \leftarrow \mathbf{l}] \stackrel{\text{def}}{=} \mathfrak{s}[x_1 \leftarrow l_1] \ldots [x_n \leftarrow l_n]$ an $\mathbf{x}$-*associate* of $\mathfrak{s}$. A structure $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models \phi$, is called a *model* of ϕ. Note that $(\mathfrak{s}, \mathfrak{h}) \models \phi$ only if $\mathsf{fv}(\phi) \subseteq \mathrm{dom}(\mathfrak{s})$.

The fragment of *symbolic heaps* is obtained by confining the negation and conjunction to the formulæ $t_1 \doteq t_2 \stackrel{\text{def}}{=} t_1 \approx t_2 \wedge \mathsf{emp}$ and $t_1 \not\doteq t_2 \stackrel{\text{def}}{=} \neg t_1 \approx t_2 \wedge \mathsf{emp}$, called *equational atoms*, by abuse of language. We denote by $\mathsf{SH}^{\mathfrak{K}}$ the set of symbolic heaps, formally defined below:

$$\phi \quad := \quad \mathsf{emp} \mid t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}) \mid p(t_1, \ldots, t_{\#p}) \mid t_1 \doteq t_2 \mid t_1 \not\doteq t_2 \mid \phi_1 * \phi_2 \mid \exists x \,.\, \phi_1$$

Given quantifier-free symbolic heaps $\phi_1, \phi_2 \in \mathsf{SH}^{\mathfrak{K}}$, it is not hard to check that $\exists x \,.\, \phi_1 * \exists y \,.\, \phi_2$ and $\exists x \exists y \,.\, \phi_1 * \phi_2$ have the same models (provided $x \neq y$). Consequently, each symbolic heap can be written in prenex form, as $\phi = \exists x_1 \ldots \exists x_n \,.\, \psi$, where ψ is a quantifier-free separating conjunction of points-to atoms and (dis-)equalities. A variable $x \in \mathsf{fv}(\psi)$ is *allocated* in ϕ iff there exists a (possibly empty) sequence of equalities $x \doteq \ldots \doteq t_0$ and a points-to atom $t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})$ in ψ.

The predicates from $\mathbb{P}$ are intepreted by a given set $\mathcal{S}$ of *rules* $p(x_1, \ldots, x_{\#p}) \Leftarrow \rho$, where ρ is a symbolic heap, such that $\mathsf{fv}(\rho) \subseteq \{x_1, \ldots, x_{\#_p}\}$. We say that $p(x_1, \ldots, x_{\#p})$ is the *head* and ρ is the *body* of the rule. For conciseness, we write $p(x_1, \ldots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho$ instead of $p(x_1, \ldots, x_{\#p}) \Leftarrow \rho \in \mathcal{S}$. In the following, we shall often refer to a given set of rules $\mathcal{S}$.

▶ **Definition 1** (Unfolding). *A formula ψ is a* step-unfolding *of a formula $\phi \in \mathsf{SL}^{\mathfrak{K}}$, written $\phi \Rightarrow_{\mathcal{S}} \psi$, if ψ is obtained by replacing an occurrence of an atom $p(t_1, \ldots, t_{\#p})$ in ϕ with $\rho[t_1/x_1, \ldots, t_{\#p}/x_{\#p}]$, for a rule $p(x_1, \ldots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho$. An* unfolding *of ϕ is a formula ψ such that $\phi \Rightarrow_{\mathcal{S}}^* \psi$.*

It is easily seen that any unfolding of a symbolic heap is again a symbolic heap. We implicitly assume that all bound variables are α-renamed throughout an unfolding, to avoid name clashes. Unfolding extends the semantics from predicate-free to arbitrary $\mathsf{SL}^{\mathfrak{K}}$ formulæ:

▶ **Definition 2.** *Given a structure $(\mathfrak{s}, \mathfrak{h})$ and a formula $\phi \in \mathsf{SL}^{\mathfrak{K}}$, we write $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{S}} \phi$ iff there exists a predicate-free unfolding $\phi \Rightarrow_{\mathcal{S}}^* \psi$ such that $(\mathfrak{s}, \mathfrak{h}) \models \psi$. In this case, $(\mathfrak{s}, \mathfrak{h})$ is an* $\mathcal{S}$-model of ϕ. *For two formulæ $\phi, \psi \in \mathsf{SL}^{\mathfrak{K}}$, we write $\phi \models_{\mathcal{S}} \psi$ iff every $\mathcal{S}$-model of ϕ is an $\mathcal{S}$-model of ψ.*

Note that, if $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{S}} \phi$, then $\mathrm{dom}(\mathfrak{s})$ might have to contain constants that do not occur in ϕ. For instance if $p(x) \Leftarrow_{\mathcal{S}} x \mapsto \mathsf{a}$ is the only rule with head $p(x)$, then any $\mathcal{S}$-model $(\mathfrak{s}, \mathfrak{h})$ must map a to some location, which is taken care of by the assumption $\mathbb{C} \subseteq \mathrm{dom}(\mathfrak{s})$, that applies to any store.

▶ **Definition 3** (Entailment). *Given symbolic heaps $\phi, \psi_1, \ldots, \psi_n$, such that ϕ is quantifier-free and $\mathsf{fv}(\phi) = \mathsf{fv}(\psi_1) = \ldots = \mathsf{fv}(\psi_n) = \emptyset$, the* sequent *$\phi \vdash \psi_1, \ldots, \psi_n$ is* valid for $\mathcal{S}$ *iff $\phi \models_{\mathcal{S}} \bigvee_{i=1}^{n} \psi_i$. An* entailment problem *$\mathcal{P} = (\mathcal{S}, \Sigma)$ consists of a set of rules $\mathcal{S}$ and a set Σ of sequents, asking whether each sequent in Σ is valid for $\mathcal{S}$.*

Note that we consider entailments between formulæ without free variables. This is not restrictive, since any free variable can be replaced by a constant from $\mathbb{C}$, with no impact on the validity status or the computational complexity of the problem. We silently assume that $\mathbb{C}$ contains enough constants to allow this replacement. For conciseness, we write $\phi \vdash_{\mathcal{P}} \psi_1, \ldots, \psi_n$ for $\phi \vdash \psi_1, \ldots, \psi_n \in \Sigma$, where Σ is the set of sequents of $\mathcal{P}$. The following example shows an entailment problem asking whether the concatenation of two acyclic lists is again an acyclic list:

▶ **Example 4.** The entailment problem below consists of four rules, defining the predicates $\mathsf{ls}(x,y)$ and $\mathsf{sls}(x,y,z)$, respectively, and two sequents:

$$\begin{array}{rcl} \mathsf{ls}(x,y) & \Leftarrow & x \mapsto y * x \neq y \mid \exists v \;.\; x \mapsto v * \mathsf{ls}(v,y) * x \neq y \\ \mathsf{sls}(x,y,z) & \Leftarrow & x \mapsto y * x \neq y * x \neq z \mid \exists v \;.\; x \mapsto v * \mathsf{sls}(v,y,z) * x \neq y * x \neq z \end{array}$$

$$\mathsf{ls}(a,b) * \mathsf{ls}(b,c) \vdash \exists x \;.\; a \mapsto x * \mathsf{ls}(x,c) * a \neq c \qquad \mathsf{sls}(a,b,c) * \mathsf{ls}(b,c) \vdash \exists x \;.\; a \mapsto x * \mathsf{ls}(x,c) * a \neq c$$

Here $\mathsf{ls}(x,y)$ describes non-empty acyclic list segments with head and tail pointed to by x and y, respectively. The first sequent is invalid, because c can be allocated within the list segment defined by $\mathsf{ls}(a,b)$, in which case the entire list has a cycle starting and ending with the location associated with c. To avoid the cycle, the left-hand side of the second sequent uses the predicate $\mathsf{sls}(x,y,z)$ describing an acyclic list segment from x to y that skips the location pointed to by z. The second sequent is valid. ⌟

The complexity analysis of the decision procedure described in this paper relies on two parameters. First, the *width* of an entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ is (roughly) the maximum among the sizes of the symbolic heaps occurring in $\mathcal{P}$ and the number of constants in $\mathbb{C}$. Second, the *size* of the entailment problem is (roughly) the number of symbols needed to represent it, namely:

$$\begin{array}{rl} \mathrm{width}(\mathcal{P}) & \stackrel{\text{def}}{=} \max\left(\{\mathrm{size}(\rho) + \#p \mid p(x_1,\ldots,x_{\#p}) \Leftarrow_{\mathcal{S}} \rho\} \cup \{\mathrm{size}(\psi_i) \mid \psi_0 \vdash_{\mathcal{P}} \psi_1,\ldots,\psi_n\} \cup \{||\mathbb{C}||\}\right) \\ \mathrm{size}(\mathcal{P}) & \stackrel{\text{def}}{=} \sum_{p(x_1,\ldots,x_{\#p}) \;\Leftarrow_{\mathcal{S}}\; \rho} (\mathrm{size}(\rho) + \#p) + \sum_{\psi_0 \;\vdash_{\mathcal{P}}\; \psi_1,\ldots,\psi_n} \sum_{i=1}^{n} \mathrm{size}(\psi_i) \end{array}$$

In the next section we give a transformation of entailment problems with a time complexity that is bounded by the product of the size and a simple exponential of the width of the input, such that, moreover, the width of the problem increases by a polynomial factor only. The latter is instrumental in proving the final 2-EXPTIME upper bound on the complexity of the entailment problem.

To alleviate the upcoming technical details, we make the following assumption:

▶ **Assumption 1.** *Distinct constants are always associated with distinct locations: for all stores $\mathfrak{s}$, and for all $c, d \in \mathbb{C}$, we have $c \neq d$ only if $\mathfrak{s}(c) \neq \mathfrak{s}(d)$.*

This assumption loses no generality, because one can enumerate all the equivalence relations on $\mathbb{C}$ and test the entailments separately for each of these relations, by replacing all the constants in the same class by a unique representative[2], while assuming that constants in distinct classes are mapped to distinct locations. The overall complexity of the procedure is still doubly exponential, since the number of such equivalence relations is bounded by the number of partitions of $\mathbb{C}$, that is $2^{\mathcal{O}(||\mathbb{C}|| \cdot \log ||\mathbb{C}||)} = 2^{\mathcal{O}(||\mathrm{width}(\mathcal{P})|| \cdot \log ||\mathrm{width}(\mathcal{P})||)}$, for any entailment problem $\mathcal{P}$. Thanks to Assumption 1, the considered symbolic heaps can be, moreover, safely assumed not to contain atoms $c \bowtie d$, with $\bowtie \in \{\simeq, \neq\}$ and $c, d \in \mathbb{C}$, since these atoms are either unsatisfiable or equivalent to emp.

3 Decidable Classes of Entailments

In general, the entailment problem (Definition 3) is undecidable and we refer the reader to [9, 1] for two different proofs. A first attempt to define a naturally expressive class of formulæ with a decidable entailment problem was reported in [8]. The entailments considered in [8] involve sets of rules restricted by three conditions, recalled below, in a slightly generalized form.

[2] The replacement must be performed also within the inductive rules, not only in the considered formulæ.

First, the *progress* condition requires that each rule adds to the heap exactly one location, associated either to a constant or to a designated parameter. Formally, we consider a mapping $\mathsf{root} : \mathbb{P} \to \mathbb{N} \cup \mathbb{C}$, such that $\mathsf{root}(p) \in [\![1 \mathinner{..} \#p]\!] \cup \mathbb{C}$, for each $p \in \mathbb{P}$. The term $\mathsf{root}(p(t_1, \dots, t_{\#p}))$ denotes either t_i if $\mathsf{root}(p) = i \in [\![1 \mathinner{..} \#p]\!]$, or the constant $\mathsf{root}(p)$ itself if $\mathsf{root}(p) \in \mathbb{C}$. The notation $\mathsf{root}(\alpha)$ is extended to points-to atoms α as $\mathsf{root}(t_0 \mapsto (t_1, \dots, t_{\mathfrak{K}})) \overset{\text{def}}{=} t_0$. Second, the *connectivity* condition requires that all locations added during an unfolding of a predicate atom form a set of connected trees (a forest) rooted in locations associated either with a parameter of the predicate or with a constant.

▶ **Definition 5** (Progress & Connectivity). *A set of rules $\mathcal{S}$ is* progressing *if each rule in $\mathcal{S}$ is of the form $p(x_1, \dots, x_{\#p}) \Leftarrow \exists z_1 \dots \exists z_m \mathrel{.} \mathsf{root}(p(x_1, \dots, x_{\#p})) \mapsto (t_1, \dots, t_{\mathfrak{K}}) * \psi$ and ψ contains no occurrences of points-to atoms. Moreover, $\mathcal{S}$ is* connected *if $\mathsf{root}(q(u_1, \dots, u_{\#q})) \in \{t_1, \dots, t_{\mathfrak{K}}\} \cup \mathbb{C}$, for each predicate atom $q(u_1, \dots, u_{\#q})$ occurring in ψ. An entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ is progressing (connected) if $\mathcal{S}$ is progressing (connected).*

The progress and connectivity conditions can be checked in polynomial time by a syntactic inspection of the rules in $\mathcal{S}$, even if the $\mathsf{root}(.)$ function is not known a priori. Note that this definition of connectivity is less restrictive that the definition from [8], that asked for $\mathsf{root}(q(u_1, \dots, u_{\#q})) \in \{t_1, \dots, t_{\mathfrak{K}}\}$. For instance, the set of rules $\{\mathsf{p}(x) \Leftarrow \exists y \mathrel{.} x \mapsto y * \mathsf{p}(y) * \mathsf{p}(\mathsf{c}), \mathsf{p}(x) \Leftarrow x \mapsto \mathsf{nil}\}$, where $\mathsf{c} \in \mathbb{C}$ is progressing and connected (with $\mathsf{root}(\mathsf{p}) = 1$) in the sense of Definition 5, but not connected in the sense of [8], because $\mathsf{c} \notin (y)$. Note also that nullary predicate symbols are allowed, for instance $\mathsf{q}() \Leftarrow \mathsf{c} \mapsto \mathsf{nil}$ is progressing and connected (with $\mathsf{root}(\mathsf{q}) = \mathsf{c}$). Further, the entailment problem from Example 4 is both progressing and connected.

Third, the *establishment* condition is defined, slightly extended from its original statement [8]:

▶ **Definition 6** (Establishment). *Given a set of rules $\mathcal{S}$, a symbolic heap $\exists x_1 \dots \exists x_n \mathrel{.} \phi$, where ϕ is quantifier-free, is $\mathcal{S}$-*established *iff every x_i for $i \in [\![1 \mathinner{..} n]\!]$ is allocated in each predicate-free unfolding $\phi \Rightarrow^*_{\mathcal{S}} \varphi$. A set of rules $\mathcal{S}$ is* established *if the body ρ of each rule $p(x_1, \dots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho$ is $\mathcal{S}$-established. An entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ is established if $\mathcal{S}$ is established, and* strongly established *if, moreover, ϕ_i is $\mathcal{S}$-established, for each sequent $\phi_0 \vdash_{\mathcal{P}} \phi_1, \dots, \phi_n$ and each $i \in [\![0 \mathinner{..} n]\!]$.*

For example, the entailment problem from Example 4 is strongly established.

In this paper, we replace establishment with a new condition that, as we show, preserves the decidability and computational complexity of progressing, connected and established entailment problems. The new condition can be checked in time linear in the size of the problem. This condition, called *equational restrictedness* (*e-restrictedness*, for short), requires that each equational atom occurring in a formula involves at least one constant. We will show that the e-restrictedness condition is more general than establishment, in the sense that every established problem can be reduced to an equivalent e-restricted problem (Theorem 13). Moreover, the class of structures defined using e-restricted symbolic heaps is a strict superset of the one defined by established symbolic heaps.

▶ **Definition 7** (E-restrictedness). *A symbolic heap ϕ is* e-restricted *if, for every equational atom $t \bowtie u$ from ϕ, where $\bowtie \in \{\simeq, \not\simeq\}$, we have $\{t, u\} \cap \mathbb{C} \neq \emptyset$. A set of rules $\mathcal{S}$ is* e-restricted *if the body ρ of each rule $p(x_1, \dots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho$ is e-restricted. An entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ is* e-restricted *if $\mathcal{S}$ is e-restricted and ϕ_i is e-restricted, for each sequent $\phi_0 \vdash_{\mathcal{P}} \phi_1, \dots, \phi_n$ and each $i \in [\![0 \mathinner{..} n]\!]$.*

For instance, the entailment problem from Example 4 is not e-restricted, because several rule bodies have disequalities between parameters, e.g. $\mathsf{ls}(x,y) \Leftarrow x \mapsto y * x \not\approx y$. However, the set of rules $\{\mathsf{ls}_\mathsf{c}(x) \Leftarrow x \mapsto c * x \not\approx \mathsf{c}, \mathsf{ls}_\mathsf{c}(x) \Leftarrow \exists y \, . \, x \mapsto y * \mathsf{ls}_\mathsf{c}(y) * x \not\approx \mathsf{c}\}$, where $\mathsf{c} \in \mathbb{C}$ and ls_c is a new predicate symbol, denoting an acyclic list ending with c, is e-restricted. Note that any atom $\mathsf{ls}(x,y)$ can be replaced by $\mathsf{ls}_\mathsf{y}(x)$, provided that y occurs free in a sequent and can be viewed as a constant.

We show next that every established entailment problem (Definition 6) can be reduced to an e-restricted entailment problem (Definition 7). The transformation incurs an exponential blowup, however, as we show, the blowup is exponential only in the width and polynomial in the size of the input problem. This is to be expected, because checking e-restrictedness of a problem can be done in linear time, in contrast with checking establishment, which is at least co-NP-hard [11].

We begin by showing that each problem can be translated into an equivalent *normalized* problem:

▶ **Definition 8** (Normalization).

(1) *A symbolic heap $\exists \mathbf{x} \, . \, \psi \in \mathsf{SH}^\mathfrak{K}$, where ψ is quantifier-free, is* normalized *iff for every atom α in ψ:*

- **a.** *if α is an equational atom, then it is of the form $x \not\approx t$ ($t \not\approx x$), where $x \in \mathbf{x}$,*
- **b.** *every variable $x \in \mathsf{fv}(\psi)$ occurs in a points-to or predicate atom of ψ,*
- **c.** *if α is a predicate atom $q(t_1, \ldots, t_{\#q})$, then $\{t_1, \ldots, t_{\#q}\} \cap \mathbb{C} = \emptyset$ and $t_i \neq t_j$, for all $i \neq j \in [\![1 \, .. \, \#q]\!]$.*

(2) *A set of rules $\mathcal{S}$ is* normalized *iff for each rule $p(x_1, \ldots, x_{\#p}) \Leftarrow_\mathcal{S} \rho$, the symbolic heap ρ is normalized and, moreover:*

- **a.** *For every $i \in [\![1 \, .. \, \#p]\!]$ and every predicate-free unfolding $p(x_1, \ldots, x_{\#p}) \Rightarrow^*_\mathcal{S} \varphi$, φ contains a points-to atom $t_0 \mapsto (t_1, \ldots, t_\mathfrak{K})$, such that $x_i \in \{t_0, \ldots, t_\mathfrak{K}\}$.*
- **b.** *There exist sets $\mathsf{palloc}_\mathcal{S}(p) \subseteq [\![1 \, .. \, \#p]\!]$ and $\mathsf{calloc}_\mathcal{S}(p) \subseteq \mathbb{C}$ such that, for each predicate-free unfolding $p(x_1, \ldots, x_{\#p}) \Rightarrow^*_\mathcal{S} \varphi$:*
  - *$i \in \mathsf{palloc}_\mathcal{S}(p)$ iff φ contains an atom $x_i \mapsto (t_1, \ldots, t_\mathfrak{K})$, for every $i \in [\![1 \, .. \, \#p]\!]$,*
  - *$c \in \mathsf{calloc}_\mathcal{S}(p)$ iff φ contains an atom $c \mapsto (t_1, \ldots, t_\mathfrak{K})$, for every $c \in \mathbb{C}$.*
- **c.** *For every predicate-free unfolding $p(x_1, \ldots, x_{\#p}) \Rightarrow^*_\mathcal{S} \varphi$, if φ contains an atom $t_0 \mapsto (t_1, \ldots, t_\mathfrak{K})$ such that $t_0 \in \mathbb{V} \setminus \{x_1, \ldots, x_{\#p}\}$, then φ also contains atoms $t_0 \not\approx c$, for every $c \in \mathbb{C}$.*

(3) *An entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ is* normalized *if $\mathcal{S}$ is normalized and, for each sequent $\phi_0 \vdash_\mathcal{P} \phi_1, \ldots, \phi_n$ the symbolic heap ϕ_i is normalized, for each $i \in [\![0 \, .. \, n]\!]$.*

The intuition behind Condition (2a) is that no term can "disappear" while unfolding an inductive definition. Condition (2b) states that the set of terms eventually allocated by a predicate atom is the same in all unfoldings. This allows to define the set of symbols that occur freely in a symbolic heap ϕ and are necessarily allocated in every unfolding of ϕ, provided that the set of rules is normalized:

▶ **Definition 9.** *Given a normalized set of rules $\mathcal{S}$ and a symbolic heap $\phi \in \mathsf{SH}^\mathfrak{K}$, the set $\mathsf{alloc}_\mathcal{S}(\phi)$ is defined recursively on the structure of ϕ:*

$$
\begin{aligned}
\mathsf{alloc}_\mathcal{S}(t_0 \mapsto (t_1, \ldots, t_\mathfrak{K})) &\stackrel{\text{def}}{=} \{t_0\} & \mathsf{alloc}_\mathcal{S}(p(t_1, \ldots, t_{\#p})) &\stackrel{\text{def}}{=} \{t_i \mid i \in \mathsf{palloc}_\mathcal{S}(p)\} \\
\mathsf{alloc}_\mathcal{S}(t_1 \bowtie t_2) &\stackrel{\text{def}}{=} \emptyset, \ \bowtie \in \{\approx, \not\approx\} & &\cup \mathsf{calloc}_\mathcal{S}(p) \\
\mathsf{alloc}_\mathcal{S}(\phi_1 * \phi_2) &\stackrel{\text{def}}{=} \mathsf{alloc}_\mathcal{S}(\phi_1) \cup \mathsf{alloc}_\mathcal{S}(\phi_2) & \mathsf{alloc}_\mathcal{S}(\exists x \, . \, \phi_1) &\stackrel{\text{def}}{=} \mathsf{alloc}_\mathcal{S}(\phi_1) \setminus \{x\}
\end{aligned}
$$

▶ **Example 10.** The rules $p(x,y) \Leftarrow \exists z \,.\, x \mapsto z * p(z,y) * x \neq y$ and $p(x,y) \Leftarrow \exists z \,.\, x \mapsto z$ are not normalized, because they contradict Conditions (1a) and (2a) of Definition 8, respectively. A set $\mathcal{S}$ containing the rules $q(x,y) \Leftarrow \exists z \,.\, x \mapsto y * q(y,z)$ and $q(x,y) \Leftarrow x \mapsto y$ is not normalized, because it is not possible to find a set $\mathsf{palloc}_{\mathcal{S}}(q)$ satisfying Condition (2b). Indeed, if $2 \in \mathsf{palloc}_{\mathcal{S}}(q)$ then the required equivalence does not hold for the second rule (because it does not allocate y), and if $2 \not\in \mathsf{palloc}_{\mathcal{S}}(q)$ then it fails for the first one (since the predicate $q(y,z)$ allocates y). On the other hand, $\mathcal{S}' = \{p(x,y) \Leftarrow \exists z \,.\, x \mapsto z * p(z,y) * z \not\approx x * z \not\approx \mathsf{nil}, p(x,y) \Leftarrow x \mapsto y, q(x,y) \Leftarrow \exists z \,.\, x \mapsto y * q(y,z) * z \not\approx \mathsf{nil}$, $q(x,y) \Leftarrow x \mapsto y * r(y), r(x) \Leftarrow x \mapsto \mathsf{nil}\}$ is normalized (assuming $\mathbb{C} = \{\mathsf{nil}\}$), with $\mathsf{palloc}_{\mathcal{S}'}(p) = \mathsf{palloc}_{\mathcal{S}'}(r) = \{1\}$, $\mathsf{palloc}_{\mathcal{S}'}(q) = \{1,2\}$ and $\mathsf{calloc}_{\mathcal{S}'}(\pi) = \emptyset$, for all $\pi \in \{p,q,r\}$. Then $\mathsf{alloc}_{\mathcal{S}'}(p(x_1,x_2) * q(x_3,x_4) * r(x_5)) = \{x_1,x_3,x_4,x_5\}$. ⌟

The following lemma states that every entailment problem can be transformed into a normalized entailment problem, by a transformation that preserves progress, connectivity, e-restricted-ness and (strong) establishment.

▶ **Lemma 11.** *A progressing and connected entailment problem $\mathcal{P}$ can be translated to an equivalent progressing, connected and normalized problem $\mathcal{P}_n$, such that* $\mathrm{width}(\mathcal{P}_n) = \mathcal{O}(\mathrm{width}(\mathcal{P})^2)$ *in time* $\mathrm{size}(\mathcal{P}) \cdot 2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^2)}$. *Further, $\mathcal{P}_n$ is e-restricted if $\mathcal{P}$ is e-restricted and (strongly) established if $\mathcal{P}$ is (strongly) established.*

▶ **Example 12.** The entailment problem $\mathcal{P} = (\mathcal{S},\ \{p(\mathsf{a},\mathsf{b}) \vdash \exists x, y \,.\, q(x,y)\})$ with:

$$\mathcal{S} \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} p(x,y) \Leftarrow \exists z \,.\, x \mapsto z * p(z,y) * x \not\approx y & q(x,y) \Leftarrow \exists z \,.\, x \mapsto y * q(y,z) * z \not\approx \mathsf{a} * z \not\approx \mathsf{b} \\ p(x,y) \Leftarrow \exists z \,.\, x \mapsto z & q(x,y) \Leftarrow x \mapsto y \end{array} \right\}$$

may be transformed into $(\mathcal{S}',\ \{p_1() \vdash \exists x, y \,.\, q_1(x,y), \exists x, y \,.\, q_2(x,y)\})$, where $\mathcal{S}'$ is the set:

$$\begin{array}{ll} p_1() \Leftarrow \exists z \,.\, \mathsf{a} \mapsto z * p_2(z) * z \not\approx \mathsf{a} * z \not\approx \mathsf{b} & p_2(x) \Leftarrow x \mapsto \mathsf{b} * p_3() \\ p_1() \Leftarrow \mathsf{a} \mapsto \mathsf{b} * p_3() & p_2(x) \Leftarrow \exists z \,.\, x \mapsto z * p_2(z) * z \not\approx \mathsf{a} * z \not\approx \mathsf{b} \\ p_1() \Leftarrow \exists z \,.\, \mathsf{a} \mapsto z & p_2(x) \Leftarrow \exists z \,.\, x \mapsto z \\ p_3() \Leftarrow \exists z \,.\, \mathsf{b} \mapsto z & q_1(x,y) \Leftarrow \exists z \,.\, x \mapsto y * q_1(y,z) * z \not\approx \mathsf{a} * z \not\approx \mathsf{b} \\ q_2(x,y) \Leftarrow x \mapsto y & q_1(x,y) \Leftarrow \exists z \,.\, x \mapsto y * q_2(y,z) * z \not\approx \mathsf{a} * z \not\approx \mathsf{b} \end{array}$$

The predicate atoms $p_1(), p_2(x)$ and $p_3()$ are equivalent to $p(\mathsf{a},\mathsf{b})$, $p(x,\mathsf{b})$ and $p(\mathsf{b},\mathsf{b})$, respectively. $q(x,y)$ is equivalent to $q_1(x,y) \vee q_2(x,y)$. Note that $p_2(x)$ is only used in a context where $x \not\approx b$ holds, thus the atom $x \not\approx b$ may be omitted from the rules of $p_2()$. Recall that a and b are mapped to distinct locations, by Assumption 1. ⌟

We show that every established problem $\mathcal{P}$ can be reduced to an e-restricted problem in time linear in the size and exponential in the width of the input, at the cost of a polynomial increase of its width:

▶ **Theorem 13.** *Every progressing, connected and established entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ can be reduced in time* $\mathrm{size}(\mathcal{P}) \cdot 2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^2)}$ *to a normalized, progressing, connected and e-restricted problem $\mathcal{P}_r$, such that* $\mathrm{width}(\mathcal{P}_r) = \mathcal{O}(\mathrm{width}(\mathcal{P}))$.

The class of e-restricted problems is more general than the class of established problems, in the following sense: for each established problem $\mathcal{P} = (\mathcal{S}, \Sigma)$, the treewidth of each $\mathcal{S}$-model of a $\mathcal{S}$-established symbolic heap ϕ is bounded by $\mathrm{width}(\mathcal{P})$ [8], while e-restricted symbolic heaps may have infinite sequences of models with strictly increasing treewidth:

▶ **Example 14.** Consider the set of rules $\{\mathsf{lls}(x,y) \Leftarrow x \mapsto (y,\mathsf{nil}),\ \mathsf{lls}(x,y) \Leftarrow \exists z \exists v \,.\, x \mapsto (z,v) * \mathsf{lls}(z,y)\}$. The existentially quantified variable v in the second rule in never allocated in any predicate-free unfolding of $\mathsf{lls}(\mathsf{a},\mathsf{b})$, thus the set of rules is not established. However, it is trivially e-restricted, because no equational atoms occur within the rules. Among the models of $\mathsf{lls}(\mathsf{a},\mathsf{b})$, there are all $n \times n$-square grid structures, known to have treewidth n, for $n > 1$ [17] (such a grid can be represented as a list of length n^2, with additional links between the elements at positions i and $i+n$). ⌟

4 Normal Structures

The decidability of e-restricted entailment problems relies on the fact that, to prove the validity of a sequent, it is sufficient to consider only a certain class of structures, called *normal*, that require the variables not mapped to the same location as a constant to be mapped to pairwise distinct locations:

▶ **Definition 15.** *A structure $(\mathfrak{s},\mathfrak{h})$ is a* normal $\mathcal{S}$-model *of a symbolic heap ϕ iff there exists:*

1. *a predicate-free unfolding $\phi \Rightarrow_{\mathcal{S}} \exists \mathbf{x} \ . \ \psi$, where ψ is quantifier-free, and*
2. *an $\mathbf{x}$-associate $\overline{\mathfrak{s}}$ of $\mathfrak{s}$, such that $(\overline{\mathfrak{s}},\mathfrak{h}) \models_{\mathcal{S}} \psi$ and $\overline{\mathfrak{s}}(x) = \overline{\mathfrak{s}}(y) \wedge x \neq y \Rightarrow \overline{\mathfrak{s}}(x) \in \mathfrak{s}(\mathbb{C})$, for all $x, y \in \mathsf{fv}(\psi)$.*

▶ **Example 16.** Consider the formula $\varphi = p(x_1) * p(x_2)$, with $p(x) \Leftarrow_{\mathcal{S}} \exists z \ . \ x \mapsto z$ and $\mathbb{C} = \{\mathsf{a}\}$. Then the structures: $(\mathfrak{s},\mathfrak{h})$ and $(\mathfrak{s},\mathfrak{h}')$ with $\mathfrak{s} = \{(x_1,\ell_1),(x_2,\ell_2),(\mathsf{a},\ell_3)\}$, $\mathfrak{h} = \{(\ell_1,\ell_3),(\ell_2,\ell_3)\}$ and $\mathfrak{h}' = \{(\ell_1,\ell_4),(\ell_2,\ell_5)\}$ are normal models of φ. On the other hand, if $\mathfrak{h}'' = \{(\ell_1,\ell_4),(\ell_2,\ell_4)\}$ (with $\ell_4 \neq \ell_3$) then $(\mathfrak{s},\mathfrak{h}'')$ is a model of φ but it is not normal, because any associate of $\mathfrak{s}$ will map the existentials from the predicate-free unfolding of $p(x_1) * p(x_2)$ into the same location, different from $\mathfrak{s}(\mathsf{a})$. ⌟

Since the left-hand side symbolic heap ϕ of each sequent $\phi \vdash \psi_1, \ldots, \psi_n$ is quantifier-free and has no free variables (Definition 3) and moreover, by Assumption 1, every constant is associated a distinct location, to check the validity of a sequent it is enough to consider only structures with injective stores. We say that a structure $(\dot{\mathfrak{s}},\mathfrak{h})$ is *injective* if the store $\dot{\mathfrak{s}}$ is injective. As a syntactic convention, by stacking a dot on the symbol denoting the store, we mean that the store is injective.

The key property of normal structures is that validity of e-restricted entailment problems can be checked considering only (injective) normal structures. The intuition is that, since the (dis-)equalities occurring in the considered formula involve a constant, it is sufficient to assume that all the existential variables not equal to a constant are mapped to pairwise distinct locations, as all other structures can be obtained from such structures by applying a morphism that preserves the truth value of the considered formulæ.

▶ **Lemma 17.** *Let $\mathcal{P} = (\mathcal{S},\Sigma)$ be a normalized and e-restricted entailment problem and let $\phi \vdash_{\mathcal{P}} \psi_1, \ldots, \psi_n$ be a sequent. Then $\phi \vdash_{\mathcal{P}} \psi_1, \ldots, \psi_n$ is valid for $\mathcal{S}$ iff $(\dot{\mathfrak{s}},\mathfrak{h}) \models_{\mathcal{S}} \bigvee_{i=1}^{n} \psi_i$, for each normal injective $\mathcal{S}$-model $(\dot{\mathfrak{s}},\mathfrak{h})$ of ϕ.*

5 Core Formulæ

Given an e-restricted entailment problem $\mathcal{P} = (\mathcal{S},\Sigma)$, the idea of the entailment checking algorithm is to compute, for each symbolic heap ϕ that occurs as the left-hand side of a sequent $\phi \vdash_{\mathcal{P}} \psi_1, \ldots, \psi_n$, a finite set of sets of formulæ $\mathcal{F}(\phi) = \{F_1, \ldots, F_m\}$, of some specific pattern, called *core formulæ*. The set $\mathcal{F}(\phi)$ defines an equivalence relation, of finite index, on the set of injective normal $\mathcal{S}$-models of ϕ, such that each set $F \in \mathcal{F}(\phi)$ encodes an equivalence class. Because the validity of each sequent can be checked by testing whether every (injective) normal model of its left-hand side is a model of some symbolic heap on the right-hand side (Lemma 17), an equivalent check is that each set $F \in \mathcal{F}(\phi)$ contains a core formula entailing some formula ψ_i, for $i = 1, \ldots, n$. To improve the presentation, we first formalize the notions of core formulæ and abstractions by sets of core formulæ, while deferring the effective construction of $\mathcal{F}(\phi)$, for a symbolic heap ϕ, to the next section (§6). In the following, we refer to a given entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$.

First, we define core formulæ as a fragment of $\mathsf{SL}^{\mathfrak{K}}$. Consider a formula $\mathsf{loc}(x) \stackrel{\mathsf{def}}{=} \exists y_0 \ldots \exists y_{\mathfrak{K}} \;.\; y_0 \mapsto (y_1, \ldots, y_{\mathfrak{K}}) * \bigvee_{i=0}^{\mathfrak{K}} x \approx y_i$. Note that a structure is a model of $\mathsf{loc}(x)$ iff the variable x is mapped to a location from the domain or the range of the heap. We define also the following bounded quantifiers:

$$\dot{\exists} x \;.\; \phi \stackrel{\mathsf{def}}{=} \exists x \;.\; \bigwedge_{t \in (\mathsf{fv}(\phi) \setminus \{x\}) \cup \mathbb{C}} \neg x \approx t \wedge \phi \qquad\qquad \exists_{\mathsf{h}} x \;.\; \phi \stackrel{\mathsf{def}}{=} \dot{\exists} x \;.\; \mathsf{loc}(x) \wedge \phi$$

$$\exists_{\neg\mathsf{h}} x \;.\; \phi \stackrel{\mathsf{def}}{=} \dot{\exists} x \;.\; \neg\mathsf{loc}(x) \wedge \phi \qquad\qquad \forall_{\neg\mathsf{h}} x \;.\; \phi \stackrel{\mathsf{def}}{=} \neg\exists_{\neg\mathsf{h}} x \;.\; \neg\phi$$

In the following, we shall be extensively using the $\exists_{\mathsf{h}} x \;.\; \phi$ and $\forall_{\neg\mathsf{h}} x \;.\; \phi$ quantifiers. The formula $\exists_{\mathsf{h}} x \;.\; \phi$ states that there exists a location ℓ which occurs in the domain or range of the heap and is distinct from the locations associated with the constants and free variables, such that ϕ holds when x is associated with ℓ. Similarly, $\forall_{\neg\mathsf{h}} x \;.\; \phi$ states that ϕ holds if x is associated with any location ℓ that is outside of the heap and distinct from all the constants and free variables. The use of these special quantifiers will allow us to restrict ourselves to injective stores (since all variables and constants are mapped to distinct locations), which greatly simplifies the handling of equalities.

The main ingredient used to define core formulæ are *context predicates*. Given a tuple of predicate symbols $(p, q_1, \ldots, q_n) \in \mathbb{P}^{n+1}$, where $n \geq 0$, we consider a context predicate symbol $\Gamma_{p,q_1,\ldots,q_n}$ of arity $\#p + \sum_{i=1}^{n} \#q_i$. The informal intuition of a context predicate atom $\Gamma_{p,q_1,\ldots,q_n}(\mathbf{t}, \mathbf{u}_1, \ldots, \mathbf{u}_n)$ is the following: a structure $(\mathfrak{s}, \mathfrak{h})$ is a model of this atom if there exist models $(\mathfrak{s}, \mathfrak{h}_i)$ of $q_i(\mathbf{u}_i)$, $i \in [\![1 \;..\; n]\!]$ respectively, with mutually disjoint heaps, an unfolding ψ of $p(\mathbf{t})$ in which the atoms $q_i(\mathbf{u}_i)$ occur, and an associate $\mathfrak{s}'$ of $\mathfrak{s}$ such that $(\mathfrak{s}', \mathfrak{h} \uplus \biguplus_{i=1}^{n} \mathfrak{h}_i)$ is a model of ψ.

For readability's sake, we adopt a notation close in spirit to SL's separating implication (known as the magic wand), and we write $\mathop{\Large\ast}_{i=1}^{n} q_i(\mathbf{y}_i) \mathrel{-\!\!\bullet} p(\mathbf{x})$ for $\Gamma_{p,q_1,\ldots,q_n}(\mathbf{x}, \mathbf{y}_1, \ldots, \mathbf{y}_n)$ and $\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{x})$, when $n = 0$[3]. The set of rules defining the interpretation of context predicates is the least set defined by the inference rules below, denoted $\mathfrak{C}_{\mathcal{S}}$:

$$\frac{}{p(\mathbf{x}) \mathrel{-\!\!\bullet} p(\mathbf{y}) \Leftarrow_{\mathfrak{C}_{\mathcal{S}}} \mathbf{x} \hat{=} \mathbf{y}} \; \mathbf{x} \cap \mathbf{y} = \emptyset \tag{I}$$

$$\frac{p(\mathbf{x}) \Leftarrow_{\mathcal{S}} \exists \mathbf{z} \;.\; \psi * \mathop{\Large\ast}_{j=1}^{m} p_j(\mathbf{w}_j) \qquad \mathop{\Large\ast}_{i=1}^{n} q_i(\mathbf{y}_i) = \mathop{\Large\ast}_{j=1}^{m} \gamma_j}{\mathop{\Large\ast}_{i=1}^{n} q_i(\mathbf{y}_i) \mathrel{-\!\!\bullet} p(\mathbf{x}) \Leftarrow_{\mathfrak{C}_{\mathcal{S}}} \exists \mathbf{v} \;.\; \psi\sigma * \mathop{\Large\ast}_{j=1}^{m} (\gamma_j \mathrel{-\!\!\bullet} p_j(\sigma(\mathbf{w}_j)))} \; \begin{array}{l} \mathbf{x}, \mathbf{z}, \mathbf{y}_1, \ldots, \mathbf{y}_n \text{ pairwise disjoint} \\ \sigma : \mathbf{z} \rightharpoonup \mathbf{x} \cup \bigcup_{i=1}^{n} \mathbf{y}_i \\ \mathbf{v} = \mathbf{z} \setminus \mathrm{dom}(\sigma) \end{array} \tag{II}$$

Note that $\mathfrak{C}_{\mathcal{S}}$ is not progressing, since the rule for $p(\mathbf{x}) \mathrel{-\!\!\bullet} p(\mathbf{y})$ does not allocate any location. However, if $\mathcal{S}$ is progressing, then the set of rules obtained by applying (II) only is

[3] Context predicates are similar to the *strong magic wand* introduced in [13]. A context predicate $\alpha \mathrel{-\!\!\bullet} \beta$ is also related to the usual separating implication $\alpha \mathrel{-\!\!*} \beta$ of separation logic, but it is not equivalent. Intuitively, $\mathrel{-\!\!*}$ represents a difference between two heaps, whereas $\mathrel{-\!\!\bullet}$ removes some atoms in an unfolding. For instance, if p and q are defined by the same inductive rules, up to a renaming of predicates, then $p(x) \mathrel{-\!\!*} q(x)$ always holds in a structure with an empty heap, whereas $p(x) \mathrel{-\!\!\bullet} q(x)$ holds if, moreover, $p(x)$ and $q(x)$ are the same atom.

also progressing. Rule (I) says that each predicate atom $p(\mathbf{t}) \mathrel{-\!\!\bullet} p(\mathbf{u})$, such that $\mathbf{t}$ and $\mathbf{u}$ are mapped to the same tuple of locations, is satisfied by the empty heap. To understand rule (II), let $(\mathfrak{s}, \mathfrak{h})$ be an $\mathcal{S}$-model of $p(\mathbf{t})$ and assume there are a predicate-free unfolding ψ of $p(\mathbf{t})$ and an associate $\mathfrak{s}'$ of $\mathfrak{s}$, such that $q_1(\mathbf{u}_1), \ldots, q_n(\mathbf{u}_n)$ occur in ψ and $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{S}} \psi$. If the first unfolding step is an instance of a rule $p(\mathbf{x}) \Leftarrow_{\mathcal{S}} \exists \mathbf{z} \,.\, \psi * \mathop{\ast}_{j=1}^{m} p_j(\mathbf{w}_j)$ then there exist a $\mathbf{z}$-associate $\overline{\mathfrak{s}}$ of $\mathfrak{s}$ and a split of $\mathfrak{h}$ into disjoint heaps $\mathfrak{h}_0, \ldots, \mathfrak{h}_m$ such that $(\overline{\mathfrak{s}}, \mathfrak{h}_0) \models \psi[\mathbf{t}/\mathbf{x}]$ and $(\overline{\mathfrak{s}}, \mathfrak{h}_j) \models_{\mathcal{S}} p_j(\mathbf{w}_j)[\mathbf{t}/\mathbf{x}]$, for all $j \in \llbracket 1 \,..\, m \rrbracket$. Assume, for simplicity, that $\mathbf{u}_1 \cup \ldots \cup \mathbf{u}_n \subseteq \mathrm{dom}(\overline{\mathfrak{s}})$ and let $\overline{\mathfrak{h}}_1, \ldots, \overline{\mathfrak{h}}_n$ be disjoint heaps such that $(\overline{\mathfrak{s}}, \overline{\mathfrak{h}}_i) \models_{\mathcal{S}} q_i(\mathbf{u}_i)$. Then there exists a partition $\big\{\{i_{j,1}, \ldots, i_{j,k_j}\} \mid j \in \llbracket 1 \,..\, m \rrbracket\big\}$ of $\llbracket 1 \,..\, n \rrbracket$, such that $\overline{\mathfrak{h}}_{i_{j,1}}, \ldots, \overline{\mathfrak{h}}_{i_{j,k_j}} \subseteq \mathfrak{h}_j$, for all $j \in \llbracket 1 \,..\, m \rrbracket$. Let $\gamma_j \overset{\mathsf{def}}{=} \mathop{\ast}_{\ell=1}^{k_j} q_\ell(\mathbf{u}_\ell)$, then $(\overline{\mathfrak{s}}, \mathfrak{h}_j \setminus (\overline{\mathfrak{h}}_{i_{j,1}} \cup \ldots \cup \overline{\mathfrak{h}}_{i_{j,k_j}})) \models_{\mathfrak{C}_{\mathcal{S}}} \gamma_j \mathrel{-\!\!\bullet} p_j(\mathbf{w}_j)[\mathbf{t}/\mathbf{x}]$, for each $j \in \llbracket 1 \,..\, m \rrbracket$. This observation leads to the inductive definition of the semantics for $\mathop{\ast}_{i=1}^{n} q_i(\mathbf{u}_i) \mathrel{-\!\!\bullet} p(\mathbf{t})$, by the rule that occurs in the conclusion of (II), where the substitution $\sigma : \mathbf{z} \rightharpoonup \mathbf{x} \cup \bigcup_{i=1}^{n} \mathbf{y}_i$ is used to instantiate[4] some of the existentially quantified variables from the original rule $p(\mathbf{x}) \Leftarrow_{\mathcal{S}} \exists \mathbf{z} \,.\, \psi * \mathop{\ast}_{j=1}^{m} p_j(\mathbf{w}_j)$.

▶ **Example 18.** Consider the set $\mathcal{S} = \{p(x) \Leftarrow \exists z_1, z_2 \,.\, x \mapsto (z_1, z_2) * q(z_1) * q(z_2), q(x) \Leftarrow x \mapsto (x, x)\}$. We have $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{S}} p(x)$ with $\mathfrak{s} = \{(x, \ell_1)\}$ and $\mathfrak{h} = \{(\ell_1, \ell_2, \ell_3), (\ell_2, \ell_2, \ell_2), (\ell_3, \ell_3, \ell_3)\}$. The atom $q(y) \mathrel{-\!\!\bullet} p(x)$ is defined by the following non-progressing rules (we only consider the rules corresponding to the case where σ is the identity, since the other rules are redundant):

$$\begin{array}{ll} q(y) \mathrel{-\!\!\bullet} p(x) \Leftarrow \exists z_1, z_2 \,.\, x \mapsto (z_1, z_2) * q(y) \mathrel{-\!\!\bullet} q(z_1) * \mathsf{emp} \mathrel{-\!\!\bullet} q(z_2) & q(y) \mathrel{-\!\!\bullet} q(x) \Leftarrow x \doteq y \\ q(y) \mathrel{-\!\!\bullet} p(x) \Leftarrow \exists z_1, z_2 \,.\, x \mapsto (z_1, z_2) * \mathsf{emp} \mathrel{-\!\!\bullet} q(z_1) * q(y) \mathrel{-\!\!\bullet} q(z_2) & \mathsf{emp} \mathrel{-\!\!\bullet} q(x) \Leftarrow x \mapsto (x, x) \end{array}$$

The two rules for $q(y) \mathrel{-\!\!\bullet} p(x)$ correspond to the two ways of distributing $q(y)$ over $q(z_1)$, $q(z_2)$. We have $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, with $\mathfrak{h}_1 = \{(\ell_1, \ell_2, \ell_3), (\ell_2, \ell_2, \ell_2)\}$ and $\mathfrak{h}_2 = \{(\ell_3, \ell_3, \ell_3)\}$. It is easy to check that $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}_1) \models_{\mathfrak{C}_{\mathcal{S}}} q(y) \mathrel{-\!\!\bullet} p(x)$, and $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}_2) \models_{\mathfrak{C}_{\mathcal{S}}} q(y)$. Note that we also have $(\mathfrak{s}[y \leftarrow \ell_2], \mathfrak{h}'_1) \models_{\mathfrak{C}_{\mathcal{S}}} q(y) \mathrel{-\!\!\bullet} p(x)$, with $\mathfrak{h}'_1 = \{(\ell_1, \ell_2, \ell_3), (\ell_3, \ell_3, \ell_3)\}$. ⌟

Having introduced context predicates, the pattern of core formulæ is defined below:

▶ **Definition 19.** *A* core formula φ *is an instance of the pattern:*

$$\exists_{\mathsf{h}} \mathbf{x} \forall_{\neg \mathsf{h}} \mathbf{y} \,.\, \mathop{\ast}_{i=1}^{n} \Big(\mathop{\ast}_{j=1}^{k_i} q^i_j(\mathbf{u}^i_j) \mathrel{-\!\!\bullet} p_i(\mathbf{t}_i)\Big) * \mathop{\ast}_{i=n+1}^{m} t^i_0 \mapsto (t^i_1, \ldots, t^i_{\mathfrak{K}}) \quad \textit{such that:}$$

(i) *each variable occurring in* $\mathbf{y}$ *also occurs in an atom in* φ*;*

(ii) *for every variable* $x \in \mathbf{x}$*, either* $x \in \mathbf{t}_i \setminus \bigcup_{i=1}^{k_i} \mathbf{u}^i_j$ *for some* $i \in \llbracket 1 \,..\, n \rrbracket$*, or* $x = t^i_j$*, for some* $i \in \llbracket n+1 \,..\, m \rrbracket$ *and some* $j \in \llbracket 0 \,..\, \mathfrak{K} \rrbracket$*;*

(iii) *each term* t *occurs at most once as* $t = \mathsf{root}(\alpha)$*, where* α *is an atom of* φ*.*

We also define the set of terms $\mathrm{roots}(\varphi) \overset{\mathsf{def}}{=} \mathrm{roots}_{\mathsf{lhs}}(\varphi) \cup \mathrm{roots}_{\mathsf{rhs}}(\varphi)$*, with* $\mathrm{roots}_{\mathsf{lhs}}(\varphi) \overset{\mathsf{def}}{=} \{\mathsf{root}(q^i_j(\mathbf{u}^i_j)) \mid i \in \llbracket 1 \,..\, n \rrbracket, j \in \llbracket 1 \,..\, k_i \rrbracket\}$ *and* $\mathrm{roots}_{\mathsf{rhs}}(\varphi) \overset{\mathsf{def}}{=} \{\mathsf{root}(p_i(\mathbf{t}_i)) \mid i \in \llbracket 1 \,..\, n \rrbracket\} \cup \{t^i_0 \mid i \in \llbracket n+1 \,..\, m \rrbracket\}$*.*

Note that an unfolding of a core formula using the rules in $\mathfrak{C}_{\mathcal{S}}$ is not necessarily a core formula, because of the unbounded existential quantifiers and equational atoms that occur in the rules from $\mathfrak{C}_{\mathcal{S}}$. Note also that a core formula cannot contain an occurrence of a predicate of the form $p(\mathbf{t}) \mathrel{-\!\!\bullet} p(\mathbf{t})$ because otherwise, Condition (iii) of Definition 19 would be violated.

Lemma 20 shows that any symbolic heap is equivalent to an effectively computable finite disjunction of core formulæ, when the interpretation of formulæ is restricted to injective

[4] Note that this instantiation is, in principle, redundant (i.e. the same rules are obtained if $\mathrm{dom}(\sigma) = \emptyset$ by chosing appropriate $\mathbf{z}$-associates) but we keep it to simplify the related proofs.

structures. For a symbolic heap $\phi \in \mathsf{SH}^{\mathfrak{K}}$, we define the set $\mathcal{T}(\phi)$, recursively on the structure of ϕ, implicitly assuming w.l.o.g. that $\mathsf{emp} * \phi = \phi * \mathsf{emp} = \phi$:

$$\begin{array}{ll}
\mathcal{T}(\mathsf{emp}) \stackrel{\mathsf{def}}{=} \{\mathsf{emp}\} & \mathcal{T}(t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})) \stackrel{\mathsf{def}}{=} \{t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})\} \\
\mathcal{T}(p(\mathbf{t})) \stackrel{\mathsf{def}}{=} \{\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t})\} & \mathcal{T}(\phi_1 * \phi_2) \stackrel{\mathsf{def}}{=} \{\psi_1 * \psi_2 \mid \psi_i \in \mathcal{T}(\phi_i),\ i = 1, 2\} \\
\mathcal{T}(t_1 \simeq t_2) \stackrel{\mathsf{def}}{=} \left\{ \begin{array}{ll} \{\mathsf{emp}\} & \text{if } t_1 = t_2 \\ \emptyset & \text{if } t_1 \neq t_2 \end{array} \right. & \mathcal{T}(t_1 \not\simeq t_2) \stackrel{\mathsf{def}}{=} \left\{ \begin{array}{ll} \emptyset & \text{if } t_1 = t_2 \\ \{\mathsf{emp}\} & \text{if } t_1 \neq t_2 \end{array} \right. \\
\multicolumn{2}{l}{\mathcal{T}(\exists x \,.\, \phi_1) \stackrel{\mathsf{def}}{=} \{\exists_{\mathsf{h}} x \,.\, \psi \mid \psi \in \mathcal{T}(\phi_1)\} \cup \{\psi \mid \psi \in \mathcal{T}(\phi_1[t/x]),\quad t \in (\mathsf{fv}(\phi_1) \setminus \{x\}) \cup \mathbb{C}\}}
\end{array}$$

For instance, if $\phi = \exists x \,.\, p(x, y) * x \not\simeq y$ and $\mathbb{C} = \{\mathsf{c}\}$, then $\mathcal{T}(\phi) = \{\exists_{\mathsf{h}} x \,.\, \mathsf{emp} \mathrel{-\!\!\bullet} p(x, y),\ \mathsf{emp} \mathrel{-\!\!\bullet} p(\mathsf{c}, y)\}$. Note that $\mathcal{T}(y \not\simeq y) = \emptyset$, thus $\mathsf{emp} \mathrel{-\!\!\bullet} p(y, y) \notin \mathcal{T}(\phi)$.

▶ **Lemma 20.** *Assume $\mathcal{S}$ is normalized. Consider an e-restricted normalized symbolic heap $\phi \in \mathsf{SH}^{\mathfrak{K}}$ with no occurrences of context predicate symbols, and an injective structure $(\dot{\mathfrak{s}}, \mathfrak{h})$, such that $\mathrm{dom}(\dot{\mathfrak{s}}) = \mathsf{fv}(\phi) \cup \mathbb{C}$. We have $(\dot{\mathfrak{s}}, \mathfrak{h}) \models_{\mathcal{S}} \phi$ iff $(\dot{\mathfrak{s}}, \mathfrak{h}) \models_{\mathfrak{C}_{\mathcal{S}}} \psi$, for some $\psi \in \mathcal{T}(\phi)$.*

Next, we give an equivalent condition for the satisfaction of a context predicate atom, that relies on an unfolding of a symbolic heap into a core formula:

▶ **Definition 21.** *A formula φ is a* core unfolding *of a predicate atom $\mathop{\Large *}_{i=1}^{n} q_i(\mathbf{u}_i) \mathrel{-\!\!\bullet} p(\mathbf{t})$, written $\mathop{\Large *}_{i=1}^{n} q_i(\mathbf{u}_i) \mathrel{-\!\!\bullet} p(\mathbf{t}) \leadsto_{\mathfrak{C}_{\mathcal{S}}} \varphi$, iff there exists:*

1. *a rule $\mathop{\Large *}_{i=1}^{n} q_i(\mathbf{y}_i) \mathrel{-\!\!\bullet} p(\mathbf{x}) \Leftarrow_{\mathfrak{C}_{\mathcal{S}}} \exists \mathbf{z} \,.\, \phi$, where ϕ is quantifier free, and*
2. *a substitution $\sigma = [\mathbf{t}/\mathbf{x}, \mathbf{u}_1/\mathbf{y}_1, \ldots, \mathbf{u}_n/\mathbf{y}_n] \cup \zeta$, $\zeta \subseteq \{(z, t) \mid z \in \mathbf{z},\ t \in \mathbf{t} \cup \bigcup_{i=1}^{n} \mathbf{u}_i\}$, such that $\varphi \in \mathcal{T}(\phi\sigma)$.*

A core unfolding of a predicate atom is always a quantifier-free formula, obtained from the translation (into a disjunctive set of core formulæ) of the quantifier-free matrix of the body of a rule, in which some of the existentially quantified variables in the rule occur instantiated by the substitution σ. For instance, the rule $\mathsf{emp} \mathrel{-\!\!\bullet} p(x) \Leftarrow_{\mathfrak{C}_{\mathcal{S}}} \exists y \,.\, x \mapsto y$ induces the core unfoldings $\mathsf{emp} \mathrel{-\!\!\bullet} p(a) \leadsto_{\mathcal{S}} a \mapsto a$ and $\mathsf{emp} \mathrel{-\!\!\bullet} p(a) \leadsto_{\mathcal{S}} a \mapsto u$, via the substitutions $[a/x, a/y]$ and $[a/x, u/y]$, respectively. Note that a core unfolding of an atom ϕ may contain variables not occurring in ϕ, corresponding to the existential variables occurring in the rules, such as the variable u in the previous example.

We now define an equivalence relation, of finite index, on the set of injective structures. Intuitively, an equivalence class is defined by the set of core formulæ that are satisfied by all structures in the class (with some additional conditions). First, we introduce the overall set of core formulæ, over which these equivalence classes are defined:

▶ **Definition 22.** *Let $\mathcal{V}_{\mathcal{P}} \stackrel{\mathsf{def}}{=} \mathcal{V}_{\mathcal{P}}^1 \cup \mathcal{V}_{\mathcal{P}}^2$, such that $\mathcal{V}_{\mathcal{P}}^1 \cap \mathcal{V}_{\mathcal{P}}^2 = \emptyset$ and $||\mathcal{V}_{\mathcal{P}}^i|| = \mathrm{width}(\mathcal{P})$, for $i = 1, 2$ and denote by $\mathsf{Core}(\mathcal{P})$ the set of core formulæ φ such that $\mathrm{roots}(\varphi) \cap \mathsf{fv}(\varphi) \subseteq \mathcal{V}_{\mathcal{P}}^1$, $\mathrm{roots}(\varphi) \setminus \mathsf{fv}(\varphi) \subseteq \mathcal{V}_{\mathcal{P}}^2 \cup \mathbb{C}$ and no variable in $\mathcal{V}_{\mathcal{P}}^1$ is bound in φ.*

Note that $\mathsf{Core}(\mathcal{P})$ is a finite set, because both $\mathcal{V}_{\mathcal{P}}$ and $\mathbb{C}$ are finite. Intuitively, $\mathcal{V}_{\mathcal{P}}^1$ will denote "local" variables introduced by unfolding the definitions on the left-hand sides of the entailments, whereas $\mathcal{V}_{\mathcal{P}}^2$ will denote existential variables occurring on the right-hand sides. The sets $\mathcal{V}_{\mathcal{P}}^1$ and $\mathcal{V}_{\mathcal{P}}^2$ can be chosen arbitrarily, provided the conditions of Definition 22 are satisfied. Second, we characterize an injective structure by the set of core formulæ it satisfies:

▶ **Definition 23.** *For a core formula $\varphi = \exists_{\mathsf{h}} \mathbf{x} \forall_{\neg\mathsf{h}} \mathbf{y} \,.\, \psi$, we denote by $\mathcal{W}_{\mathcal{S}}(\dot{\mathfrak{s}}, \mathfrak{h}, \varphi)$ the set of stores $\dot{\overline{\mathfrak{s}}}$ that are injective $(\mathbf{x} \cup \mathbf{y})$-associates of $\dot{\mathfrak{s}}$, and such that:*

(1) *$(\dot{\overline{\mathfrak{s}}}, \mathfrak{h}) \models_{\mathfrak{C}_{\mathcal{S}}} \psi$,*

(2) *$\dot{\overline{\mathfrak{s}}}(\mathbf{x}) \subseteq \mathrm{loc}(\mathfrak{h})$, and*

(3) *$\dot{\overline{\mathfrak{s}}}(\mathbf{y}) \cap \mathrm{loc}(\mathfrak{h}) = \emptyset$.*

The elements of this set are called witnesses for $(\dot{\mathfrak{s}}, \mathfrak{h})$ and φ.

The core abstraction *of an injective structure* $(\dot{\mathfrak{s}}, \mathfrak{h})$ *is the set* $\mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$ *of core formulæ* $\varphi \in \mathsf{Core}(\mathcal{P})$ *for which there exists a witness* $\dot{\bar{\mathfrak{s}}} \in \mathcal{W}_{\mathcal{S}}(\dot{\mathfrak{s}}, \mathfrak{h}, \varphi)$ *such that* $\dot{\bar{\mathfrak{s}}}(\mathrm{roots}_{\mathsf{lhs}}(\varphi)) \cap \mathrm{dom}(\mathfrak{h}) = \emptyset$.

An injective structure $(\dot{\mathfrak{s}}, \mathfrak{h})$ satisfies each core formula $\varphi \in \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$, a fact that is witnessed by an extension of the store assigning the universally quantified variables random locations outside of the heap. Further, any core formula φ such that $(\dot{\mathfrak{s}}, \mathfrak{h}) \models \varphi$ and $\mathrm{roots}_{\mathsf{lhs}}(\varphi) = \emptyset$ occurs in $\mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$.

Our entailment checking algorithm relies on the definition of the *profile* of a symbolic heap. Since each symbolic heap is equivalent to a finite disjunction of existential core formulæ, when interpreted over injective normal structures, it is sufficient to consider only profiles of core formulæ:

▶ **Definition 24.** *The* profile *of an entailment problem* $\mathcal{P} = (\mathcal{S}, \Sigma)$ *is the relation* $\mathcal{F} \subseteq \mathsf{Core}(\mathcal{P}) \times 2^{\mathsf{Core}(\mathcal{P})}$ *such that, for any core formula* $\phi \in \mathsf{Core}(\mathcal{P})$ *and any set of core formulæ* $F \in 2^{\mathsf{Core}(\mathcal{P})}$*, we have* $(\phi, F) \in \mathcal{F}$ *iff* $F = \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$*, for some injective normal* $\mathfrak{C}_{\mathcal{S}}$*-model* $(\dot{\mathfrak{s}}, \mathfrak{h})$ *of* ϕ*, with* $\mathrm{dom}(\dot{\mathfrak{s}}) = \mathsf{fv}(\phi) \cup \mathbb{C}$.

Assuming the existence of a profile, the effective construction of which will be given in Section 6, the following lemma provides an algorithm that decides the validity of $\mathcal{P}$:

▶ **Lemma 25.** *Let* $\mathcal{P} = (\mathcal{S}, \Sigma)$ *be a normalized e-restricted entailment problem and* $\mathcal{F} \subseteq \mathsf{Core}(\mathcal{P}) \times 2^{\mathsf{Core}(\mathcal{P})}$ *be a profile for* $\mathcal{P}$*. Then* $\mathcal{P}$ *is valid iff, for each sequent* $\phi \vdash_{\mathcal{P}} \psi_1, \ldots, \psi_n$*, each core formula* $\varphi \in \mathcal{T}(\phi)$ *and each pair* $(\varphi, F) \in \mathcal{F}$*, we have* $F \cap \mathcal{T}(\psi_i) \neq \emptyset$*, for some* $i \in [\![1 \mathinner{..} n]\!]$.

The proof relies on Lemma 17, according to which entailments can be tested by considering only normal models. As one expects, Lemma 20 is used in this proof to ensure that the translation $\mathcal{T}(.)$ of symbolic heaps into core formulæ preserves the injective models.

6 Construction of the Profile Relation

For a given normalized entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$, we describe the construction of a profile $\mathcal{F}_{\mathcal{P}} \subseteq \mathsf{Core}(\mathcal{P}) \times 2^{\mathsf{Core}(\mathcal{P})}$, recursively on the structure of core formulæ. We assume that the set of rules $\mathcal{S}$ is progressing, connected and e-restricted. The relation $\mathcal{F}_{\mathcal{P}}$ is the least set satisfying the recursive constraints (1), (2), (3) and (4), given in this section. Since these recursive definitions are monotonic, the least fixed point exists and is unique.

Points-to Atoms. For a points-to atom $t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})$, with $t_0, \ldots, t_{\mathfrak{K}} \in \mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C}$, we have:

$$\begin{gathered}(t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}),\ F) \in \mathcal{F}_{\mathcal{P}}, \text{ iff } F \text{ is the set containing } t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}) \text{ and all core formulæ} \\ \text{of the form } \forall_{\neg\mathsf{h}} \mathbf{z} \mathrel{.} \mathop{\huge\ast}_{i=1}^{n} q_i(\mathbf{u}_i) \multimapdot p(\mathbf{t}) \in \mathsf{Core}(\mathcal{P}),\ \text{ where } \mathbf{z} = (\mathbf{t} \cup \mathbf{u}_1 \cup \ldots \cup \mathbf{u}_n) \setminus (\{t_0, \ldots, t_{\mathfrak{K}}\} \cup \mathbb{C}) \\ \text{such that } \mathsf{emp} \multimapdot p(\mathbf{t}) \leadsto_{\mathfrak{C}_{\mathcal{S}}} t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}) * \mathop{\huge\ast}_{i=1}^{n} \mathsf{emp} \multimapdot q_i(\mathbf{u}_i)\end{gathered} \tag{1}$$

For instance, if $\mathcal{S} = \{p(x) \Leftarrow \exists y, z \mathrel{.} x \mapsto y * q(y, z),\ q(x, y) \Leftarrow x \mapsto y\}$, with $\mathcal{V}^1_{\mathcal{P}} = \{u, v\}$ and $\mathcal{V}^2_{\mathcal{P}} = \{z\}$, then $\mathcal{F}_{\mathcal{P}}$ contains the pair $(u \mapsto v, F)$ with $F = \{u \mapsto v, \mathsf{emp} \multimapdot q(u, v), \forall_{\neg\mathsf{h}} z \mathrel{.} q(v, z) \multimapdot p(u)\}$.

Predicate Atoms. Since profiles involve only the core formulæ obtained by the syntactic translation of a symbolic heap, the only predicate atoms that occur in the argument of a profile are of the form $\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t})$. We consider the constraint:

$$(\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t}),\ F) \in \mathcal{F}_{\mathcal{P}} \text{ if } (\exists_{\mathsf{h}} \mathbf{y} \,.\, \psi,\ F) \in \mathcal{F}_{\mathcal{P}}, \mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t}) \leadsto_{\mathfrak{C}_{\mathcal{S}}} \psi \in \mathsf{Core}(\mathcal{P}) \text{ and } \mathbf{y} = \mathsf{fv}(\psi) \backslash \mathbf{t} \quad (2)$$

Separating Conjunctions. Computing the profile of a separating conjunction is the most technical point of the construction. To ease the presentation, we assume the existence of a binary operation called *composition*:

▶ **Definition 26.** *Given a set $D \subseteq \mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C}$, a binary operator $\circledast_D : 2^{\mathsf{Core}(\mathcal{P})} \times 2^{\mathsf{Core}(\mathcal{P})} \to 2^{\mathsf{Core}(\mathcal{P})}$ is a* composition *if $\mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h}_1) \circledast_D \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h}_2) = \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$, for any injective structure $(\dot{\mathfrak{s}}, \mathfrak{h})$, such that*

(i) $\mathrm{dom}(\dot{\mathfrak{s}}) \subseteq \mathcal{V}^1_{\mathcal{P}}$,
(ii) $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$,
(iii) $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) \subseteq \dot{\mathfrak{s}}(\mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C})$,
(iv) $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) \cap \mathrm{dom}(\mathfrak{h}) \subseteq \dot{\mathfrak{s}}(D) \subseteq \mathrm{dom}(\mathfrak{h})$.

We recall that $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) = \mathrm{loc}(\mathfrak{h}_1) \cap \mathrm{loc}(\mathfrak{h}_2)$. If $\mathcal{S}$ is a normalized set of rules, then for any core formula ϕ whose only occurrences of predicate atoms are of the form $\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t})$, we define $\mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\phi)$ as the homomorphic extension of $\mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t})) \stackrel{\mathsf{def}}{=} \mathsf{alloc}_{\mathcal{S}}(p(\mathbf{t}))$ to ϕ (see Definition 9). Assuming that $\mathcal{S}$ is a normalized set of rules and that a composition operation $\circledast_D$ (the construction of which will be described below, see Lemma 30) exists, we define the profile of a separating conjunction:

$$(\phi_1 * \phi_2, \mathsf{add}(X_1, F_1) \circledast_D \mathsf{add}(X_2, F_2)) \in \mathcal{F}_{\mathcal{P}}, \text{ if } (\phi_i, F_i) \in \mathcal{F}_{\mathcal{P}} \ \ X_i \stackrel{\mathsf{def}}{=} \mathsf{fv}(\phi_{3-i}) \setminus \mathsf{fv}(\phi_i),\ i = 1, 2$$
$$\mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\phi_1) \cap \mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\phi_2) = \emptyset, \quad D \stackrel{\mathsf{def}}{=} \mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\phi_1 * \phi_2) \cap (\mathsf{fv}(\phi_1) \cap \mathsf{fv}(\phi_2) \cup \mathbb{C})$$
$$\mathsf{add}(x, F) \stackrel{\mathsf{def}}{=} \{\exists_{\mathsf{h}} \mathbf{y} \forall_{\neg\mathsf{h}} \mathbf{z} \,.\, \psi \mid \exists_{\mathsf{h}} \mathbf{y} \forall_{\neg\mathsf{h}} \mathbf{z} \forall_{\neg\mathsf{h}} \hat{x} \,.\, \psi[\hat{x}/x] \in F\},\ \mathsf{add}(\{x_1, \ldots, x_n\}, F) \stackrel{\mathsf{def}}{=} \mathsf{add}(x_1, \ldots \mathsf{add}(x_n, F)) \quad (3)$$

The choice of the set D above ensures (together with the restriction to normal models) that $\circledast_D$ is indeed a composition operator. Intuitively, since the considered models are normal, every location in the frontier between the heaps corresponding to ϕ_1 and ϕ_2 will be associated with a variable, thus D denotes the set of allocated locations on the frontier. Note that, because $\mathcal{P}$ is normalized, $\mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\phi_1 * \phi_2)$ is well-defined. Because the properties of the composition operation hold when the models of its operands share the same store (Definition 26), we use the $\mathsf{add}(x, F)$ function that adds free variables (mapped to locations outside of the heap) to each core formula in F.

Existential Quantifiers. Since profiles involve only core formulæ obtained by the syntactic translation of a symbolic heap (Lemma 25), it is sufficient to consider only existentially quantified core formulæ, because the syntactic translation $\mathcal{T}(.)$ does not produce universal quantifiers. The profile of an existentially quantified core formula is given by the constraint:

$$(\exists_{\mathsf{h}} x' \,.\, \phi[x'/x],\ \mathsf{rem}(x, F)) \in \mathcal{F}_{\mathcal{P}}, \text{ if } x \in \mathsf{fv}(\phi),\ x' \in \mathcal{V}^2_{\mathcal{P}},\ x' \text{ not bound in } \phi,\ (\phi, F) \in \mathcal{F}_{\mathcal{P}},$$
$$\mathsf{rem}(x, F) \stackrel{\mathsf{def}}{=} \{\exists_{\mathsf{h}} \hat{x} \,.\, \psi[\hat{x}/x] \mid \psi \in F,\ x \in \mathsf{fv}(\psi),\ \hat{x} \text{ not in } \psi\} \cap \mathsf{Core}(\mathcal{P}) \cup \{\psi \mid \psi \in F,\ x \notin \mathsf{fv}(\psi)\} \quad (4)$$
$$\mathsf{rem}(\{x_1, \ldots, x_n\}, F) \stackrel{\mathsf{def}}{=} \mathsf{rem}(x_1, \ldots \mathsf{rem}(x_n, F) \ldots)$$

Note that $\hat{x}$ is a fresh variable, which is not bound or free in ψ. In particular, if $x \in \mathrm{roots}(\psi)$, then we must have $\hat{x} \in \mathcal{V}^2_{\mathcal{P}}$, so that $\exists_{\mathsf{h}} \hat{x} \,.\, \psi[\hat{x}/x] \in \mathsf{Core}(\mathcal{P})$. Similarly the variable x is replaced by a fresh variable $x' \in \mathcal{V}^2_{\mathcal{P}}$ in $\exists_{\mathsf{h}} x' \,.\, \phi[x'/x]$ to ensure that $\exists_{\mathsf{h}} x' \,.\, \phi[x'/x]$ is a core formula.

The Profile Function. Let $\mathcal{F}_{\mathcal{P}}$ be the least relation that satisfies the constraints (1), (2), (3) and (4). We prove that $\mathcal{F}_{\mathcal{P}}$ is a valid profile for $\mathcal{P}$, in the sense of Definition 24:

▶ **Lemma 27.** *Given a progressing and normalized entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$, a symbolic heap $\varphi \in \mathsf{SH}^{\mathfrak{R}}$ with $\mathsf{fv}(\varphi) \subseteq \mathcal{V}^1_{\mathcal{P}}$, a core formula $\phi \in \mathcal{T}(\varphi)$ and a set of core formulæ $F \subseteq \mathsf{Core}(\mathcal{P})$, we have $(\phi, F) \in \mathcal{F}_{\mathcal{P}}$ iff $F = \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$, for some injective normal $\mathfrak{C}_{\mathcal{S}}$-model $(\dot{\mathfrak{s}}, \mathfrak{h})$ of ϕ, with* $\mathrm{dom}(\dot{\mathfrak{s}}) = \mathsf{fv}(\varphi) \cup \mathbb{C}$.

The composition operation $\circledast_D$ works symbolically on core formulæ, by saturating the separating conjunction of two core formulæ via a *modus ponens*-style consequence operator.

▶ **Definition 28.** *Given formulæ ϕ, ψ, we write $\phi \Vdash \psi$ if $\phi = \varphi * [\alpha \mathrel{-\!\!\bullet} p(\mathbf{t})] * [(\beta * p(\mathbf{t})) \mathrel{-\!\!\bullet} q(\mathbf{u})]$ and $\psi = \varphi * [(\alpha * \beta) \mathrel{-\!\!\bullet} q(\mathbf{u})]$ (up to the commutativity of $*$ and the neutrality of emp) for some formula φ, predicate atoms $p(\mathbf{t})$ and $q(\mathbf{u})$ and conjunctions of predicate atoms α and β.*

▶ **Example 29.** Consider the structure $(\mathfrak{s}, \mathfrak{h})$ and the rules of Example 18. We have $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, with $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}_1) \models_{\mathfrak{C}_{\mathcal{S}}} q(y) \mathrel{-\!\!\bullet} p(x)$ and $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}_2) \models_{\mathcal{S}} q(y)$, i.e., $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}_2) \models_{\mathfrak{C}_{\mathcal{S}}} \mathsf{emp} \mathrel{-\!\!\bullet} q(y)$, thus $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}) \models_{\mathfrak{C}_{\mathcal{S}}} q(y) \mathrel{-\!\!\bullet} p(x) * \mathsf{emp} \mathrel{-\!\!\bullet} q(y) \Vdash \mathsf{emp} \mathrel{-\!\!\bullet} p(x)$. ⌟

We define a relation on the set of core formulæ $\mathsf{Core}(\mathcal{P})$, parameterized by a set $D \subseteq \mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C}$:

$$\begin{gathered} \exists_{\mathsf{h}} \mathbf{x}_1 \forall_{\neg\mathsf{h}} \mathbf{y}_1 \,.\, \psi_1, \exists_{\mathsf{h}} \mathbf{x}_2 \forall_{\neg\mathsf{h}} \mathbf{y}_2 \,.\, \psi_2 \Vvdash_D \exists_{\mathsf{h}} \mathbf{x} \forall_{\neg\mathsf{h}} \mathbf{y} \,.\, \psi \\ \text{if } \psi_1 * \psi_2 \Vdash^* \psi, \mathbf{x}_1 \cap \mathbf{x}_2 = \emptyset, \mathbf{x} = (\mathbf{x}_1 \cup \mathbf{x}_2) \cap \mathsf{fv}(\psi), \mathbf{y} = ((\mathbf{y}_1 \cup \mathbf{y}_2) \cap \mathsf{fv}(\psi)) \setminus \mathbf{x}, \mathrm{roots}_{\mathsf{lhs}}(\psi) \cap D = \emptyset. \end{gathered} \tag{5}$$

The composition operator is defined by lifting the $\Vvdash$ relation to sets of core formulæ:

$$F_1 \circledast_D F_2 \overset{\mathsf{def}}{=} \{\psi \mid \phi_1 \in F_1, \phi_2 \in F_2, \phi_1, \phi_2 \Vvdash_D \psi\} \tag{6}$$

We show that $\circledast_D$ is indeed a composition, in the sense of Definition 26:

▶ **Lemma 30.** *Let $\mathcal{S}$ be a normalized, progressing, connected and e-restricted set of rules, $D \subseteq \mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C}$ be a set of terms and $(\dot{\mathfrak{s}}, \mathfrak{h})$ be an injective structure, with* $\mathrm{dom}(\dot{\mathfrak{s}}) \subseteq \mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C}$. *Let $\mathfrak{h}_1$ and $\mathfrak{h}_2$ be two disjoint heaps, such that:*
(1) $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$,
(2) $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) \subseteq \dot{\mathfrak{s}}(\mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C})$ *and*
(3) $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) \cap \mathrm{dom}(\mathfrak{h}) \subseteq \dot{\mathfrak{s}}(D) \subseteq \mathrm{dom}(\mathfrak{h})$.
Then, we have $\mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h}) = \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h}_1) \circledast_D \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h}_2)$.

7 Main Result

In this section, we state the main complexity result of the paper. As a prerequisite, we prove that the size of the core formulæ needed to solve an entailment problem $\mathcal{P}$ is polynomial in $\mathrm{width}(\mathcal{P})$ and the number of such formulæ is simply exponential in $\mathrm{width}(\mathcal{P}) + \log(\mathrm{size}(\mathcal{P}))$.

▶ **Lemma 31.** *Given an entailment problem $\mathcal{P}$, for every formula $\phi \in \mathsf{Core}(\mathcal{P})$, we have $\mathrm{size}(\phi) = \mathcal{O}(\mathrm{width}(\mathcal{P})^2)$ and $||\mathsf{Core}(\mathcal{P})|| = 2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^3 \times \log(\mathrm{size}(\mathcal{P})))}$.*

▶ **Theorem 32.** *Checking the validity of progressing, connected and e-restricted entailment problems is* 2-EXPTIME*-complete.*

Proof. 2-EXPTIME-hardness follows from [6]; since the reduction in [6] involves no (dis-)equality, the considered systems are trivially e-restricted. We now prove 2-EXPTIME-membership. Let $\mathcal{P}$ be an e-restricted problem. By Lemma 11, we compute, in time $\mathrm{size}(\mathcal{P}) \cdot 2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^2)}$, an equivalent normalized e-restricted problem $\mathcal{P}_n$ of $\mathrm{size}(\mathcal{P}_n) =$

$\mathrm{size}(\mathcal{P}) \times 2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^2)}$ and $\mathrm{width}(\mathcal{P}_n) = \mathcal{O}(\mathrm{width}(\mathcal{P})^2)$. We fix an arbitrary set of variables $\mathcal{V}_{\mathcal{P}_n} = \mathcal{V}^1_{\mathcal{P}_n} \uplus \mathcal{V}^2_{\mathcal{P}_n}$ with $||\mathcal{V}^i_{\mathcal{P}_n}|| = \mathrm{width}(\mathcal{P}_n)$, for $i = 1, 2$ and we compute the relation $\mathcal{F}_{\mathcal{P}_n}$, using a Kleene iteration, as explained in Section 6 (Lemma 27). By Lemma 31, if $\psi \in \mathsf{Core}(\mathcal{P}_n)$ then $\mathrm{size}(\psi) = \mathcal{O}(\mathrm{width}(\mathcal{P})^2)$ and if $(\psi, F) \in \mathcal{F}_{\mathcal{P}_n}$ then $||F|| = 2^{\mathcal{O}(\mathrm{width}(\mathcal{P}_n)^3 \times \log(\mathrm{size}(\mathcal{P}_n)))} = 2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^8 \times \log(\mathrm{size}(\mathcal{P})))}$, hence $\mathcal{F}_{\mathcal{P}}$ can be computed in $2^{2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^8 \times \log(\mathrm{size}(\mathcal{P})))}}$ steps. It thus suffices to check that each of these steps can be performed in polynomial time w.r.t. $\mathsf{Core}(\mathcal{P}_n)$ and $\mathrm{size}(\mathcal{P}_n)$. This is straightforward for points-to atoms, predicate atoms and existential formulæ, by iterating on the rules in $\mathcal{P}_n$ and applying the construction rules (1), (2) and (4) respectively. For the disjoint composition, one has to compute the relation $\Vdash^*$, needed to build the operator $\circledast_D$, according to (5) and (6). We use again a Kleene iteration. It is easy to check that $\phi \Vdash \psi \Rightarrow \mathrm{size}(\psi) \leq \mathrm{size}(\phi)$, furthermore, one only needs to check relations of the form $\phi_1 * \phi_2 \Vdash \psi$ with $\phi_1, \phi_2, \psi \in \mathsf{Core}(\mathcal{P}_n)$. This entails that the number of iteration steps is $2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^8 \times \log(\mathrm{size}(\mathcal{P})))}$ and, moreover, each step can be performed in time polynomial w.r.t. $\mathsf{Core}(\mathcal{P}_n)$. Finally, we apply Lemma 25 to check that all the entailments in $\mathcal{P}_n$ are valid. This test can be performed in time polynomial w.r.t. $||\mathcal{F}_{\mathcal{P}_n}||$ and $\mathrm{size}(\mathcal{P}_n)$. ◀

8 Conclusion and Future Work

We presented a class of SL formulæ built from a set of inductively defined predicates, used to describe pointer-linked recursive data structures, whose entailment problem is 2-EXPTIME-complete. This fragment, consisting of so-called e-restricted formulæ, is a strict generalization of previous work defining three sufficient conditions for the decidability of entailments between SL formulæ, namely progress, connectivity and establishment [8, 12, 14]. On one hand, every progressing, connected and established entailment problem can be translated into an e-restricted problem. On the other hand, the models of e-restricted formulæ form a strict superset of the models of established formulæ. The proof for the 2-EXPTIME upper bound for e-restricted entailments leverages a novel technique used to prove the upper bound of established entailments [12, 14]. A natural question is whether the e-restrictedness condition can be dropped. We conjecture that this is not the case, and that entailment is undecidable for progressing, connected and non-e-restricted sets. Another issue is whether the generalization of symbolic heaps to use guarded negation, magic wand and septraction from [15] is possible for e-restricted entailment problems. The proof of these conjectures is on-going work.

References

1 Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In Anca Muscholl, editor, *FOSSACS 2014, ETAPS 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 411–425, 2014.

2 Josh Berdine, Byron Cook, and Samin Ishtiaq. Slayer: Memory safety for systems-level code. In Ganesh Gopalakrishnan andShaz Qadeer, editor, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *LNCS*, pages 178–183. Springer, 2011.

3 Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter W. O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving fast with software verification. In Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, volume 9058 of *LNCS*, pages 3–11. Springer, 2015.

4 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

5 Kamil Dudka, Petr Peringer, and Tomás Vojnar. Predator: A practical tool for checking manipulation of dynamic data structures using separation logic. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *LNCS*, pages 372–378. Springer, 2011.

6 Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Entailment checking in separation logic with inductive definitions is 2-exptime hard. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 191–211. EasyChair, 2020. URL: `https://easychair.org/publications/paper/DdNg`.

7 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag New York, Inc., 2006.

8 Radu Iosif, Adam Rogalewicz, and Jiri Simacek. The tree width of separation logic with recursive definitions. In *Proc. of CADE-24*, volume 7898 of *LNCS*, 2013.

9 Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Deciding entailments in inductive separation logic with tree automata. In Franck Cassez and Jean-François Raskin, editors, *ATVA 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 201–218. Springer, 2014.

10 Samin S Ishtiaq and Peter W O'Hearn. Bi as an assertion language for mutable data structures. In *ACM SIGPLAN Notices*, volume 36, pages 14–26, 2001.

11 Christina Jansen, Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. Unified reasoning about robustness properties of symbolic-heap separation logic. In Hongseok Yang, editor, *Programming Languages and Systems (ESOP'17)*, pages 611–638. Springer Berlin Heidelberg, 2017.

12 Jens Katelaan, Christoph Matheja, and Florian Zuleger. Effective entailment checking for separation logic with inductive definitions. In Tomás Vojnar and Lijun Zhang, editors, *TACAS 2019, Proceedings, Part II*, volume 11428 of *Lecture Notes in Computer Science*, pages 319–336. Springer, 2019.

13 Koji Nakazawa, Makoto Tatsuta, Daisuke Kimura, and Mitsuru Yamamura. Cyclic Theorem Prover for Separation Logic by Magic Wand. In *ADSL 18 (First Workshop on Automated Deduction for Separation Logics)*, July 2018. Oxford, United Kingdom.

14 Jens Pagel, Christoph Matheja, and Florian Zuleger. Complete entailment checking for separation logic with inductive definitions, 2020. `arXiv:2002.01202`.

15 Jens Pagel and Florian Zuleger. Beyond symbolic heaps: Deciding separation logic with inductive definitions. In *LPAR-23*, volume 73 of *EPiC Series in Computing*, pages 390–408. EasyChair, 2020. URL: `https://easychair.org/publications/paper/VTGk`.

16 J.C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. of LICS'02*, 2002.

17 Neil Robertson and P.D Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

Church's Thesis and Related Axioms in Coq's Type Theory

Yannick Forster
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
forster@cs.uni-saarland.de

Abstract

"Church's thesis" (CT) as an axiom in constructive logic states that every total function of type $\mathbb{N} \to \mathbb{N}$ is computable, i.e. definable in a model of computation. CT is inconsistent both in classical mathematics and in Brouwer's intuitionism since it contradicts weak Kőnig's lemma and the fan theorem, respectively. Recently, CT was proved consistent for (univalent) constructive type theory.

Since neither weak Kőnig's lemma nor the fan theorem is a consequence of just logical axioms or just choice-like axioms assumed in constructive logic, it seems likely that CT is inconsistent only with a combination of classical logic and choice axioms. We study consequences of CT and its relation to several classes of axioms in Coq's type theory, a constructive type theory with a universe of propositions which proves neither classical logical axioms nor strong choice axioms.

We thereby provide a partial answer to the question as to which axioms may preserve computational intuitions inherent to type theory, and which certainly do not. The paper can also be read as a broad survey of axioms in type theory, with all results mechanised in the Coq proof assistant.

2012 ACM Subject Classification Theory of computation → Constructive mathematics; Theory of computation → Type theory

Keywords and phrases Church's thesis, constructive type theory, constructive reverse mathematics, synthetic computability theory, Coq

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.21

Supplementary Material `https://github.com/uds-psl/churchs-thesis-coq`

Acknowledgements I want to thank Gert Smolka, Andrej Dudenhefner, Dominik Kirst, and Dominique Larchey-Wendling for discussions and feedback on drafts of this paper. Special thanks go to the anonymous reviewers for their helpful ideas, constructive comments, and editorial suggestions.

1 Introduction

The intuition that the concept of a constructively defined function and a computable function can be identified is prevalent in intuitionistic logic since the advent of recursion theory and is maybe most natural in constructive type theory, where computation is primitive.

A formalisation of the intuition is the axiom CT ("Church's thesis"), stating that every function is computable, i.e. definable in a model of computation. CT is well-studied as part of Russian constructivism [34] and in the field of constructive reverse mathematics [11, 25].

CT allows proving results of recursion theory without extensive references to a model of computation, since one can reason with functions instead. While such synthethic developments of computability theory [1, 7, 37] can be carried out in principle without assuming any axioms [14], assuming CT allows stronger results: CT essentially provides a universal machine w.r.t. all functions in the logic, allowing to show the non-existence of certain deciding functions – whose existence is logically independent with no axioms present.

It is easy to see that CT is in conflict with traditional classical mathematics, since the law of excluded middle LEM together with a form of the axiom of countable choice $\mathsf{AC}_{\mathbb{N},\mathbb{N}}$ allows the definition of non-computable functions [46]. This observation can be sharpened in various ways: To define a non-computable function directly, the weak limited principle of omniscience WLPO and the countable unique choice axiom $\mathsf{AUC}_{\mathbb{N},\mathbb{B}}$ suffice. Alternatively,

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 21; pp. 21:1–21:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Kleene noticed that there is a decidable tree predicate with infinitely many nodes but no computable infinite path [28]. If functions and computable functions are identified via CT, a Kleene tree is in conflict with weak Kőnig's lemma WKL and with Brouwer's fan theorem.

It is however well-known that CT is consistent in Heyting arithmetic with Markov's principle MP [27] which given CT states that termination of computation is stable under double negation. Recently, Swan and Uemura [43] proved that CT is consistent in univalent type theory with propositional truncation and MP.

While predicative Martin-Löf type theory as formalisation of Bishop's constructive mathematics proves the full axiom of choice AC, univalent type theory usually only proves the axiom of unique choice AUC. But since $\mathsf{AUC}_{\mathbb{N},\mathbb{B}}$ suffices to show that LEM implies $\neg$CT, classical logic is incompatible with CT in both predicative and in univalent type theory.

In the (polymorphic) calculus of (cumulative) inductive constructions, a constructive type theory with a separate, impredicative universe of propositions as implemented by the proof assistant Coq [44], none of AC, AUC, and $\mathsf{AUC}_{\mathbb{N},\mathbb{B}}$ are provable. This is because large eliminations on existential quantifications are not allowed in general [35], meaning one can not recover a function in general from a proof of $\forall x.\exists y.\ Rxy$. However, choice axioms as well al LEM can be consistently assumed in Coq's type theory [47]. Furthermore, it seems likely that the consistency proof for CT in [43] can be adapted for Coq's type theory.

This puts Coq's type theory in a special position: Since to disprove CT one needs a (weak) classical logical axiom and a (weak) choice axiom, assuming just classical logical axioms or just choice axioms might be consistent with CT. This paper is intended to serve as a preliminary report towards this consistency question, approximating it by surveying results from intuitionistic logic and constructive reverse mathematics in constructive type theory with a separate universe of propositions, with a special focus on CT and other axioms based on notions from computability theory. Specifically, we discuss these propositional axioms:

- computational enumerability axioms (EA, EPF) and Kleene trees (KT) in Section 5
- extensionality axioms like functional extensionality (Fext), propositional extensionality (Pext), and proof irrelevance (PI) in Section 6
- classical logical axioms like the principle of excluded middle (LEM, WLEM), independence of premises (IP), and limited principles of omniscience (LPO, WLPO, LLPO) in Section 7
- axioms of Russian constructivism like Markov's principle (MP) in Section 8
- choice axioms like the axiom of choice (AC), countable choice (ACC, $\mathsf{AC}_{\mathbb{N},\mathbb{N}}$, $\mathsf{AC}_{\mathbb{N},\mathbb{B}}$), dependent choice (ADC), and unique choice (AUC, $\mathsf{AUC}_{\mathbb{N},\mathbb{B}}$) in Section 9
- axioms on trees like weak Kőnig's lemma (WKL) and the fan theorem (FAN) in Section 10
- axioms regarding continuity and Brouwerian principles (Homeo, Cont, WC-N) in Section 11

The following hyper-linked diagram displays provable implications and incompatible axioms.

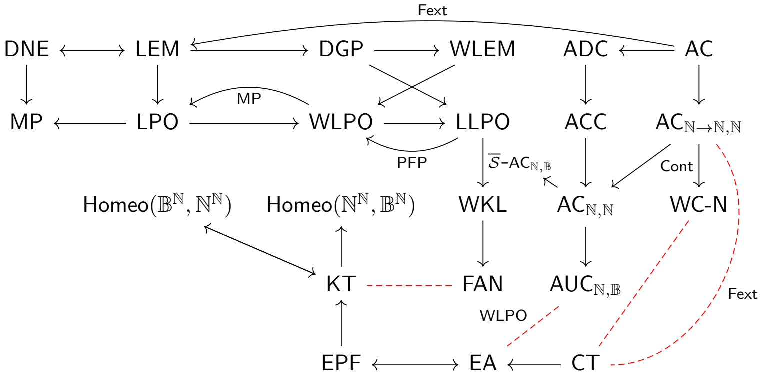

Fext
DNE
LEM
DGP
WLEM
ADC
AC
MP
MP
LPO
WLPO
LLPO
ACC
$\mathsf{AC}_{\mathbb{N}\to\mathbb{N},\mathbb{N}}$
PFP
$\mathcal{S}$-$\mathsf{AC}_{\mathbb{N},\mathbb{B}}$
Cont
$\mathsf{Homeo}(\mathbb{B}^{\mathbb{N}},\mathbb{N}^{\mathbb{N}})$
$\mathsf{Homeo}(\mathbb{N}^{\mathbb{N}},\mathbb{B}^{\mathbb{N}})$
WKL
$\mathsf{AC}_{\mathbb{N},\mathbb{N}}$
WC-N
KT
FAN
$\mathsf{AUC}_{\mathbb{N},\mathbb{B}}$
Fext
WLPO
EPF
EA
CT

Figure 1 Overview of results. → are implications, - - - denotes incompatible axioms.

All results in this paper are mechanised in the Coq proof assistant and the proof scripts are accessible at `https://github.com/uds-psl/churchs-thesis-coq`. The statements in this document are hyperlinked to their Coq proof, indicated by a 🐓-symbol.

Outline. Section 2 establishes necessary preliminaries regarding Coq's type theory and introduces the notions of (synthetic) decidability, enumerability, and semi-decidability. Section 3 introduces CT formally, together with the related synthetic axioms EA and EPF. Section 4 contains undecidability proofs based on CT. Section 5 introduces decidable binary trees and constructs a Kleene tree. The connection of CT to the classes of axioms as listed above is surveyed in Sections 6 to 11. Section 12 contains concluding remarks.

2 Preliminaries

We work in the polymorphic calculus of cumulative inductive constructions as implemented by the Coq proof assistant [44], which we will refer to as "Coq's type theory". The calculus is a constructive type theory with a cumulative hierarchy of types $\mathbb{T}_i$ (where i is a natural number, but we leave out the index from now on), an impredicative universe of propositions $\mathbb{P} \subseteq \mathbb{T}$, and inductive types in every universe. The inductive types of interest in this paper are

$$\begin{array}{ll} n : \mathbb{N} ::= 0 \mid \mathsf{S}\, n & b : \mathbb{B} ::= \mathsf{false} \mid \mathsf{true} \\ o : \mathbb{O}A ::= \mathsf{None} \mid \mathsf{Some}\, a \quad \textit{where } a : A & l : \mathbb{L}A ::= [] \mid a :: l \quad \textit{where } a : A \\ A + B := \mathsf{inl}\, a \mid \mathsf{inr}\, b \quad \textit{where } a : A \textit{ and } b : B & A \times B := (a, b) \quad \textit{where } a : A \textit{ and } b : B \end{array}$$

One can easily construct a pairing function $\langle \_ , \_ \rangle : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ and for all $f : \mathbb{N} \to \mathbb{N} \to X$ an inverse construction $\lambda \langle n, m \rangle.\ fnm$ of type $\mathbb{N} \to X$ s.t. $(\lambda \langle n, m \rangle.\ fnm)\langle n, m \rangle = fnm$.

We write $n =_{\mathbb{B}} m$ for the boolean equality decider on $\mathbb{N}$, and $\neg_{\mathbb{B}}$ for boolean negation.

If $l : \mathbb{L}A$ then $l[n] : \mathbb{O}A$ denotes the n-th element of l. If $n < |l|$ we can assume $l[n] : A$.

We write $\forall x : X.\ Ax$ for both dependent functions and logical universal quantification, $\exists x : X.\ Ax$ where $A : X \to \mathbb{P}$ for existential quantification and $\Sigma x : X.\ Ax$ where $A : X \to \mathbb{T}$ for dependent pairs, with elements (x, y). Dependent pairs can be eliminated into arbitrary types, i.e. there is an elimination principle of type $\forall p : (\Sigma x.\ Ax) \to \mathbb{T}.\ (\forall (x : X)(y : Ax).\ p(x, y)) \to \forall (s : \Sigma x.\ Ax).\ ps$. We call such a principle eliminating a proposition into arbitrary types a *large elimination principle*, following the terminology "large elimination" for Coq's case analysis construct `match` [35]. Crucially, Coq's type theory proves a large elimination principle for the falsity proposition $\bot$, i.e. explosion applies to arbitrary types: $\forall A : \mathbb{T}.\ \bot \to A$. In contrast, existential quantification can only be eliminated for $p : (\exists x.\ Ax) \to \mathbb{P}$, but the following more specific large elimination principle is provable:

🐓 **Lemma 1.** *There is a guarded minimisation function $\mu_{\mathbb{N}}$ of the following type:*

$$\mu_{\mathbb{N}} : \forall f : \mathbb{N} \to \mathbb{B}.\ (\exists n.\ fn = \mathsf{true}) \to \Sigma n.\ fn = \mathsf{true} \land \forall m.\ fm = \mathsf{true} \to m \geq n.$$

There are various implementations of such a minimisation function in Coq's Standard Library[1] One uses a (recursive) large elimination principle for the accessibility predicate, see e.g. [32, §2.7, §4.1, §4.2] and [6, §14.2.3, §15.4] for a contemporary overview how to implement large eliminations principles. We will not need any other large elimination principle in this paper. A restriction of large elimination in general is necessary for consistency of Coq [8]. As a by-product, the computational universe $\mathbb{T}$ is separated from the logical universe $\mathbb{P}$, allowing classical logic in $\mathbb{P}$ to be assumed while the computational intuitions for $\mathbb{T}$ remain intact.

[1] The idea was conceived independently by Benjamin Werner and Jean-François Monin in the 1990s.

$\mathsf{part}\,A : \mathbb{T}$	partial values over $A : \mathbb{T}$	
$\overset{!}{=} : \mathsf{part}\,A \to A \to \mathbb{P}$	definedness of values	$x \overset{!}{=} a_1 \to x \overset{!}{=} a_2 \to a_1 = a_2$
$(x : \mathsf{part}\,A)\downarrow : \mathbb{P}$		$x\downarrow := \exists a.\ x \overset{!}{=} a$
$\equiv_{\mathsf{part}\,A} : \mathsf{part}\,A \to \mathsf{part}\,A \to \mathbb{P}$	equivalence	$x \equiv_{\mathsf{part}\,A} y := (\forall a.\ x \overset{!}{=} a \leftrightarrow y \overset{!}{=} a)$
$\mathsf{ret} : A \to \mathsf{part}\,A$	monadic return	$\mathsf{ret}\,a \overset{!}{=} a$
$\mathsf{undef} : \mathsf{part}\,A$	undefined value	$\nexists a.\mathsf{undef} \overset{!}{=} a$
$\gg\!\!= : \mathsf{part}\,A \to (A \to \mathsf{part}\,B) \to \mathsf{part}\,B$	monadic bind	$x \gg\!\!= f \overset{!}{=} b \leftrightarrow (\exists a.\ x \overset{!}{=} a \wedge fa \overset{!}{=} b)$
$\mu : (\mathbb{N} \to \mathbb{B}) \to \mathsf{part}\,\mathbb{N}$	unbounded search	$\mu f \overset{!}{=} n \leftrightarrow fn = \mathsf{true} \wedge$ $\forall m < n.\ fm = \mathsf{false}$
$\mathsf{seval} : \mathsf{part}\,A \to \mathbb{N} \to \mathbb{O}A$	step-indexed evaluation	$x \overset{!}{=} a \leftrightarrow \exists n.\ \mathsf{seval}\,xn = \mathsf{Some}\,a$

Figure 2 A monad for partial values.

2.1 Partial Functions

All definable functions in type theory are total by definition. To model partiality, one often resorts to functional relations $R : A \to B \to \mathbb{P}$ or step-indexed functions $A \to \mathbb{N} \to \mathbb{O}B$, as for instance pioneered by Richman [37] in constructive logic, see e.g. [12] for a comprehensive overview.

For our purpose, we simply assume a type $\mathsf{part}\,A$ for $A : \mathbb{T}$ and a definedness relation $\overset{!}{=} : \mathsf{part}\,A \to A \to \mathbb{P}$ and write $A \nrightarrow B$ for $A \to \mathsf{part}\,B$. We assume monadic structure for part (ret and $\gg\!\!=$), an undefined value (undef), a minimisation operation (μ), and a step-indexed evaluator (seval). The operations and their specifications are listed in Figure 2.

2.2 Equivalence relations on functions

Besides intensional equality ($=$), we will consider other more extensional equivalence relations in this paper. For instance, extensional equality of functions f, g ($\forall x.\ fx = gx$), extensional equivalence of predicates p, q ($\forall x.\ px \leftrightarrow qx$), or range equivalence of functions f, g ($\forall x.\ (\exists y.\ fy = x) \leftrightarrow (\exists y.\ gy = x)$). We will denote all of these equivalence relations with the symbol $\equiv$ and indicate what is meant by an index. For discrete X (e.g. $\mathbb{N}$, $\mathbb{O}\mathbb{N}$, $\mathbb{L}\mathbb{B}$, $\dots$), $\equiv_X$ denotes equality, $\equiv_{\mathbb{P}}$ denotes logical equivalence, $\equiv_{A \to B}$ denotes an extensional lift of $\equiv_B$, $\equiv_{A \to \mathbb{P}}$ denotes extensional equivalence, and $\equiv_{\mathsf{ran}}$ denotes range equivalence.

Assuming the existence of surjections $A \to (A \to B)$ may or may not be consistent, depending on the particular equivalence relation. We introduce the notion of *surjection w.r.t.* $\equiv_B$ as $\forall b : B.\ \exists a : A. fa \equiv_B b$. We call a function $f : A \to B$ an *injection w.r.t.* $\equiv_A$ *and* $\equiv_B$ if $\forall a_1 a_2.\ fa_1 \equiv_B fa_2 \to a_1 \equiv_A a_2$ and a *bijection* if it is an injection and surjection.

One formulation of Cantor's theorem is that there is no surjection $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ w.r.t. $=$. However, the same proof can be used for the following strengthening of Cantor's theorem:

Fact 2 (Cantor). *There is no surjection* $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$ *w.r.t.* $\equiv_{\mathbb{N} \to \mathbb{N}}$.

2.3 Decidability, Semi-decidability, Enumerability, Reducibility

We define decidability, (co-)semi-decidability, and enumerability for predicates $p : X \to \mathbb{P}$:

$$\begin{array}{lll r}
\mathcal{D}p & := \exists f : X \to \mathbb{B}. & \forall x.\ px \leftrightarrow fx = \mathsf{true} & \text{(“p is decidable”)} \\
\mathcal{S}p & := \exists f : X \to \mathbb{N} \to \mathbb{B}. & \forall x.\ px \leftrightarrow \exists n. fxn = \mathsf{true} & \text{(“p is semi-decidable”)} \\
\overline{\mathcal{S}}p & := \exists f : X \to \mathbb{N} \to \mathbb{B}. & \forall x.\ px \leftrightarrow \forall n. fxn = \mathsf{false} & \text{(“p is co-semi-decidable”)} \\
\mathcal{E}p & := \exists f : \mathbb{N} \to \mathbb{O}X. & \forall x.\ px \leftrightarrow \exists n. fn = \mathsf{Some}\,x & \text{(“p is enumerable”)}
\end{array}$$

Although all notions are defined on unary predicates, we use them on n-ary relations via (implicit) uncurrying. We write $\overline{p}$ for the complement $\lambda x.\ \neg px$ of p. We call a type X *discrete* if its equality relation $=_X$ is decidable and *enumerable* if the predicate $\lambda x.\top$ is enumerable.

Traditionally, propositions P s.t. $P \leftrightarrow (\exists n.\ fn = \mathsf{true})$ for some f are often called Σ^0_1 or "simply existential", and P s.t. $P \leftrightarrow (\forall n.\ fn = \mathsf{false})$ are called Π^0_1 or "simply universal". Semi-decidable predicates are pointwise Σ^0_1, and co-semi-decidable predicates are pointwise Π^0_1. Note that neither $\overline{\mathcal{S}}p \to \mathcal{S}\overline{p}$ nor the converse is provable, only the following connections:

Lemma 3. *The following hold:*
1. *Decidable predicates are semi-decidable and co-semi-decidable.*
2. *Semi-decidable predicates on enumerable types are enumerable.*
3. *Enumerable predicates on discrete types are semi-decidable.*
4. *The complement of semi-decidable predicates is co-semi-decidable.*

Lemma 4. *Decidable predicates are closed under complementation. Decidable, enumerable, and semi-decidable predicates are closed under (pointwise) conjunction and disjunction.*

3 Church's thesis in type theory

Church's thesis for total functions (CT) states that every function of type $\mathbb{N} \to \mathbb{N}$ is algorithmic. Thus CT is a relativisation of the function space $\mathbb{N} \to \mathbb{N}$ w.r.t. a given (Turing-complete) model of computation, reminiscent of the axiom $V = L$ in set theory [29].

We first define CT by abstracting away from a concrete model of computation and work with an *abstract model of computation*, consisting of an *abstract computation function* $Tcxn$ (with $T : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \to \mathbb{O}\mathbb{N}$), assigning to a code c (to be interpreted as the code of a partial recursive function in a model of computation), an input number x, and a step index n an output number y if the code terminates in n steps on x with value y. The function Tcx is assumed to be monotonic, i.e. increasing the step index does not change the potential value:

$$Tcxn_1 = \mathsf{Some}\,y \to \forall n_2 \geq n_1.\ Tcxn_2 = \mathsf{Some}\,y.$$

Based on T we define a computability relation between $c : \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$:

$$c \sim f := \forall x.\exists n.\ Tcxn = \mathsf{Some}\,(fx).$$

Since T is monotonic, $\sim$ is extensional, i.e. $n \sim f_1 \to n \sim f_2 \to \forall x.\ f_1 x = f_2 x$. We define Church's thesis for total functions relative to an abstract computation function T:

$$\mathsf{CT}_T := \forall f : \mathbb{N} \to \mathbb{N}.\exists n : \mathbb{N}.\ n \sim f$$

Note that CT_T is clearly not consistent for every choice of T. If we write CT without index, we mean T to be the step-indexed evaluation function of a concrete, Turing-complete model of computation. For the mechanisation we could for instance pick the equivalent models of Turing machines [17], λ-calculus [21], μ-recursive functions [30], or register machines [18, 31]. It seems likely that the consistency proof of CT in [43] can be adapted to Coq.

Since specific properties of the model of computation are not needed, we develop and mechanise all results of this paper parameterised in an arbitrary T. Thus, we could also state all results in terms of a fully synthetic Church's thesis axiom $\Sigma T.\mathsf{CT}_T$.

▶ **Fact 5.** $\mathsf{CT} \to \Sigma T.\mathsf{CT}_T$

Note that the implication is strict: An abstract computation function does not rule out oracles for e.g. the halting problem of Turing machines, whereas CT – with T defined in terms of a standard, Turing-complete model of computation – proves the undecidability of the Turing machine halting problem.

3.1 Bauer's enumerability axiom EA

In proofs of theorems with CT_T as assumption, T can be used as replacement for a *universal machine*. Bauer [1] develops computability theory synthetically using the axiom "the set of enumerable sets of natural numbers is enumerable", which is equivalent to $\Sigma T.\mathsf{CT}_T$ and thus strictly weaker than CT, but can also be used in place of a universal machine. We introduce Bauer's axiom in our setting as EA' and immediately introduce a strengthening EA s.t. $(\Sigma T.\mathsf{CT}_T) \leftrightarrow \mathsf{EA}$ and $\mathsf{EA} \to \mathsf{EA}'$:

$$\mathsf{EA}' := \Sigma \mathcal{W} : \mathbb{N} \to (\mathbb{N} \to \mathbb{P}).\forall p : \mathbb{N} \to \mathbb{P}.\ \mathcal{E}p \leftrightarrow \exists c.\ \mathcal{W}c \equiv_{\mathbb{N}\to\mathbb{P}} p$$

That is, EA' states that there is an enumerator $\mathcal{W}$ of all enumerable predicates, up to extensionality. In contrast, EA poses the existence of an enumerator of all possible enumerators, up to range equivalence:

$$\mathsf{EA} := \Sigma \varphi : \mathbb{N} \to (\mathbb{N} \to \mathbb{O}\mathbb{N}).\forall f : \mathbb{N} \to \mathbb{O}\mathbb{N}.\exists c.\ \varphi c \equiv_{\mathsf{ran}} f$$

That is, φ is a surjection w.r.t. range equivalence $f \equiv_{\mathsf{ran}} g$, where $\varphi c \equiv_{\mathsf{ran}} f \leftrightarrow \forall x.(\exists n.\varphi c n = \mathsf{Some}\, x) \leftrightarrow (\exists n.f n = \mathsf{Some}\, x)$.

Note the two different roles of natural numbers in the two axioms: If we would consider predicates over a general type X we would have $\mathcal{W} : \mathbb{N} \to (X \to \mathbb{P})$ and $\varphi : \mathbb{N} \to (\mathbb{N} \to \mathbb{O}X)$, i.e. $\mathcal{W}c$ would be an enumerable predicate and φc an enumerator of a predicate $X \to \mathbb{P}$.

We start by proving $\mathsf{CT}_T \to \mathsf{EA}$ by constructing φ from an arbitrary T:

$$\varphi c\langle n, m\rangle := \textbf{if } Tcnm \textbf{ is } \mathsf{Some}\, x \textbf{ then } \mathsf{S}x \textbf{ else } 0$$

Lemma 6. *If* CT_T *then* $\forall f : \mathbb{N} \to \mathbb{O}\mathbb{N}.\exists c.\ \varphi c \equiv_{\mathsf{ran}} f$.

Proof. The direction from left to right to establish $\equiv_{\mathsf{ran}}$ is based on the fact that if $Tcxn_1 = \mathsf{Some}\, y_1$ and $Tcxn_2 = \mathsf{Some}\, y_2$ then $y_1 = y_2$. The other direction is straightforward. ◀

Theorem 7. $\forall T.\ \mathsf{CT}_T \to \mathsf{EA}$

We now prove $\mathsf{EA} \to \mathsf{EA}'$ by constructing $\mathcal{W}$ from φ: $\mathcal{W}cx := \exists n.\varphi c n = \mathsf{Some}\, x$.

Lemma 8. *If* EA *then* $\forall p : \mathbb{N} \to \mathbb{P}.\ \mathcal{E}p \leftrightarrow \exists c.\ \mathcal{W}c \equiv_{\mathbb{N}\to\mathbb{P}} p$.

Proof.
$$\begin{aligned} \mathcal{E}p &\leftrightarrow \exists f : \mathbb{N} \to \mathbb{O}\mathbb{N}.\forall x.\ px \leftrightarrow \exists n.\ fn = \mathsf{Some}\, x && \text{(def. } \mathcal{E}\text{)} \\ &\leftrightarrow \exists c.\forall x.\ px \leftrightarrow \exists n.\ \varphi c n = \mathsf{Some}\, x && (\mathsf{EA}) \\ &\leftrightarrow \exists c.\ \mathcal{W}c \equiv_{\mathbb{N}\to\mathbb{P}} p && \text{(def. } \equiv_{\mathbb{N}\to\mathbb{P}}\text{)} \end{aligned}$$
◀

Theorem 9. $\mathsf{EA} \to \mathsf{EA}'$

3.2 Richman's Enumerability of Partial Functions EPF

Richman [37] introduces a different purely synthetic axiom as replacement for a universal machine and assumes that "partial functions are countable", which is equivalent to EA.

$$\mathsf{EPF} := \Sigma e : \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N}).\forall f : \mathbb{N} \rightharpoonup \mathbb{N}.\exists n.\ en \equiv_{\mathbb{N}\rightharpoonup\mathbb{N}} f$$

Theorem 10. $\mathsf{EPF} \to \mathsf{EA}$

Proof. Let e be given. $\varphi c\langle n, m\rangle := \mathsf{seval}\ (ecn)\ m$ is the wanted enumerator. ◀

Theorem 11. $\mathsf{EA} \to \mathsf{EPF}$

Proof. Let φ be given. Then

$$\begin{aligned} ecx &:= (\mu\,(\lambda n.\ \textbf{if}\ \varphi cn\ \textbf{is}\ \mathsf{Some}\,\langle x', y'\rangle\ \textbf{then}\ x =_{\mathbb{B}} x'\ \textbf{else}\ \mathsf{false})) \gg\!= \\ &\qquad \lambda n.\ \textbf{if}\ \varphi cn\ \textbf{is}\ \mathsf{Some}\,\langle x', y'\rangle\ \textbf{then}\ \mathsf{ret}\, y'\ \textbf{else}\ \mathsf{undef} \end{aligned}$$

is the wanted enumerator. ◀

EPF implies the fully synthetic version of CT:

Lemma 12. $\mathsf{EPF} \to \Sigma T.\ \mathsf{CT}_T$

Proof. Assume $e : \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N})$ surjective w.r.t. $\equiv_{\mathbb{N} \rightharpoonup \mathbb{N}}$. Define $Tcxn := \mathsf{seval}\ (ecx)\ n$. It is straightforward to prove that T is monotonic and that CT holds. ◀

The axiom EPF can be weakened to cover just boolean functions:

$$\mathsf{EPF}_{\mathbb{B}} := \Sigma e : \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{B}). \forall f : \mathbb{N} \rightharpoonup \mathbb{B}. \exists n.\ en \equiv_{\mathbb{N} \rightharpoonup \mathbb{B}} f$$

Lemma 13. $\mathsf{EPF} \to \mathsf{EPF}_{\mathbb{B}}$

The reverse direction seems not to be provable.

4 Halting Problems

For this section we assume EA, i.e. $\varphi : \mathbb{N} \to (\mathbb{N} \to \mathbb{O}\mathbb{N})$ s.t. $\forall f : \mathbb{N} \to \mathbb{O}\mathbb{N}. \exists c.\ \varphi c \equiv_{\mathsf{ran}} f$. Recall Lemma 8 stating that $\forall p : \mathbb{N} \to \mathbb{P}.\ \mathcal{E}p \leftrightarrow \exists c.\ \mathcal{W}c \equiv_{\mathbb{N} \to \mathbb{P}} p$.

We define $\mathsf{K}_0 n := \mathcal{W}nn$ and prove our first negative result:

Lemma 14. $\neg\mathcal{E}\overline{\mathsf{K}_0}$

Proof. Assume $\mathcal{E}(\lambda n. \neg\mathcal{W}nn)$. By specification of $\mathcal{W}$ there is c s.t. $\forall n. \mathcal{W}cn \leftrightarrow \neg\mathcal{W}nn$. In particular, $\mathcal{W}cc \leftrightarrow \neg\mathcal{W}cc$, which is contradictory. ◀

Corollary 15. *$\neg\mathcal{D}\mathsf{K}_0$, $\neg\mathcal{D}\overline{\mathsf{K}_0}$, $\neg\mathcal{D}\mathcal{W}$ and $\neg\mathcal{D}\overline{\mathcal{W}}$.*

Intuitively, K_0 can be seen as analogous to the self-halting problem: $\mathsf{K}_0 n$ states that n considered as an enumerator outputs itself in its range (rather than halting on itself).

It is also easy to show that $\mathcal{W}$ and thus K_0 are enumerable:

Lemma 16. $\mathcal{E}\mathcal{W}$

Proof. Via $f\langle n, m\rangle := \textbf{if}\ \varphi nm\ \textbf{is}\ \mathsf{Some}\, k\ \textbf{then}\ \mathsf{Some}\,(n, k)\ \textbf{else}\ \mathsf{None}$. ◀

Corollary 17. $\mathcal{E}\mathsf{K}_0$

Since Bauer [1] bases his development on EA′, he needs the axiom of countable choice to prove that $\mathcal{W}$ is enumerable, whereas EA allows an axiom-free proof of this fact.

Another well-known traditional result is that a problem is enumerable if and only if it many-one reduces to the halting problem K, which can be proved without reference to EA.

$$p \preceq_m q := \exists f : X \to Y. \forall x.\ px \leftrightarrow q(fx) \qquad \mathsf{K}(f : \mathbb{N} \to \mathbb{B}) := \exists n.\ fn = \mathsf{true}$$

Fact 18. *For all $p : X \to \mathbb{P}$, $p \preceq_m \mathsf{K} \leftrightarrow \mathcal{S}p$.*

Corollary 19. $\mathcal{S}\mathsf{K}$

Corollary 20. *For all* $p : \mathbb{N} \to \mathbb{P}$, $p \preceq_m \mathsf{K} \leftrightarrow \mathcal{E}p$.

Using the non-enumerability of $\overline{\mathsf{K}_0}$ we can now prove our first negative result by reduction:

Corollary 21. $\mathsf{K}_0 \preceq_m \mathsf{K}$, *and thus* $\neg\mathcal{E}\overline{\mathsf{K}}$, $\neg\mathcal{D}\mathsf{K}$, *and* $\neg\mathcal{D}\overline{\mathsf{K}}$.

We can also define $\mathsf{K}_\mathbb{N} := \lambda f : \mathbb{N} \to \mathbb{N}.\ \exists n.\ fn \neq 0$:

Fact 22. $\mathsf{K} \preceq_m \mathsf{K}_\mathbb{N}$, $\mathsf{K}_\mathbb{N} \preceq_m \mathsf{K}$, $\overline{\mathsf{K}_\mathbb{N}} \equiv_{(\mathbb{N}\to\mathbb{N})\to\mathbb{P}} \lambda f.\ \forall n.\ fn = 0$, *and thus* $\neg\mathcal{D}(\lambda f.\ \forall n.\ fn = 0)$.

5 Kleene Trees

In a lecture in 1953 Kleene [28] gave an example how the axioms of Brouwer's intuitionism fail if all functions are considered computable by constructing an infinite decidable binary tree with no computable infinite path. The existence of such a Kleene tree (KT) is in contradiction to Brouwer's fan theorem, which we will discuss later. We prove that $\mathsf{EPF}_\mathbb{B}$ implies KT.

For this purpose, we call a predicate $\tau : \mathbb{LB} \to \mathbb{P}$ a (decidable) *binary tree* if

(a) τ is decidable: $\exists f.\forall u.\tau u \leftrightarrow fu = \mathsf{true}$

(b) τ is non-empty: $\exists u.\tau u$

(c) τ is prefix-closed: If τu_2 and $u_1 \sqsubseteq u_2$ then τu_1 (where $u_1 \sqsubseteq u_2 := \exists u'.\ u_2 = u_1 + u'$).

We will just speak of trees instead of decidable binary trees in the following.

Fact 23. *For every tree* τ, $\tau[]$ *holds.*

Furthermore, a decidable binary tree τ …

- …is *bounded* if $\exists n.\forall u.|u| \geq n \to \neg\tau u$
- …is *well-founded* if $\forall f.\exists n.\neg\tau[f0,\dots,fn]$
- …has an *infinite path* if $\exists f.\forall n.\tau[f0,\dots,fn]$

Fact 24. *A tree is not bounded if and only if it is* infinite, *defined as* $\forall n.\exists u.\ |u| \geq n \wedge \tau u$.

Fact 25. *Every bounded tree is well-founded and every tree with an infinite path is infinite.*

Note that both implications are strict: In our setting we cannot prove boundedness from well-foundedness nor obtain an infinite path from infiniteness, as can be seen from a Kleene tree:

KT := *There exists an infinite, well-founded, decidable binary tree.*

We follow Bauer [2] to construct a Kleene tree.

Lemma 26. *Given* $\mathsf{EPF}_\mathbb{B}$ *one can construct* $d : \mathbb{N} \rightharpoonup \mathbb{B}$ *s.t.* $\forall f : \mathbb{N} \to \mathbb{B}.\exists nb.\ dn \overset{!}{=} b \wedge fn \neq b$.

Proof. Define $dn := enn \gg\!= \lambda b.\ \mathsf{ret}\,(\neg_\mathbb{B} b)$. ◀

We define $\tau_K u := \forall n < |u|.\forall x.\ \mathsf{seval}\ (dn)\ |u| = \mathsf{Some}\,x \to u[n] = \mathsf{Some}\,x$. Intuitively, τ_K contains all paths $u = [b_0, b_1, \dots, b_n]$ which might be prefixes of d given n as step index, i.e. where n does not suffice to verify that d is no prefix of d. An infinite path through τ_K would be a totalisation of d.

Theorem 27. $\mathsf{EPF}_\mathbb{B} \to \mathsf{KT}$

Proof. We show that τ_K is a Kleene tree. That τ_K is a decidable tree is immediate. To show that τ_K is infinite let k be given. We define $f0 := []$ and $f(Sn) := fn + [\mathbf{if}\ Dkn\ \mathbf{is}\ \mathsf{Some}\,x\ \mathbf{then}\ x\ \mathbf{else}\ \mathsf{false}]$. We have $|fn| = n$. In particular, $|fk| \geq k$ and $\tau_K(fk)$.

For well-foundedness let $f : \mathbb{N} \to \mathbb{B}$ be given. There is n s.t. $dn \overset{!}{=} b$ and $fn \neq b$. Thus there is k s.t. $\mathsf{seval}\ (dn)\ k = \mathsf{Some}\,b$. Now $\neg\tau_K u$ for $u := [f0, \dots, f(n+k)]$. ◀

6 Extensionality Axioms

Coq's type theory is intensional, i.e. $f \equiv_{A\to B} g$ and $f = g$ do not coincide. Extensionality properties can however be consistently assumed as axioms. In this section we briefly discuss the relationship between CT and functional extensionality Fext, propositional extensionality Pext and proof irrelevance PI, defined as follows:

$$\begin{aligned}\mathsf{Fext} &:= \forall AB.\forall fg : A \to B.\ (\forall a. fa = ga) \to f = g\\ \mathsf{Pext} &:= \forall PQ : \mathbb{P}.\ (P \leftrightarrow Q) \to P = Q\\ \mathsf{PI} &:= \forall P : \mathbb{P}.\forall (x_1 x_2 : P).\ x_1 = x_2\end{aligned}$$

Fact 28. Pext → PI

Swan and Uemura [43] prove that intensional predicative Martin-Löf type theory remains consistent if CT, the axiom of univalence, and propositional truncation are added. Since functional extensionality and propositional extensionality are a consequence of univalence, and propositions are semantically defined as exactly the irrelevant types, Fext, Pext, and PI hold in this extension of type theory. It seems likely that the consistency result can then be adapted to Coq's type theory, yielding a consistency proof for CT with Fext, Pext, and PI.

It is however crucial to formulate CT using $\exists$ instead of Σ. The formulation as $\mathsf{CT}_\Sigma := \forall f.\ \Sigma n.\ n \sim f$ is inconsistent with functional extensionality Fext, as already observed in [46].

Lemma 29. $\mathsf{CT}_\Sigma \to \mathsf{Fext} \to \bot$

Proof. Since CT_Σ implies EA, it suffices to prove that $\lambda f.\forall n.\ fn = 0$ is decidable by Fact 22. Assume $G : \forall f.\ \Sigma c.\ c \sim f$ and let $Ff :=$ **if** $\pi_1(Gf) = \pi_1(G(\lambda x.0))$ **then** true **else** false.

If $Ff = \mathsf{true}$, then $\pi_1(Gf) = \pi_1(G(\lambda x.0))$ and by extensionality of $\sim$, $fn = (\lambda x.0)n = 0$.

If $\forall n.\ fn = 0$, then $f = \lambda x.\ 0$ by Fext, thus $\pi_1(Gf) = \pi_1(G(\lambda x.\ 0))$ and $Ff = \mathsf{true}$. ◀

7 Classical Logical Axioms

In this section we consider consequences of the law of excluded middle LEM. Precisely, besides LEM, we consider the weak law of excluded middle WLEM, the Gödel-Dummett-Principle DGP[2] and the principle of independence of premises IP, together with their respective restriction of propositions to the satisfiability of boolean functions, resulting in the limited principle of omniscience LPO, the weak limited principle of omniscience WLPO, and the lesser limited principle of omniscience LLPO.

$$\begin{aligned}\mathsf{LEM} &:= \forall P : \mathbb{P}.\ P \lor \neg P & \mathsf{LPO} &:= \forall f : \mathbb{N} \to \mathbb{B}.\ (\exists n.\ fn = \mathsf{true}) \lor \neg(\exists n.\ fn = \mathsf{true})\\ \mathsf{WLEM} &:= \forall P : \mathbb{P}.\ \neg\neg P \lor \neg P & \mathsf{WLPO} &:= \forall f : \mathbb{N} \to \mathbb{B}.\ \neg\neg(\exists n.\ fn = \mathsf{true}) \lor \neg(\exists n.\ fn = \mathsf{true})\\ \mathsf{DGP} &:= \forall PQ : \mathbb{P}.(P \to Q) \lor (Q \to P) & \mathsf{LLPO} &:= \forall fg : \mathbb{N} \to \mathbb{B}.\ ((\exists n.\ fn = \mathsf{true}) \to (\exists n.\ gn = \mathsf{true}))\\ & & &\qquad \lor ((\exists n.\ gn = \mathsf{true}) \to (\exists n.\ fn = \mathsf{true}))\\ \mathsf{IP} &:= \quad \forall P : \mathbb{P}.\forall q : \mathbb{N} \to \mathbb{P}.\ (P \to \exists n. qn) \to \exists n.\ P \to qn\end{aligned}$$

Fact 30. LEM → DGP, DGP → WLEM, LEM → IP.

The converses are likely not provable: Diener constructs a topological model where DGP holds but not LEM, and one where WLEM holds but not DGP [11, Proposition 8.5.3]. Pédrot and Tabareau [36] construct a syntactic model where IP holds, but LEM does not.

[2] We follow Diener [11] in using the abbreviation DGP instead of GDP.

Fact 31. $\mathsf{LPO} \to \mathsf{WLPO}$ *and* $\mathsf{WLPO} \to \mathsf{LLPO}$.

The converses are likely not provable: Both implications are strict over IZF with dependent choice [23, Theorem 5.1].

LPO is Σ^0_1-LEM and WLPO is simultaneously Σ^0_1-WLEM and Π^0_1-LEM, due to the following:

Fact 32. $(\forall n. fn = \mathsf{false}) \leftrightarrow \neg(\exists n. fn = \mathsf{true})$

Both can also be formulated for predicates:

Fact 33. *The following equivalences hold:*

1. $\mathsf{LPO} \quad\ \leftrightarrow \forall X. \forall (p : X \to \mathbb{P}).\ \mathcal{S}p \to \forall x.\ \ px \lor \neg px$
2. $\mathsf{WLPO} \leftrightarrow \forall X. \forall (p : X \to \mathbb{P}).\ \mathcal{S}p \to \forall x. \neg px \lor \neg\neg px$
3. $\mathsf{WLPO} \leftrightarrow \forall X. \forall (p : X \to \mathbb{P}).\ \overline{\mathcal{S}}p \to \forall x.\ \ px \lor \neg px$

In our formulation, LLPO is the Gödel-Dummet rule for Σ^0_1 propositions. It can also be formulated as Σ^0_1 or $\mathcal{S}$ De Morgan rule (**2**, **3** in the following Lemma), $\mathcal{S}$-DGP (**4**), or as a double negation elimination principle on $\overline{\mathcal{S}}$ relations into booleans (**5**):

Lemma 34. *The following are equivalent:*

1. LLPO
2. $\forall fg : \mathbb{N} \to \mathbb{B}.\ \neg((\exists n. fn = \mathsf{true}) \land (\exists n. gn = \mathsf{true})) \to \neg(\exists n. fn = \mathsf{true}) \lor \neg(\exists n. gn = \mathsf{true})$
3. $\forall X. \forall (p\ q : X \to \mathbb{P}).\ \mathcal{S}p \to \mathcal{S}q \to \forall x.\ \neg(px \land qx) \to \neg px \lor \neg qx$
4. $\forall X. \forall (p : X \to \mathbb{P}).\ \mathcal{S}p \to \forall xy.\ (px \to py) \lor (py \to px)$
5. $\forall X. \forall (R : X \to \mathbb{B} \to \mathbb{P}).\ \overline{\mathcal{S}}R \to \forall x.\ \neg\neg(\exists b.\ Rxb) \to \exists b.\ Rxb$
6. $\forall f.\ (\forall nm. fn = \mathsf{true} \to fm = \mathsf{true} \to n = m) \to (\forall n. f(2n) = \mathsf{false}) \lor (\forall n. f(2n+1) = \mathsf{false})$

We define the *principle of finite possibility* as $\mathsf{PFP} := \forall f. \exists g.\ (\forall n.\ fn = \mathsf{false}) \leftrightarrow (\exists n.\ gn = \mathsf{true})$. PFP unifies WLPO and LLPO:

Fact 35. $\mathsf{WLPO} \leftrightarrow \mathsf{LLPO} \land \mathsf{PFP}$

A principle unifying the classical axioms with their counterparts for Σ^0_1 is *Kripke's schema* $\mathsf{KS} := \forall P : \mathbb{P}. \exists f : \mathbb{N} \to \mathbb{B}.\ P \leftrightarrow \exists n.\ fn = \mathsf{true}$:

Fact 36. $\mathsf{LEM} \to \mathsf{KS}$

Fact 37. *Given* KS *we have* $\mathsf{LPO} \to \mathsf{LEM}$, $\mathsf{WLPO} \to \mathsf{WLEM}$, *and* $\mathsf{LLPO} \to \mathsf{DGP}$.

KS could be strengthened to state that every predicate is semi-decidable (to which KS is equivalent using $\mathsf{AC}_{\mathbb{N},\mathbb{N}\to\mathbb{N}}$). The strengthening would be incompatible with CT.

In general, the compatibility of classical logical axioms (without assuming choice principles) with CT seems open. We conjecture that Coq's restriction preventing large elimination principles for non-sub-singleton propositions makes LEM and CT consistent in Coq.

8 Axioms of Russian Constructivism

The Russian school of constructivism morally identifies functions with computable functions, sometimes assuming CT explicitly. Another axiom considered valid is Markov's principle:

$$\mathsf{MP} := \forall f : \mathbb{N} \to \mathbb{B}.\ \neg\neg(\exists n.\ fn = \mathsf{true}) \to \exists n.\ fn = \mathsf{true}$$

Markov's principle is consistent with CT [43] and follows from LPO:

Fact 38. $\mathsf{LPO} \leftrightarrow \mathsf{WLPO} \land \mathsf{MP}$

Corollary 39. $\mathsf{LPO} \to \mathsf{MP}$.

It seems likely that the converse is not provable: There is a logic where MP holds, but not LPO [24]. As observed by Herbelin [24] and Pedrót and Tabareau [36], $\mathsf{IP} \wedge \mathsf{MP}$ yields LPO:

Lemma 40. $\mathsf{MP} \to \mathsf{IP} \to \mathsf{LPO}$

Proof. Given $f : \mathbb{N} \to \mathbb{B}$ there is $n_0 : \mathbb{N}$ s.t. $\forall k.\ fk = \mathsf{true} \to fn_0 = \mathsf{true}$ using MP and IP: By MP, $\neg\neg(\exists k.\ fk = \mathsf{true}) \to \exists n.\ fn = \mathsf{true}$ and by IP, $\exists n. \neg\neg(\exists k. fk = \mathsf{true}) \to fn = \mathsf{true}$, which suffices. Now $fn_0 = \mathsf{true} \leftrightarrow \exists n.\ fn = \mathsf{true}$ and LPO follows. ◀

A nicer factorisation would be to prove $\mathsf{IP} \to \mathsf{WLPO}$, but the implication seems unlikely.

Lemma 41. *The following are equivalent:*

1. MP
2. $\forall X. \forall p : X \to \mathbb{P}.\ \mathcal{S}p \to \forall x.\ \neg\neg px \to px$
3. $\forall X. \forall p : X \to \mathbb{P}.\ \mathcal{S}p \to \mathcal{S}\overline{p} \to \forall x.\ px \vee \neg px$
4. $\forall X. \forall p : X \to \mathbb{P}.\ \mathcal{S}p \to \mathcal{S}\overline{p} \to \mathcal{D}p$
5. $\forall X. \forall (R : X \to \mathbb{B} \to \mathbb{P}).\ \mathcal{S}R \to \forall x.\ \neg\neg(\exists b.\ Rxb) \to \exists b.\ Rxb$

Proof.
- **1** $\to$ **2** is immediate.
- **2** $\to$ **3**: Since $\mathcal{S}$ is closed under disjunctions and since $\neg\neg(px \vee \neg px)$ is a tautology.
- **3** $\to$ **4** is immediate by Lemma 49 with $Rxb := (px \wedge b = \mathsf{true}) \vee (\neg px \wedge b = \mathsf{false})$.
- **4** $\to$ **1**: Let $\neg\neg(\exists n. fn = \mathsf{true})$. Let $p(x : \mathbb{N}) := \exists n. fn = \mathsf{true}$. Now p is semi-decided by $\lambda x. f$, $\overline{p}$ by $\lambda xn.\mathsf{false}$, and $p0 \vee \neg p0$ by **4**. One case is easy, the other contradictory. ◀

Note that **4** is often called "Post's theorem". **1** $\leftrightarrow$ **3** $\leftrightarrow$ **4** is already discussed in [14]. **5** is dual to Lemma 34 (**5**). Replacing $\mathcal{S}p$ with $\overline{\mathcal{S}}p$ in **2** does however not result in an equivalent of LLPO, but turns **2** into an assumption-free fact. While in general $\mathcal{S}\overline{p} \leftrightarrow \overline{\mathcal{S}}p$ does not hold it seems possible that they can be exchanged in **3** and **4**, but we are not aware of a proof.

9 Choice Axioms

We consider the axioms of choice AC, unique choice AUC, dependent choice ADC, and countable choice ACC. $\mathsf{AC}_{\mathbb{N},\mathbb{N}}$ and $\mathsf{AC}_{\mathbb{N}\to\mathbb{N},\mathbb{N}}$ are often called $\mathsf{AC}_{0,0}$ and $\mathsf{AC}_{1,0}$ in the literature.

$$\mathsf{AC}_{X,Y} := \forall R : X \to Y \to \mathbb{P}.(\forall x. \exists y. Rxy) \to \exists f : X \to Y. \forall x.\ Rx(fx)$$

$$\mathsf{AUC}_{X,Y} := \forall R : X \to Y \to \mathbb{P}.(\forall x. \exists! y. Rxy) \to \exists f : X \to Y. \forall x.\ Rx(fx)$$

$$\mathsf{ADC}_{X} := \forall R : X \to X \to \mathbb{P}.(\forall x. \exists x'. Rxx') \to \forall x_0. \exists f : \mathbb{N} \to X. f0 = x_0 \wedge \forall n.\ R(fn)(f(n+1)))$$

$$\mathsf{AC} := \forall XY : \mathbb{T}.\ \mathsf{AC}_{X,Y} \qquad \mathsf{AUC} := \forall XY.\ \mathsf{AUC}_{X,Y} \qquad \mathsf{ADC} := \forall X : \mathbb{T}.\ \mathsf{ADC}_X \qquad \mathsf{ACC} := \forall X : \mathbb{T}.\ \mathsf{AC}_{\mathbb{N},X}$$

Fact 42. $\mathsf{AC}_{X,X} \to \mathsf{ADC}_X$*,* $\mathsf{AC}_{X,Y} \to \mathsf{AUC}_{X,Y}$*,* $\mathsf{ADC} \to \mathsf{ACC}$*,* $\mathsf{ACC} \to \mathsf{AC}_{\mathbb{N},\mathbb{N}}$*, and* $\mathsf{AC}_{\mathbb{N}\to\mathbb{N},\mathbb{N}} \to \mathsf{AC}_{\mathbb{N},\mathbb{N}}$*.*

The following well-known fact is due to Diaconescu [10] and Myhill and Goodman [22]:

Fact 43. $\mathsf{AC} \to \mathsf{Fext} \to \mathsf{Pext} \to \mathsf{LEM}$

Given that $\mathsf{AC}_{\mathbb{N}\to\mathbb{N},\mathbb{N}}$ turns CT into CT_Σ, and that $\mathsf{EA} \leftrightarrow \Sigma T. \mathsf{CT}_T$ we have:

Fact 44. $\mathsf{AC}_{\mathbb{N}\to\mathbb{N},\mathbb{N}} \to \mathsf{Fext} \to \mathsf{EA} \to \bot$

We will later see that $\mathsf{LLPO} \wedge \mathsf{AC}_{\mathbb{N},\mathbb{N}}$ implies weak Kőnig's lemma, which is incompatible with KT. Already now we can prove that $\mathsf{WLPO} \wedge \mathsf{AUC}_{\mathbb{N},\mathbb{B}}$ is incompatible with EA:

Fact 45. $\mathsf{AUC}_{\mathbb{N},\mathbb{B}} \to (\forall n : \mathbb{N}.\ pn \vee \neg pn) \to \mathcal{D}p$

Lemma 46. $\mathsf{WLPO} \to \mathsf{AUC}_{\mathbb{N},\mathbb{B}} \to \mathsf{EA} \to \mathcal{D}\overline{\mathsf{K}_0}$

Proof. WLPO implies $\forall n. \neg \mathsf{K}_0 n \vee \neg\neg \mathsf{K}_0 n$. By $\mathsf{AUC}_{\mathbb{N},\mathbb{B}}$ and the last lemma $\overline{\mathsf{K}_0}$ is decidable. ◀

Corollary 47. $\mathsf{WLPO} \to \mathsf{AUC}_{\mathbb{N},\mathbb{B}} \to \mathsf{EA} \to \bot$

9.1 Provable choice axioms

In contrast to predicative Martin-Löf type theory, Coq's type theory does not prove the axiom of choice, nor the axioms of dependent and countable choice. This is due to the fact that arbitrary large eliminations are not allowed. However, recall that a large elimination principle for the accessibility predicate is provable, resulting in Lemma 1. Using Lemma 1 we can then prove $\mathcal{D}\text{-}\mathsf{AC}_{X,\mathbb{N}}$ for all X, i.e. choice for decidable relations into natural numbers:

Lemma 48. $\forall X.\forall R : X \to \mathbb{N} \to \mathbb{P}.\ \mathcal{D}R \to (\forall x.\exists n.Rxn) \to \exists f : X \to \mathbb{N}.\forall x.\ Rx(fx)$.

As a consequence and with no further reference to Lemma 1 we can then prove choice principles for semi-decidable and enumerable relations, i.e. $\mathcal{S}\text{-}\mathsf{AC}_{X,\mathbb{N}}$ and $\mathcal{E}\text{-}\mathsf{AC}_{\mathbb{N},X}$ for all X:

Lemma 49. *The following two choice principles are provable*[3]*:*

1. $\forall X.\forall R : X \to \mathbb{N} \to \mathbb{P}.\ \mathcal{S}R \to (\forall x.\exists n.\ Rxn) \to \exists f : X \to \mathbb{N}.\forall x.\ Rx(fx)$
2. $\forall X.\forall R : \mathbb{N} \to X \to \mathbb{P}.\ \mathcal{E}R \to (\forall n.\exists x.\ Rnx) \to \exists f : \mathbb{N} \to X.\forall n.\ Rn(fn)$

Principle **2** can be relaxed to arbitrary discrete types instead of $\mathbb{N}$, and in particular $\mathcal{S}\text{-}\mathsf{AC}_{\mathbb{N},\mathbb{B}}$ follows from **1**. In Appendix A we discuss consequences of the here mentioned principles with regards to CT for oracles and in the next section $\overline{\mathcal{S}}\text{-}\mathsf{AC}_{\mathbb{N},\mathbb{B}}$ will be central.

10 Axioms on Trees

We have already introduced (decidable) binary trees and Kleene trees in Section 5. We now give a broader overview and give formulations of LPO, WLPO, LLPO, and MP in terms of decidable binary trees, following Berger et al. [5].

Fact 50. *Let τ be a tree. Then $\tau_u v := \tau(u +\!\!+ v)$ is a tree if and only if τu.*

If τu holds we call τ_u a *subtree* of τ and $\tau_{[b]}$ a *direct subtree* of τ.

Lemma 51. *The following equivalences hold:*

1. $\mathsf{LPO} \leftrightarrow$ *every tree is bounded or infinite.*
2. $\mathsf{WLPO} \leftrightarrow$ *every tree is infinite or not infinite.*
3. $\mathsf{LLPO} \leftrightarrow$ *every infinite tree has a direct infinite subtree.*
4. $\mathsf{MP} \leftrightarrow$ *if a tree is not infinite it is bounded.*
5. $\mathsf{MP} \leftrightarrow$ *if a tree has no infinite path it is well-founded.*

Recall Fact 25 stating that every bounded tree is well-founded and that every tree with an infinite path is infinite. The respective converse implications are known as Brouwer's *fan theorem* FAN and *weak Kőnig's lemma* WKL respectively:

$\mathsf{FAN} :=$ *Every well-founded decidable binary tree is bounded.*

$\mathsf{WKL} :=$ *Every infinite decidable binary tree has an infinite path.*

Fact 52. $\mathsf{KT} \to \neg\mathsf{FAN}$ *and* $\mathsf{KT} \to \neg\mathsf{WKL}$.

Note that FAN is called FAN'_Δ in [26] and FAN_Δ in [11], and WKL is called $\mathsf{WKL}_\mathcal{D}$ in [15]. Ishihara [26] shows how to deduce FAN from WKL constructively:

[3] A formulation of (**1**) for disjunctions (equivalently: $R : X \to \mathbb{B} \to \mathbb{P}$) is due to Andrej Dudenhefner and was received in private communication. (**2**) was anticipated by Larchey-Wendling [30], who formulated it for μ-recursively enumerable instead of synthetically enumerable predicates.

Fact 53. *Bounded trees τ have a longest element, i.e. $\exists u.\ \tau u \wedge \forall v.\ \tau v \to |v| \le |u|$.*

Lemma 54. *For every tree τ there is an infinite tree τ' s.t. for any infinite path f of τ' $\forall u.\ \tau u \to \tau[f0, \dots, f|u|]$.*

Theorem 55. WKL → FAN

Proof. Let τ be well-founded. By Lemma 54 and WKL, there is f s.t. $\forall a.\ \tau u \to \tau[f0, \dots, f|u|]$. Since τ is well-founded there is n s.t. $\neg\tau[f0, \dots, fn]$. Then n is a bound for τ: For u with $|u| > n$ and τu we have $\tau[f0, \dots, fn, \dots, f|u|]$. But then $\tau[f0, \dots, fn]$, contradiction. ◀

Corollary 56. KT → ¬WKL.

Berger and Ishihara [4] show that FAN ↔ WKL!, a restriction of WKL stating that every infinite decidable binary tree with *at most one* infinite path has an infinite path. Schwichtenberg [40] gives a more direct construction and mechanises the proof in Minlog.

Berger, Ishihara, and Schuster [5] characterise WKL as the combination of the logical principle LLPO and the function existence principle $\overline{\mathcal{S}}\text{-}\mathsf{AC}_{\mathbb{N},\mathbb{B}}$ (called $\Pi^0_1\text{-}\mathsf{ACC}^\vee$ in [5]). We observe that WKL can also be characterised as one particular choice or dependent choice principle. The proofs are essentially rearrangements of [5, Theorem 27 and Corollary 5].

Theorem 57. *The following are equivalent:*

1. WKL
2. $\mathsf{LLPO} \wedge \overline{\mathcal{S}}\text{-}\mathsf{AC}_{\mathbb{N},\mathbb{B}}$
3. $\forall R : \quad \mathbb{N} \to \mathbb{B} \to \mathbb{P}.\ \overline{\mathcal{S}}R \to (\forall n.\neg\neg\exists b.Rnb) \to \exists f : \mathbb{N} \to \mathbb{B}.\forall n.\ R\ n\ (fn)$
4. $\forall R : \mathbb{LB} \to \mathbb{B} \to \mathbb{P}.\ \overline{\mathcal{S}}R \to (\forall u.\neg\neg\exists b.Rub) \to \exists f : \mathbb{N} \to \mathbb{B}.\forall n.\ R\ [f0, \dots, f(n-1)]\ (fn)$

Proof. For WKL → LLPO we use the characterisation **3** of LLPO from Lemma 51. Let τ be an infinite tree. By WKL there is an infinite path f. Then $\tau_{[f0]}$ is a direct infinite subtree.

For $\mathsf{WKL} \to \overline{\mathcal{S}}\text{-}\mathsf{AC}_{\mathbb{N},\mathbb{B}}$ let R be total and f s.t. $\forall nb.\ Rnb \leftrightarrow \forall m.fnbm = \mathsf{false}$. Define the tree $\tau u := \forall i < |u|.\forall m < |u|.\ fi(u[i])m = \mathsf{false}$. Infinity of τ follows from $\forall n.\exists u.|u| = n \wedge \forall i < n.Ri(u[i])$, proved by induction on n using totality of R. If g is an infinite path of τ, $Rn(gn)$ follows from $\forall m.\tau[g0, \dots, g(n+m+1)]$.

2 → **3** is immediate using characterisation **3** of LLPO from Lemma 34.

For **3** → **4** let $F : \mathbb{N} \to \mathbb{LB}$ and $G : \mathbb{LB} \to \mathbb{N}$ invert each other.[4] Let $R : \mathbb{LB} \to \mathbb{B} \to \mathbb{P}$ and f be the choice function obtained from **3** for $\lambda nb.R(Fn)b$. Then $\lambda n.f(G(gn))$ where $g0 := []$ and $g(\mathsf{S}n) := gn \mathbin{+\!\!+} [f(G(gn))]$ is a choice function for R as wanted.

For **4** → **1** let τ be an infinite tree and let $d_u m := \exists v.|v| = m \wedge \tau_u v$, i.e. $d_u m$ if τ_u has depth at least m and in particular τ_u is infinite iff $\forall m.d_u m$. Define $Rub := \forall m.d_{u+\!\!+[b]}m \vee \neg d_{u+\!\!+[\neg_{\mathbb{B}} b]}$. R is co-semi-decidable (since d is decidable), and $\neg R\,u\,\mathsf{true} \wedge \neg R\,u\,\mathsf{false}$ is contradictory. Thus **4** yields a choice function f which fulfils $\tau[f0, \dots, fn]$ by induction on n. ◀

11 Continuity: Baire Space, Cantor Space, and Brouwer's Intuitionism

The total function space $\mathbb{N} \to \mathbb{N}$ is often called *Baire space*, whereas $\mathbb{N} \to \mathbb{B}$ is called *Cantor space*. We will from now on write $\mathbb{N}^\mathbb{N}$ and $\mathbb{B}^\mathbb{N}$ for the spaces.

Constructively, one cannot prove that $\mathbb{N}^\mathbb{N}$ and $\mathbb{B}^\mathbb{N}$ are in bijection. However, KT is equivalent to the existence of a continuous bijection $\mathbb{B}^\mathbb{N} \to \mathbb{N}^\mathbb{N}$ with a continuous modulus of continuity, i.e. a modulus function which is continuous (in the point) itself [11]. Furthermore, KT yields a continuous bijection $\mathbb{N}^\mathbb{N} \to \mathbb{B}^\mathbb{N}$ [3].

[4] These so called coding functions is easy to construct even formally using e.g. techniques from [14].

We call a function $F : A^{\mathbb{N}} \to B^{\mathbb{N}}$ *continuous* if $\forall f : A^{\mathbb{N}}.\forall n : \mathbb{N}.\exists L : \mathbb{L}\mathbb{N}.\forall g : A^{\mathbb{N}}.\, (\mathsf{map}\ f\ L = \mathsf{map}\ g\ L) \to Ffn = Fgn$. A function $M : A^{\mathbb{N}} \to \mathbb{N} \to \mathbb{L}\mathbb{N}$ is called the *modulus of continuity* for F if $\forall n : \mathbb{N}.\forall fg : A^{\mathbb{N}}.\ \mathsf{map}\ f\ (Mfn) = \mathsf{map}\ g\ (Mfn) \to Ffn = Fgn$. We define:

$$\mathsf{Homeo}(A^{\mathbb{N}}, B^{\mathbb{N}}) := \exists F : A^{\mathbb{N}} \to B^{\mathbb{N}}.\exists M.\ M \textit{ is a continuous modulus of continuity for } F$$

We start by proving that $\mathsf{KT} \leftrightarrow \mathsf{Homeo}(\mathbb{B}^{\mathbb{N}}, \mathbb{N}^{\mathbb{N}})$. To do so, we say that $u +\!\!+ [b]$ is a *leaf of a Kleene tree* τ_K if $\tau_K u$, but $\neg\tau_K(u +\!\!+ [b])$.

Fact 58. *For every τ_K, there is an injective enumeration $\ell : \mathbb{N} \to \mathbb{L}\mathbb{B}$ of the leaves of τ_K.*

We define $F(f : \mathbb{N} \to \mathbb{N})n := (\ell(f0) +\!\!+ \cdots +\!\!+ \ell(f(n+1)))[n]$. Since leaves cannot be empty, the length of the accessed list is always larger than n and F is well-defined.

Lemma 59. *F is injective w.r.t. $\equiv_{\mathbb{N}^{\mathbb{B}}}$ and $\equiv_{\mathbb{N}^{\mathbb{N}}}$.*

Lemma 60. *F is continuous with continuous modulus of continuity.*

Lemma 61. *The following hold for a Kleene tree τ_K:*
1. *There is a function $\ell^{-1} : \mathbb{L}\mathbb{B} \to \mathbb{N}$ s.t. for all leafs l, $\ell(\ell^{-1} l) = l$.*
2. *For all l s.t. $\neg\tau_K l$ there exists $l' \sqsubseteq l$ s.t. l' is a leaf of τ_K.*
3. *There is $\mathsf{pref}\ : (\mathbb{N} \to \mathbb{B}) \to \mathbb{L}\mathbb{B}$ s.t. $\mathsf{pref}\,g$ is a leaf of τ_K and $\exists n.\ \mathsf{pref}\,g = \mathsf{map}\ g\ [0, \ldots, n]$.*

We can now define the inverse as $G\ g\ n := \ell^{-1}(\mathsf{pref}\ (\mathsf{nxt}^n g))$ where $\mathsf{nxt}\ g\ n := g(n + |\mathsf{pref}\ g|)$.

Lemma 62. *$F\ (G\ g) \equiv_{\mathbb{N}\to\mathbb{B}} g$*

Lemma 63. *G is continuous with continuous modulus of continuity.*

The following proof is due to Diener [11, Proposition 5.3.2].

Lemma 64. $\mathsf{Homeo}(\mathbb{B}^{\mathbb{N}}, \mathbb{N}^{\mathbb{B}}) \to \mathsf{KT}$

Proof. Let F be a bijection with continuous modulus of continuity M. Then $\tau u := \forall 0 < i \leq |u|.\exists k < i.k \in M(\lambda n.\textbf{if}\ l[n]\ \textbf{is}\ \mathsf{Some}\,b\ \textbf{then}\ b\ \textbf{else}\ \mathsf{false})\ 0$ is a Kleene tree. ◀

Theorem 65. $\mathsf{KT} \leftrightarrow \mathsf{Homeo}(\mathbb{B}^{\mathbb{N}}, \mathbb{N}^{\mathbb{N}})$ *and* $\mathsf{KT} \to \mathsf{Homeo}(\mathbb{N}^{\mathbb{N}}, \mathbb{B}^{\mathbb{N}})$.

Deiser [9] proves in a classical setting that $\mathsf{Homeo}(\mathbb{N}^{\mathbb{N}}, \mathbb{B}^{\mathbb{N}})$ holds. It would be interesting to see whether the proof can be adapted to a constructive proof $\mathsf{WKL} \to \mathsf{Homeo}(\mathbb{N}^{\mathbb{N}}, \mathbb{B}^{\mathbb{N}})$.

We have already seen that CT is inconsistent with FAN. Besides FAN, in Brouwer's intuitionism the continuity of functionals $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ is routinely assumed:

$$\mathsf{Cont} := \forall F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}.\ \forall f : \mathbb{N} \to A.\exists L : \mathbb{L}\mathbb{N}.\forall g : \mathbb{N} \to A.\ (\mathsf{map}\ f\ L = \mathsf{map}\ g\ L) \to Ff \equiv_B Fg$$

Since every computable function is continuous, we believe Cont to be consistent with CT. Combining Cont with $\mathsf{AC}_{\mathbb{N}\to\mathbb{N},\mathbb{N}}$ yields *Brouwer's continuity principle*[5] called $\mathsf{WC\text{-}N}$ in [46]:

$$\mathsf{WC\text{-}N} := \forall R : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{P}.(\forall f.\exists n.Rfn) \to \forall f.\exists Ln.\forall g.\ \mathsf{map}\ f\ L = \mathsf{map}\ g\ L \to Rgn$$

[5] But note that $\mathsf{Cont} \to \mathsf{AC}_{\mathbb{N}\to\mathbb{N},\mathbb{N}} \to \bot$, since the resulting modulus of continuity function allows for the construction of a non-continuous function [13].

Theorem 66. $\mathsf{WC\text{-}N} \to \mathsf{Cont}$

$\mathsf{WC\text{-}N}$ is inconsistent with CT, since the computability relation $\sim$ is not continuous:

Theorem 67. $\mathsf{WC\text{-}N} \to \mathsf{CT} \to \bot$

Proof. Recall that if two functions have the same code they are extensionally equal. By CT, $\lambda fc.c \sim f$ is a total relation. Using $\mathsf{WC\text{-}N}$ for this relation and $\lambda x.\ 0$ yields a list L and a code c s.t. $\forall g.\ \mathsf{map}\ g\ L = [0, \ldots, 0] \to c \sim g$.

The functions $\lambda x.\ 0$ and $\lambda x.\ \textbf{if}\ x \in L\ \textbf{then}\ 0\ \textbf{else}\ 1$ both fulfil the hypothesis and thus have the same code – a contradiction since they are not extensionally equal. ◀

12 Conclusion

In this paper we surveyed the known connections of axioms in Coq's type theory, a constructive type theory with a separate, impredicative universe of propositions, with a special focus on Church's thesis CT and formulations of axioms in terms of notions of synthetic computability. Furthermore, all results are mechanised in the Coq proof assistant.

In constructive mathematics, countable choice is often silently assumed, as critised e.g. by Richman [38, 39]. In contrast, constructive type theory with a universe of propositions seems to be a suitable base system for matters of constructive (reverse) mathematics sensitive to applications of countable choice. Due to the separate universe of propositions, such a constructive type theory neither proves countable nor dependent choice, allowing equivalences like the one in Theorem 57 to be stated sensitively to choice. We conjecture that Lemma 49 deducing $\mathcal{S}\text{-}\mathsf{AC}_{X,\mathbb{N}}$ and $\mathcal{E}\text{-}\mathsf{AC}_{\mathbb{N},X}$ directly from $\mathcal{D}\text{-}\mathsf{AC}_{X,\mathbb{N}}$ cannot be significantly strengthened. The proof of $\mathcal{D}\text{-}\mathsf{AC}_{X,\mathbb{N}}$ in turn crucially relies on a large elimination principle for $\exists n.\ fn = \mathtt{true}$ (Lemma 1). The theory of [5] proves $\mathcal{D}\text{-}\mathsf{AC}_{\mathbb{N},\mathbb{B}}$ and thus likely also $\mathcal{S}\text{-}\mathsf{AC}_{\mathbb{N},\mathbb{B}}$.

Predicative Martin-Löf type theory proves AC and type theories with propositional truncation and a semantic notion of (homotopy) propositions prove $\mathsf{AUC}_{\mathbb{N},\mathbb{B}}$, thus LEM suffices to disprove CT for both these flavours of type theory. Based on the current state of knowledge in the literature it seems likely that $\mathcal{S}\text{-}\mathsf{AC}_{\mathbb{N},\mathbb{B}}$ and LEM together do not suffice to disprove CT, which seems to require at least classical logic of the strength of LLPO and a choice axiom for *co*-semi-decidable predicates. Thus we conjecture that a consistency proof of e.g. $\mathsf{LEM} \land \mathsf{CT}$ might be possible for Coq's type theory.

Another advantage of basing constructive investigations on constructive type theory is that implementations of type theory in proof assistants already exist. For this paper, mechanising the results in Coq was tremendously helpful in keeping track of all details. For example, many of the presented proofs are very sensitive to small changes in formulations, and Coq actually helped in understanding the proofs and getting them right.

Besides consistency, another interesting property of axioms is admissibility. For instance, Pédrot and Tabareau [36] prove MP admissible in constructive type theory. CT seems to be admissible in constructive type theory in the sense that for every defined function $f : \mathbb{N} \to \mathbb{N}$ one can define a program in a model of computation with the same input output behaviour, as witnessed by the certifying extraction for a fragment of Coq to the λ-calculus [16]. An admissibility proof of CT could then serve as a theoretical underpinning of the Coq library of undecidability proofs [19]. However, any formal admissibility proof would have to deal with the intricacies of Coq's type theory. It would be interesting to investigate whether Letouzey's semantic proof for the correctness of type and proof erasure [33] can be connected with the mechanisation of meta-theoretical properties of Coq's type theory [41] in the MetaCoq project [42], yielding a mechanised admissibility proof for CT in Coq's type theory.

References

1 Andrej Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006.

2 Andrej Bauer. König's lemma and Kleene tree. *unpublished notes*, 2006.

3 Michael J. Beeson. *Foundations of constructive mathematics: Metamathematical studies*, volume 6. Springer Verlag, 1987.

4 Josef Berger and Hajime Ishihara. Brouwer's fan theorem and unique existence in constructive analysis. *Mathematical Logic Quarterly*, 51(4):360–364, 2005.

5 Josef Berger, Hajime Ishihara, and Peter Schuster. The weak König lemma, Brouwer's fan theorem, De Morgan's law, and dependent choice. *Reports on Mathematical Logic*, (47):63, 2012.

6 Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

7 Douglas Bridges and Fred Richman. *Varieties of constructive mathematics*, volume 97. Cambridge University Press, 1987.

8 T. Coquand. Metamathematical investigations of a calculus of constructions. Technical Report RR-1088, INRIA, September 1989. URL: `https://hal.inria.fr/inria-00075471`.

9 Oliver Deiser. A simple continuous bijection from natural sequences to dyadic sequences. *The American Mathematical Monthly*, 116(7):643–646, 2009.

10 Radu Diaconescu. Axiom of choice and complementation. *Proceedings of the American Mathematical Society*, 51(1):176–178, 1975.

11 Hannes Diener. Constructive Reverse Mathematics. *arXiv:1804.05495 [math]*, April 2020. `arXiv:1804.05495`.

12 Martín H. Escardó and Cory M. Knapp. Partial Elements and Recursion via Dominances in Univalent Type Theory. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.CSL.2017.21`.

13 Martín Hötzel Escardó and Chuangjie Xu. The inconsistency of a Brouwerian continuity principle with the Curry–Howard interpretation. In *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

14 Yannick Forster, Dominik Kirst, and Gert Smolka. On synthetic undecidability in Coq, with an application to the Entscheidungsproblem. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 38–51, 2019.

15 Yannick Forster, Dominik Kirst, and Dominik Wehr. Completeness theorems for first-order logic analysed in constructive type theory (extended version). *arXiv preprint arXiv:2006.04399*, 2020.

16 Yannick Forster and Fabian Kunze. A Certifying Extraction with Time Bounds from Coq to Call-By-Value Lambda Calculus. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ITP.2019.17`.

17 Yannick Forster, Fabian Kunze, and Maximilian Wuttke. Verified programming of Turing machines in Coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 114–128, 2020.

18 Yannick Forster and Dominique Larchey-Wendling. Certified undecidability of intuitionistic linear logic via binary stack machines and minsky machines. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 104–117, 2019.

19 Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. A Coq library of undecidable problems. In *The Sixth International Workshop on Coq for Programming Languages (CoqPL 2020).*, 2020. URL: `https://github.com/uds-psl/coq-library-undecidability`.

20 Yannick Forster and Gert Smolka. Weak call-by-value lambda calculus as a model of computation in coq. In *International Conference on Interactive Theorem Proving*, pages 189–206. Springer, 2017.

21 Yannick Forster and Gert Smolka. Call-by-value lambda calculus as a model of computation in Coq. *Journal of Automated Reasoning*, 63(2):393–413, 2019.

22 Noah Goodman and John Myhill. Choice implies excluded middle. *Mathematical Logic Quarterly*, 24(25-30):461–461, 1978. `doi:10.1002/malq.19780242514`.

23 Matt Hendtlass and Robert Lubarsky. Separating fragments of WLEM, LPO, and MP. *The Journal of Symbolic Logic*, 81(4):1315–1343, December 2016. `doi:10.1017/jsl.2016.38`.

24 Hugo Herbelin. An intuitionistic logic that proves Markov's principle. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 50–56. IEEE, 2010.

25 Hajime Ishihara. Reverse mathematics in Bishop's constructive mathematics. *Philosophia Scientiæ. Travaux d'histoire et de philosophie des sciences*, (CS 6):43–59, 2006.

26 Hajime Ishihara. Weak König's lemma implies Brouwer's fan theorem: a direct proof. *Notre Dame Journal of Formal Logic*, 47(2):249–252, 2006.

27 Stephen Cole Kleene. On the interpretation of intuitionistic number theory. *The journal of symbolic logic*, 10(4):109–124, 1945.

28 Stephen Cole Kleene. Recursive functions and intuitionistic mathematics, 1953.

29 Georg Kreisel. Church's thesis: a kind of reducibility axiom for constructive mathematics. In *Studies in Logic and the Foundations of Mathematics*, volume 60, pages 121–150. 1970.

30 Dominique Larchey-Wendling. Typing total recursive functions in Coq. In *International Conference on Interactive Theorem Proving*, pages 371–388. Springer, 2017.

31 Dominique Larchey-Wendling and Yannick Forster. Hilbert's tenth problem in Coq. *arXiv preprint arXiv:2003.04604*, 2020.

32 Dominique Larchey-Wendling and Jean-François Monin. The Braga method: Extracting certified algorithms from complex recursive schemes in Coq. In Klaus Mainzer, Peter Schuster, and Helmut Schwichtenberg, editors, *Proof and Computation: From Proof Theory and Univalent Mathematics to Program Extraction and Verification*. World Scientific Singapore, 2021.

33 Pierre Letouzey. *Programmation fonctionnelle certifiée: l'extraction de programmes dans l'assistant Coq*. PhD thesis, L'Université de Paris-Sud, July 2004. URL: `http://www.pps.jussieu.fr/~letouzey/download/these_letouzey.pdf`.

34 Andrei Andreevich Markov. The theory of algorithms. *Trudy Matematicheskogo Instituta Imeni VA Steklova*, 42:3–375, 1954.

35 Christine Paulin-Mohring. Inductive definitions in the system Coq rules and properties. In *International Conference on Typed Lambda Calculi and Applications*, pages 328–345. Springer, 1993.

36 Pierre-Marie Pédrot and Nicolas Tabareau. Failure is not an option. In *European Symposium on Programming*, pages 245–271. Springer, 2018.

37 Fred Richman. Church's thesis without tears. *The Journal of symbolic logic*, 48(3):797–803, 1983.

38 Fred Richman. The fundamental theorem of algebra: a constructive development without choice. *Pacific Journal of Mathematics*, 196(1):213–230, 2000.

39 Fred Richman. Constructive Mathematics without Choice. In Peter Schuster, Ulrich Berger, and Horst Osswald, editors, *Reuniting the Antipodes — Constructive and Nonstandard Views of the Continuum*, pages 199–205. Springer Netherlands, Dordrecht, 2001. `doi:10.1007/978-94-015-9757-9_17`.

40 Helmut Schwichtenberg. A direct proof of the equivalence between Brouwer's fan theorem and König's lemma with a uniqueness hypothesis. *J. UCS*, 11(12):2086–2095, 2005.

41 Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The MetaCoq Project. *Journal of Automated Reasoning*, February 2020. `doi:10.1007/s10817-019-09540-0`.

42 Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. Coq Coq correct! verification of type checking and erasure for Coq, in Coq. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–28, 2019.

43 Andrew Swan and Taichi Uemura. On Church's thesis in cubical assemblies. *arXiv preprint arXiv:1905.03014*, 2019.

44 The Coq Development Team. The Coq Proof Assistant, version 8.11.0. `https://doi.org/10.5281/zenodo.3744225`, jan 2020. `doi:10.5281/zenodo.3744225`.

45 The Coq std++ Team. An extended "standard library" for Coq. `https://gitlab.mpi-sws.org/iris/stdpp`, 2020.

46 Anne Sjerp Troelstra and Dirk van Dalen. Constructivism in mathematics. vol. i, volume 121 of. *Studies in Logic and the Foundations of Mathematics*, 26, 1988.

47 Benjamin Werner. Sets in types, types in sets. In *International Symposium on Theoretical Aspects of Computer Software*, pages 530–546. Springer, 1997.

A Modesty and Oracles

Using $\mathcal{D}\text{-}\mathsf{AC}_{\mathbb{N},\mathbb{N}}$ from Lemma 48 allows proving a choice axiom w.r.t. models of computation, observed by Larchey-Wendling [30] and called "modesty" by Forster and Smolka [20].

▶ **Lemma 68.** *Let T be an abstract computation function. We have*

$$\forall c.(\forall n.\exists mk.\ Tcnk = \mathsf{Some}\, m) \to \exists f : \mathbb{N} \to \mathbb{N}.\forall n.\exists k.\ Tcnk = \mathsf{Some}\,(fn)$$

That is, if c is the code of a function inside the model of computation which is provably total, the total function can be computed outside of the model. This modesty principle simplifies the mechanisation of computability theory in type theory as e.g. in [21]. For instance, it allows to prove that defining decidability as "a total function in the model of computation deciding the predicate" and as "a meta-level function deciding the predicate which is computable in the model of computation" is equivalent.

However, the modesty principle prevents synthetic treatments of computability theory based on oracles. Traditionally, computability theory based on oracles is formulated using a computability function T_p, s.t. for $p : \mathbb{N} \to \mathbb{P}$ there exists a code c_p representing a total function s.t. $\forall n.(\exists k.Tc_pnk = \mathsf{Some}\, 0) \leftrightarrow pn$.

Synthetically, we would now like to assume an abstract computability function for every p as "Church's thesis with oracles". "Church's thesis with oracles" implies CT, and we know that under CT the predicate K_0 is not decidable. However, under the presence of $\mathcal{D}\text{-}\mathsf{AC}_{\mathbb{N},\mathbb{N}}$ we can use T_{K_0} and obtain c_{K_0} which can be turned into a decider $f : \mathbb{N} \to \mathbb{B}$ for K_0 using the choice principle above – a contradiction.

B Coq mechanisation

The Coq mechanisation of the paper comprises 4250 lines of code, with 3300 lines of proofs and 950 lines of statements and definitions, i.e. 77% proofs. The mechanisation is based on the Coq-std++ library [45], plus around 1500 additional lines of code with custom extensions to Coq's standard library which are shared with the Coq library of undecidability proofs [19].

The 4250 lines of the main development are distributed as follows: The basics of synthetic computability (decidablility, semi-decidability, enumerability, many-one reductions) need 1150 lines of code. The mechanisation of Section 3, covering CT, EA, and EPF, comprises 400 lines of code. 120 lines of codes are needed for the undecidability results of Section 4. Section 5 and Section 10, covering trees and in particular Kleene trees, need 1000 lines of code. Section 11 on continuity is mechanised in 800 lines. The rest, i.e. Sections 6 to 9, needs 750 lines of code.

No advanced mechanisation techniques were needed. Discreteness and enumerability proofs for types were eased using type classes to assemble proofs for compound types such as $\mathbb{LB} \times \mathbb{ON}$, as already done in [14]. Defining the notions of $\equiv_{A \to B}$, $\equiv_{A \to \mathbb{P}}$, and so on was made possible by using type classes as well.

The technically most challenging mechanised proofs correspond to Lemmas 59 - 63, i.e. prove $\mathsf{KT} \to \mathsf{Homeo}(\mathbb{B}^{\mathbb{N}}, \mathbb{N}^{\mathbb{N}}) \wedge \mathsf{Homeo}(\mathbb{N}^{\mathbb{N}}, \mathbb{B}^{\mathbb{N}})$. For these proofs, lots of manipulation of prefixes of lists was needed, and while the functions `firstn` and `dropn` are defined in Coq's standard library, the very useful lemmas of Coq-std++ where needed to make the proofs feasible.

In the development of this paper, the Coq proof assistant, while also acting as proof checker, was truly used as an assistant: Lots of proofs were developed and understood directly while working in Coq rather than on paper, allowing to identify for instance the equivalent characterisations of LLPO, MP, and WKL as in Lemma 34 (**5**), Lemma 41 (**5**), and Theorem 57 (**3**,**4**), which are hard to observe on paper because lots of bookkeeping for side-conditions would have to be done manually then.

Computing Measure as a Primitive Operation in Real Number Computation

Christine Gaßner
Universität Greifswald, Germany
gassnerc@uni-greifswald.de

Arno Pauly
Department of Computer Science, Swansea University, UK
https://www.cs.swan.ac.uk/~cspauly/
arno.m.pauly@gmail.com

Florian Steinberg
INRIA, Sophia Antipolis, France
fsteinberg@gmail.com

Abstract

We study the power of BSS-machines enhanced with abilities such as computing the measure of a BSS-decidable set or computing limits of BSS-computable converging sequences. Our variations coalesce into just two equivalence classes, each of which also can be described as a lower cone in the Weihrauch degrees.

We then classify computational tasks such as computing the measure of Δ^0_2-set of reals, integrating piece-wise continuous functions and recovering a continuous function from an $L_1([0,1])$-description. All these share the Weihrauch degree lim.

2012 ACM Subject Classification Theory of computation → Abstract machines; Theory of computation → Turing machines; Mathematics of computing → Point-set topology; Mathematics of computing → Integral calculus

Keywords and phrases BSS-machine, Weihrauch reducibility, integrable function, Lebesgue measure, computable analysis

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.22

1 Introduction

Computing over abstract data types using register machines makes the fully realistic notion of computability that computable analysis uses more approachable and easier to use in applications. This is well known and the central topic of work such as [9, 38]. While the models we consider in this paper can compute non-computable functions, they still allow us to discuss algorithms in a general sense as they are used e.g. in numerical analysis/scientific computing (cf. [30, 25]). We report on two separate but related investigations.

In the first, we consider extensions of the Blum-Shub-Smale (BSS) machines [1] for computing functions of type $f : \mathbb{R}^* \to \mathbb{R}^*$. We explore the strength of a machine that can compute the measure of a BSS-decidable set in a single step. A concrete inspiration for our operations is found in the ν-operator studied by Moschovakis [29]. The ability to compute the measure of a decidable set can also be seen as an analogue to counting classes such as $\sharp$P in the context of BSS-computation – except that here, we gain computability-theoretic strength, rather than just efficiency[1].

We also consider adding a command that returns the limit of a BSS-computable converging sequence as a primitive. It turns out that all our enhanced BSS-machines can already compute all real functions computable in the sense of computable analysis. We can therefore use the

[1] This is a different approach to the one taken by Bürgisser and Cucker [13], though.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 22; pp. 22:1–22:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

framework of Weihrauch reducibility [8] to describe the resulting computational strength. The usefulness of Weihrauch reducibility to study algebraic computation models had already been observed in [30]. More generally, the use of topological concepts in this context was pioneered in [18].

Our second line of investigation looks into representations (in the sense of computable analysis) of real function classes, continuing a programme initiated in [35]. For example, if we merely have the information on a function $f\colon \mathbb{R} \to \mathbb{R}$ befitting an integrable function, but we know f to actually be continuous, how complicated is it to obtain the information about f as a continuous function? Classically, we know that any measurable function $f\colon [0,1] \to [0,1]$ is already integrable, but already for Δ^0_2-measurable functions this implication no longer holds computably. This is a connecting point to our exploration of BSS-machines, as BSS-computable functions are Δ^0_2-measurable, and the representation of integrable functions essentially allows us to compute all integrals of this function. Again, Weihrauch reducibility will be the framework we use to describe the complexity of these translations.

Overview of the paper

Our results are collated in three theorems that can be found in Section 3.1 and in the introduction of Section 4 respectively. The paper is structured such that either of the Sections 3 and 4 can be read independently by a reader familiar with the content of Section 2.

The general outline of the paper is as follows: In Section 2 we provide the necessary background on computable analysis and Weihrauch reducibility. A reader already familiar with these topics may skip this part.

Section 3 starts off by recollecting the model of computing with generalized register machines whose registers can hold properly infinite objects. Section 3.1 states our two main theorems about such algebraic models of computation: Each of Theorem 15 and Theorem 16 describes an equivalence class of computational models. The remainder of the section is devoted to their proofs.

Section 4 is about measurability and integrability and states our main result Theorem 24 right away. The theorem says that the Weihrauch degree of various operations that correspond to finding the measure of a set or the integral of a function is lim. The rest of the section explains and proves the individual parts of theorem.

2 Background from computable analysis

Let us start with some computable analysis as its viewpoint is important throughout the paper. Introductionary sources on computable analysis that go way beyond what is the scope of this paper include the seminal textbook [40]. A briefer introduction can be found in [10], an approach in a language close to that of this paper in [31].

In computable analysis computations on abstract structures like the real numbers $\mathbb{R}$, are carried out by means of representations. A **representation** of a set X is a partial surjective mapping $\delta\colon\subseteq \mathbb{N}^{\mathbb{N}} \to X$ from Baire space to that set. A representation may be understood to assign to each element x of the mathematical structure X a set $\delta^{-1}(x) \subseteq \mathbb{N}^{\mathbb{N}}$ of names. Each name of an abstract object $x \in X$ encodes this object by providing on demand information about it. A **represented space** is a pair $\mathbf{X} = (X, \delta_{\mathbf{X}})$ of a set X and a representation $\delta_{\mathbf{X}}\colon\subseteq \mathbb{N}^{\mathbb{N}} \to X$ of that set. For convenience we often replace the two copies of natural numbers in Baire space with countable sets that come with canonical enumerations.

▶ **Example 1** (The Cauchy representation)**.** Let (M, d) be a separable metric space with a distinguished countable dense subset D. The **Cauchy representation** assigns a mapping $p\colon \mathbb{Q}^+ \to D$ as name to $x \in M$ if each $p(\varepsilon)$ is an ε-approximation to x in that $d(x, p(\varepsilon)) \leq \varepsilon$.

The most important instance of this construction is $\mathbb{R}$ with the standard enumeration of the rationals. Other metric spaces in Cauchy representation that we encounter are the continuous functions on the unit interval $\mathcal{C}([0,1])$ and the space $\mathrm{L}_1([0,1])$ of equivalence classes of integrable functions.

Baire space comes with a natural topology and any represented space can be equipped with the final topology of its representation. The notion of **admissibility** of a representation, informally spoken, states that we can identify the represented space with the induced topological space.

Let us consider two examples of represented spaces whose induced topology is not metrizable and that can therefore not be constructed using a Cauchy representation. The first example is a finite non-discrete space that we heavily use:

▶ **Example 2** (Sierpiński space)**.** Sierpiński space $\mathbb{S}$ is the set $\{\top, \bot\}$ equipped with the total representation $\delta_{\mathbb{S}}$ that assigns some $p\colon \mathbb{N} \to \mathbb{B}$ (where $\mathbb{B}$ is the discrete two point space) as name to $\top$ if and only if $\exists n, p(n) = 1$.

The representation of Sierpiński space is admissible and induces the topology where $\{\top\}$ is open but $\{\bot\}$ is not. Thus, the continuous functions from any topological space to Sierpiński space are exactly the characteristic functions of the open sets.

The other example is a representation of the real numbers that provides strictly less information than the Cauchy representation we usually use.

▶ **Example 3** (The lower reals)**.** The lower reals $\mathbb{R}_<$ is the set of real numbers equipped with the representation where a bounded sequence $p\colon \mathbb{N} \to \mathbb{Q}$ of rationals is a name of its supremum $x := \sup\{p(n) \mid n \in \mathbb{N}\}$. Access to a pair of a name of $x \in \mathbb{R}_<$ and one of $-x \in \mathbb{R}_<$ is the same as having access to a name of x in the space $\mathbb{R}$ with the Cauchy representation.

We consider multifunctions between represented spaces as abstractions of computational tasks. A multifunction $f\colon \mathbf{X} \rightrightarrows \mathbf{Y}$ assigns to each $x \in \mathbf{X}$ a set $f(x) \subseteq \mathbf{Y}$ of eligible return values. The domain of f is the set of those x for which $f(x)$ is non-empty and the associated task is to produce from an x and provided that $x \in \mathrm{dom}(f)$ some $y \in f(x)$.

Computability and continuity of multivalued functions between represented spaces is defined via realizers: Some $F\colon \subseteq \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ is a realizer of a multi-valued function $f\colon \mathbf{X} \rightrightarrows \mathbf{Y}$ if it carries names of the input to names of eligible return values, i.e. $\delta_{\mathbf{X}}(p) = x$ implies $F(p)$ is defined and $\delta_{\mathbf{Y}}(F(p)) \in f(x)$. A multivalued function is called **computable** if it has a computable realizer and **continuously realizable** if it has a continuous realizer. Admissibility of the target space implies that a function is topologically continuous if and only if it is continuously realizable. Computability on Baire space is defined as existence of an oracle machine that computes the return-value whenever the input is taken as oracle.

A representation δ is called **computably translatable** to another representation δ' of the same set X, if the identity function is computable as a function from (X, δ) to (X, δ'). Unfolded this means that there exists a **translation**, i.e. a computable $T\colon \subseteq \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ such that whenever p is a name of x in the first representation, $T(p)$ is a name of x in the second representation. Intuitively, existence of a computable translation means that a name in the input representation provides at least as much information about the object as a name in the output representation. If computable translations in both directions exist the representations are called **equivalent** and the identity function is an isomorphism between the spaces. Moreover we say that a representation is **minimal with some property** if any other representation with this property can be computably translated to it.

Finally, a metric space with a distinguished dense sequence is called a **computable metric space** if the metric is computable with respect to the Cauchy representation on itself and on the reals. All metric spaces we encounter in this paper are computable metric spaces.

2.1 Products and exponentials of represented spaces

Given represented spaces $\mathbf{X}$ and $\mathbf{Y}$ there are standard ways to construct new represented spaces. To equip the Cartesian product of the underlying spaces with a representation fix a standard pairing function $\langle \cdot, \cdot \rangle$ of the natural numbers and lift this pairing function to Baire space by $\langle p, q\rangle(n) := \langle p(n), q(n)\rangle$. A name of $(x, y) \in \mathbf{X} \times \mathbf{Y}$ is a pair of names of x and y respectively. To each $A \subseteq \mathbf{X}$ we assign a represented space $\mathbf{A_X}$ by equipping A with the corestriction of the representation of $\mathbf{X}$ to A.

Finally, we use the function space construction of computable analysis. Namely for represented spaces $\mathbf{X}$ and $\mathbf{Y}$ the space $\mathcal{C}(\mathbf{X}, \mathbf{Y})$ of continuously realizable functions.

If $\mathbf{Y}$ is admissible, the continuously realizable functions are exactly the continuous functions. The computable elements of $\mathcal{C}(\mathbf{X}, \mathbf{Y})$ are exactly the computable functions. The function space representation is minimal with the property that evaluation is computable.

The representation is not easy to work with but there often exist simpler equivalent representations.

▶ **Example 4** (Concrete representations for spaces of functions)**.** The computable Weierstraß approximation theorem can be understood to say that $\mathcal{C}([0, 1]) \simeq \mathcal{C}([0, 1], \mathbb{R})$. Here, $\mathcal{C}([0, 1], \mathbb{R})$ is the space of continuous functions with the function space representation while $\mathcal{C}([0, 1])$ has the same underlying set but in Cauchy representation with respect to the supremum norm and the rational polynomials as dense subset. More specifically the metric is given by $d(f, g) := \|f - g\|_\infty$, where $\|f\|_\infty := \sup\{|f(x)| \mid x \in [0, 1]\}$ is the supremum norm.

Finally, for a represented space $\mathbf{X}$ we use the space $\mathbf{X}^*$ of finite lists over $\mathbf{X}$. A name of such a finite word takes the same kind of questions that one can ask about elements of $\mathbf{X}$ but returns a finite list of answers for each of the elements of the list.

2.2 Weihrauch reducibility and Weihrauch degrees

A reasonable notion of comparison of computational tasks f and g is to ask whether a solution to f can be specified if – in addition to computable operations – an arbitrary solution to g may be involved once. Weihrauch reductions make this idea formal. Recall that $\langle p, q\rangle$ is the pairing of elements p and q of Baire space.

▶ **Definition 5.** *Let f and g be multivalued functions between represented spaces. A **Weihrauch reduction** of f to g is a pair of computable functions $H, K\colon \mathbb{N}^\mathbb{N} \to \mathbb{N}^\mathbb{N}$ called **pre-** and **post-processor** such that for any realizer G of g the function $p \mapsto K(\langle G(H(p)), p\rangle)$ is a realizer of f.*

We write $f \leq_\mathrm{W} g$ if there exists a Weihrauch reduction and use $f \equiv_\mathrm{W} g$ as abbreviation for $f \leq_\mathrm{W} g \wedge g \leq_\mathrm{W} f$. The **Weihrauch degree** of f is its equivalence class under Weihrauch reductions.

For additional information on the theory of Weihrauch degrees we refer the reader to [8].

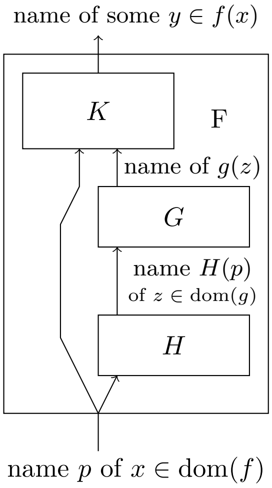

name of some $y \in f(x)$
K
F
name of $g(z)$
G
name $H(p)$
of $z \in \mathrm{dom}(g)$
H
name p of $x \in \mathrm{dom}(f)$

Figure 1 $f \leq_{\mathrm{W}} g$.

▶ **Example 6** (Weihrauch degrees). The following Weihrauch degrees are important to us:

LPO: By $\mathrm{LPO}\colon \mathbb{N}^{\mathbb{N}} \to \mathbb{B}$ we denote the characteristic map of the singleton set containing the constant zero function and its Weihrauch degree. The name is short for 'lesser principle of omniscience' and originates from constructive mathematics, where this principle often serves as a Brouwerian counterexample to show that statements are not constructively provable. Other representatives of LPO include the mapping from $\mathbb{S}$ to $\mathbb{B}$ that sends $\top$ to 1 and $\bot$ to 0 and checking equality of real numbers.

lim and $\mathrm{C}_{\mathbb{N}}$: For a represented space $\mathbf{X}$ consider the multivalued function $\lim_{\mathbf{X}}\colon \mathbf{X}^{\mathbb{N}} \rightrightarrows \mathbf{X}$ that sends a sequence $(x_n)_{n\in\mathbb{N}}$ to the set $\{x \mid \lim_{n\to\infty} x_n = x\}$ of its limit points. It holds that $\lim_{\mathbb{R}} \equiv_{\mathrm{W}} \lim_{\mathbb{N}^{\mathbb{N}}} \equiv_{\mathrm{W}} \lim_{[0,1]}$, and we just write lim for this Weihrauch degree. We also use the degree of $\lim_{\mathbb{N}}$, which is equivalent to closed choice on the natural numbers, $\mathrm{C}_{\mathbb{N}}$, to be discussed in Section 2.3.

There are several ways to construct new Weihrauch degrees from known ones, let us go through some that particularly important for our causes. First consider the parallelization, where the new task is to solve the original task a countable number of times in parallel. For a given multivalued function $f\colon \mathbf{X} \rightrightarrows \mathbf{Y}$ let $\hat{f}\colon \mathbf{X}^{\mathbb{N}} \rightrightarrows \mathbf{Y}^{\mathbb{N}}$ be the operation of applying f pointwise, i.e. $\hat{f}((x_n)_{n\in\mathbb{N}}) := \{(y_n)_{n\in\mathbb{N}} \mid \forall n \in \mathbb{N}\colon y_n \in f(x_n)\}$. One verifies that $f \leq_{\mathrm{W}} g$ implies $\hat{f} \leq_{\mathrm{W}} \hat{g}$, and hence the operation lifts to Weihrauch degrees.

▶ **Lemma 7** (Parallelization of Weihrauch degrees [6]). *The following equivalences hold:*

$$\widehat{LPO} \equiv_W \widehat{\mathrm{C}_{\mathbb{N}}} \equiv_W \widehat{\lim} \equiv_W \lim .$$

There are some more complicated constructions that we also need, namely the sequential composition and most prominently the diamond operator where a task can be used sequentially as many times as needed. However, for a proper discussion of this operator we need register machines and we thus postpone it to Section 3. An additional operations is the finite parallel execution f^* where the task is given any integer n to solve the task corresponding to f in parallel n-times.

2.3 Spaces of sets and choice principles

Closed choice on the natural numbers, denoted by $\mathrm{C}_{\mathbb{N}}$, is the task of selecting an element of a closed set of natural numbers. To make this formal let us first recall how to introduce spaces of open and closed subsets of represented spaces.

Recall from Example 2 that the functions from a topological space to Sierpiński space $\mathbb{S}$ are exactly the characteristic functions of the open sets. In analogy to this one may generalize from topological spaces to an arbitrary represented space $\mathbf{X}$ and consider $\mathcal{O}(\mathbf{X}) := \mathcal{C}(\mathbf{X}, \mathbb{S})$ the space of open subsets of $\mathbf{X}$. This means that for an arbitrary represented space $\mathbf{X}$ we consider a subset $A \subseteq \mathbf{X}$ to be open if its characteristic function $\chi_A \colon \mathbf{X} \to \mathbb{S}$ is continuously realizable. Similarly, the represented space $\mathcal{A}(\mathbf{X})$ of closed sets consists of the complements of open sets and names them accordingly.

▶ **Lemma 8** ([12]). *If $\mathbf{X}$ is a metric space in the Cauchy representation, then the information that a name of $A \in \mathcal{O}(\mathbf{X})$ provides is exactly a sequence of balls of rational radius around elements of the dense sequence that exhausts A.*

In particular, for a computable metric space the computable elements of $\mathcal{O}(\mathbf{X})$ are exactly those open sets that are computably enumerable in the usual sense. As an example and for later use recall that $\mathbb{R}_<$ denotes the lower reals from Example 3.

▶ **Example 9** (Lower approximations to the volume of open sets). The Lebesgue measure is computable as function $\lambda \colon \mathcal{O}([0,1]) \to \mathbb{R}_<$. This is because according to the previous lemma a name of an open set is a sequence of rational intervals that exhaust the set.

Closed choice for a space $\mathbf{X}$ is the multivalued function $\mathrm{C}_{\mathbf{X}} \colon \mathcal{A}(\mathbf{X}) \rightrightarrows \mathbf{X}$ where the eligible return values are the elements of the closed set provided as input. The domain of $\mathrm{C}_{\mathbf{X}}$ consists of the non-empty closed sets. Closed sets are encoded as positive information about their complement, thus $\mathrm{C}_{\mathbf{X}}$ is usually discontinuous and in particular incomputable. For instance $\mathrm{C}_{\mathbb{N}}$ can be reformulated as finding a number not occurring in an enumeration.

We also need the next higher level of complexity, namely the spaces of Σ^0_2 and Π^0_2 sets. The jump in complexity can be done using $\lim_{\mathbb{N}^{\mathbb{N}}}$ as a jump operator [15] respectively computable endofunctor [34, 32]. Namely, given a represented space $\mathbf{X} = (X, \delta_{\mathbf{X}})$ define the lim-jump of $\mathbf{X}$ to be the represented space that has the same underlying set but with the representation $\delta'_{\mathbf{X}} := \delta \circ \lim_{\mathbb{N}^{\mathbb{N}}} \circ \delta_{(\mathbb{N}^{\mathbb{N}})^{\mathbb{N}}}$. A function $f \colon \mathbf{X} \to \mathbf{Y}$ is called lim-computable resp. lim-continuous if it is computable resp. continuous as a function from $\mathbf{X}$ to the lim-jump of $\mathbf{Y}$. If $\mathbf{Y}$ is $\mathbb{R}$ or $\mathbb{R}^*$, this is the same as being (effectively) Baire class 1. The notion of lim-computability can also be expressed through Weihrauch degrees: A function $f \colon \mathbf{X} \to \mathbf{Y}$ is lim-computable if and only if $f \leq_{\mathrm{W}} \lim$.

We plan on replacing Sierpiński space $\mathbb{S}$ in the definitions of open and closed sets above by its lim-jump. Before we state these definitions let us specify a space $\mathbb{S}'$ that is isomorphic to the lim-jump of $\mathbb{S}$ but has a more approachable representation and convenient naming of its elements. Equip the set $\{\top', \bot'\}$ with the total representation $\delta_{\mathbb{S}'}$ that assigns $p \colon \mathbb{N} \to \mathbb{B}$ as name to $\top'$ if $p(n) = 0$ only for finitely many n.

Now that we have fixed $\mathbb{S}'$ we can set $\Sigma^0_2(\mathbf{X}) := \mathcal{C}(\mathbf{X}, \mathbb{S}')$ and let the names of an element of $\Pi^0_2(\mathbf{X})$ be the $\Sigma^0_2(\mathbf{X})$ names of its complement. As $\mathbb{S}'$ is isomorphic to the lim-jump of $\mathbb{S}$ this means that for an arbitrary represented space $\mathbf{X}$ we consider a subset $A \subseteq \mathbf{X}$ to be a Σ^0_2-set if its characteristic function $\chi_A \colon \mathbf{X} \to \mathbb{S}$ is lim-continuous. Here we follow the approach of synthetic descriptive set theory suggested in [34]. These definitions recreate more familiar ones that are only available in special cases. For instance: recall that the Π^0_2-subsets of a metric space are the countable intersections of open subsets. Following Brattka [5] one would define the names of a Π^0_2-set A to be the union of all name sets of sequences $(U_n)_{n\in\mathbb{N}} \in \mathcal{O}(\mathbf{X})^{\mathbb{N}}$ such that $A = \bigcap_{n\in\mathbb{N}} U_n$.

▶ **Lemma 10** (proven in [24]). *The representation of the Π^0_2-sets in the sense of Brattka as presented above can be computably translated to the representation of $\Pi^0_2(\mathbf{X})$. If $\mathbf{X}$ is a computable metric space, then a translation in the inverse direction is also possible.*

Using $\Pi^0_2(\mathbf{X})$ we can introduce another important Weihrauch degree:

▶ **Definition 11** (Π^0_2-choice). *Let $\Pi^0_2\mathrm{C}_{\mathbf{X}} \colon \Pi^0_2(\mathbf{X}) \rightrightarrows \mathbf{X}$ return the elements of A on input A.*

The domain of the multifunction $\Pi^0_2\mathrm{C}_{\mathbf{X}}$ are the non-empty Π^0_2-subsets of $\mathbf{X}$. Some other representatives of the Weihrauch degree $\Pi^0_2\mathrm{C}_{\mathbb{N}}$ can for instance be found in [7].

3 Algebraic models of computation and the diamond operator

Computation on algebraic structures has been studied in various ways, e.g. [26, 39]. The most influential approach is by Blum, Shub and Smale [2] introducing the algebraic model of computation on the reals that came to be named BSS-machines after the authors. We particularly wish to highlight Moschovakis' [29] as initial inspiration for our considerations. We use the notion of a register machine over some algebraic structure, as defined in [30] by following Gaßner [19, 20, 21] and Tavana and Weihrauch [38].

For us, an algebraic structure is a tuple $\mathfrak{A} = (A, f_1, f_2, \ldots, T_1, T_2, \ldots)$, where A is a set, each f_i is a (partial) function of type $f_i \colon \subseteq A^{k_i} \to A$, and each T_i is a relation of type $T_i \subseteq A^{l_i}$. A $\mathfrak{A}$-register machine computes functions of type $g \colon \subseteq A^* \to A^*$. It has registers $(Z_i)_{i\in\mathbb{N}}$ holding elements of A, and index registers $(I_n)_{n\in\mathbb{N}}$ holding natural numbers. Programs are finite lists of commands, consisting of:

- standard register machine operations on the index registers
- copying the value of the register Z_{I_1} indexed by I_1 into Z_{I_0}
- applying some f_i to the values contained in $Z_1, \ldots, Z_{k_i}$ and writing the result into Z_0
- jumping to a line in the program if the content of $Z_0, \ldots, Z_{l_i-1}$ is an element of T_i
- HALT, in which case the values in the registers $Z_0, \ldots, Z_{I_0}$ constitute the return value

At the start of the computation the register I_0 contains the length of the input, all other I_i start at 0. The input is in $Z_1, \ldots, Z_{I_0}$, all other Z_i contain some fixed value $a_0 \in A$. If the program either fails to halt on some input – which in particular happens if it invokes a partial function on some values outside its domain – the computed function is undefined on these values.

▶ **Definition 12.** *A BSS-machine is a $(\mathbb{R}; 0, 1, (q)_{q\in\mathbb{Q}}, +, \times, <)$-register machine.*

We can add any constant function $c \colon \mathbb{R}^0 \to \mathbb{R}$ taking a computable value without it making a difference anywhere in our paper, although this obviously increases the computational power whenever we add irrational constants. For a more detailed introduction, see [22].

As long as our signatures are finite, with the potential exception of constants, we can code $\mathfrak{A}$-register machine programs as inputs for an $\mathfrak{A}$-register machine using the length of the input for the discrete part, and elements of A for the constants. This in particular ensures the existence of universal machines, as these codes can be effectively decoded.

In computable analysis register machines whose signature only contains computable operations are a popular tool for making proofs more accessible [38]. Weihrauch reducibility has lead to a widespread use of register machines with incomputable signature:

▶ **Example 13** (The diamond operator). The Weihrauch degree $f^\diamond$ captures all tasks that we can solve in a finite computation with oracle access to f. A representative of $f^\diamond$ can then be given as a multifunction that takes as input an index of a register machine M together with an input x for M, and outputs whatever M would output on x. An equivalent definition in terms of reduction games was given by Jockusch and Hirschfeldt [27].

A proof that $\mathrm{C}_\mathbb{N} \equiv_\mathrm{W} \mathrm{LPO}^\diamond$ was given in [30]. Recall that LPO is the Weihrauch degree of deciding equality of real numbers. As BSS-machines are register machines with the capability to do branching over equalities of reals this equivalence provides a close link between closed choice on the natural numbers to the power of BSS-machines.

The composition of Weihrauch degrees is an operation related to the diamond operator that we need only in the passing. The Weihrauch degree $f \star g$ essentially means "do something computable, then invoke g, do some more computation, invoke f and after some more computation return an answer". We have $f \star g \equiv_\mathrm{W} \max_{\leq_\mathrm{W}}\{F \circ G \mid F \leq_\mathrm{W} f \wedge G \leq_\mathrm{W} g\}$, where the maximum is only taken over f and g that are composable. It is not obvious that the maximum exists, but a concrete construction of a representative can be found in [11]. The diamond operation corresponds to the closure under composition in the sense that $\mathrm{id}_{\mathbb{N}^\mathbb{N}} \leq_\mathrm{W} f = f \star f$ is equivalent to $f = f^\diamond$ as proven by Westrick [41].

3.1 Enhancing BSS-machines and statement of our results

In this section we consider functions $f\colon \mathbb{R}^* \to \mathbb{R}^*$, where $\mathbb{R}^*$ is the space of finite sequences of real numbers as introduced at the very end of Section 2.1. We compare several models of computation by register machines that may produce incomputable functions. The models we consider fall into two classes: One adapts BSS-machines with primitive operations for computing measures and the other adds capabilities of computing limits.

Starting from a BSS-machine, we add the ability to compute the Lebesgue measure of a given BSS-decidable subset of $[0,1]^d$ to obtain BSS+λ-machines. More specifically the run of a BSS-machine proceed as follows in evaluating the Lebesgue measure (for more details see the appendix):

▶ **Definition 14.** *If a BSS+λ-machine reaches a λ-command, the content of I_0 is interpreted as a Gödel-number of a BSS-machine M using k constants. The contents $a_1, \dots, a_k$ of $Z_1, \dots, Z_k$ are used as the values for the k constants and the content of I_1 as the dimension n of the input. If $M(a_1, \dots, a_k)$ halts for every input $x \in [0,1]^n$ and outputs either 0 or 1, then we replace the content of Z_0 with the Lebesgue measure of the set $\{x \in [0,1]^n \mid M(a_1, \dots, a_k)(x) = 1\}$ and let the computation continue. If M is not as desired we do not modify the state so that the computation loops on this command.*

As it is also meaningful to talk about indices of BSS+λ-machines, we may iterate this process once. That is, we use BSS+λ+λ-machines that have an additional primitive operator for computing the Lebesgue measure of a set decidable by a BSS+λ-machine. One may produce a more formal definition by replacement of the type of index used in the previous definition.

Motivated by our proofs we also consider machines with access to arbitrary (partial) computable functions. We call such machines BSS+Comp-machines and as most of the signature of BSS-machines is computable these machines can just use tests for strict inequality of real numbers in addition to the computable functions on the reals. We add to these machines the capability of computing measure very similar to how this was done for BSS- and BSS+λ-machines. There is a small issue to address here as the infinite signature of BSS+Comp-machines makes talking about programs slightly more complicated. However, we may replace the infinite signature with a finite one by use of a universal computable function $u\colon\subseteq\mathbb{R}\to\mathbb{R}$. This accounts for all unary computable functions and by the effective Kolmogorov superposition theorem [4], together with addition this already suffices to construct all computable functions of arbitrary finite arity. By adding a primitive for the measure of a set decidable in this setting (which is just a Δ^0_2-set), we obtain the BSS+Comp+λ-machines.

Our second class of models adds abilities to compute certain limits. The operator c-lim (for controlled limit) maps a program for a BSS-machine that computes a sequence of real numbers x_i with $|x_i - x_j| < 2^{-\min\{i,j\}}$ to the limit of this sequence. The operator u-lim (uncontrolled limit) accepts a program computing an arbitrary converging sequence of real numbers, and also outputs the limit. These operators correspond to the strongly and weakly analytic machines going back to Chadzelek and Hotz [14]. However, in our model the c-lim- and u-lim-commands can be used multiple times throughout the computation. Analytic machines only allow one application at the end of the computation. Thus, we consider the closure of what strongly/weakly analytic machines compute under composition here. We denote the respective machine models BSS+c-lim and BSS+u-lim and the details of what the commands do are similar to what was laid out in Definition 14.

In [30] it was shown that BSS-machines and strongly analytic machines can be characterized by a complete Weihrauch degree: every function computable in that model is Weihrauch reducible to the complete degree and the complete degree has a representative computable in the model. For the models we consider here we obtain the stronger characterization that the computed functions are exactly those functions from a lower cone in the Weihrauch degrees that are of suitable type. All of the models mentioned above coalesce into just two equivalence classes:

▶ **Theorem 15.** *For a function $f\colon\subseteq\mathbb{R}^*\to\mathbb{R}^*$ the following are equivalent:*

1. $f \leq_W \Pi^0_2\mathrm{C}_\mathbb{N}$
2. *f is computable by a BSS+λ-machine.*
3. *f is computable by a BSS+c-lim-machine.*
4. *f is computable by a BSS+Comp-machine with oracle access to the BSS-Halting problem*[2]*.*
5. *There is a uniform sequence of Π^0_2-sets $(A_n)_{n\in\mathbb{N}}$ such that $\mathrm{dom}(f) = \bigcup_{n\in\mathbb{N}} A_n$ and each $f|_{A_n}$ is computable.*

Anticipating the notion of being piece-wise continuous as introduced in Section 4.3 the last item may be reformulated as f being a computable element of the Π^0_2-piece-wise continuous functions. Another equivalent model that we omit here is to be computable by a weakly non-deterministic Type-2 machine with advice space $\mathbb{N}$ as introduced by Ziegler [42].

▶ **Theorem 16.** *For a function $f\colon\subseteq\mathbb{R}^*\to\mathbb{R}^*$ the following are equivalent:*

1. $f \leq_W \lim^\diamond$
2. *f is computable by a BSS+λ+λ-machine.*
3. *f is computable by a BSS+u-lim-machine.*
4. *f is computable by a BSS+Comp+λ-machine.*
5. *There is a uniform cover $\bigcup_{n\in\mathbb{N}} A_n = \mathrm{dom}(f)$ where A_n is Π^0_n, and $f|_{A_n}$ is of effective Baire class n.*

From known properties of the Weihrauch degrees we can now for instance draw conclusions about how far computability is preserved point-wise by functions computed in these models:

[2] Note that it does not matter whether we use the Halting problem for additive machines or the full strength, or even just $\mathbb{Q}$ as an oracle.

▶ **Corollary 17.** *A BSS+λ-machine with computable constants on computable input either diverges or produces a computable value. A BSS+λ+λ-machine with computable constants can, on computable input, produce outputs in any finite level of the arithmetical hierarchy.*

Proof. To see the first claim, we observe that $\Pi^0_2\mathrm{C}_\mathbb{N}$ returns natural numbers. Thus $f \leq_\mathrm{W} \Pi^0_2\mathrm{C}_\mathbb{N}$ implies that f can only take computable values on computable inputs. For the second claim, observe that one the one hand if $p \in \mathbb{B}^\mathbb{N}$ is arithmetical, then the constant function $x \mapsto p\colon \mathbb{R}^* \to \mathbb{R}^*$ (where we identify Cantor space $\mathbb{B}^\mathbb{N}$ and the Cantor middle third set) is Weihrauch reducible to $\lim^\diamond$. Conversely, since every computation of $\lim^\diamond$-machine involves only finitely many limits, its output on a computable input is always arithmetical. ◀

3.2 Measure, controlled limits and the Weihrauch degree of sorting

First, we show how a controlled limit can be reduced to the measure of a decidable set. Note that strongly analytic machines can compute all computable functions $f\colon \mathbb{R}^* \to \mathbb{R}^*$, so the following in particular implies that BSS+λ-machines can do the same.

▶ **Lemma 18.** *From a strongly analytic machine with parameters $(a_1, \ldots, a_d)$ that computes a function $f\colon \mathbb{R}^d \to [0,1]$ we can compute a BSS-machine (also using $(a_1, \ldots, a_d)$) that decides some subset $A_{a_1,\ldots,a_d} \subseteq [0,1]$ such that $f(a_1, \ldots, a_d) = \lambda(A_{a_1,\ldots,a_d})$.*

Proof. We partition the unit interval into $\{0\}$ and $\left((2/3)^{i+1}, (2/3)^i\right]$ for $i \in \mathbb{N}$. The set $A_{a_1,\ldots,a_d}$ will be of the form $A_{a_1,\ldots,a_d} = \bigcup_{i \in B_{a_1,\ldots,a_d}} \left((2/3)^{i+1}, (2/3)^i\right]$ for some $B_{a_1,\ldots,a_d} \subseteq \mathbb{N}$. A BSS-machine can decide $A_{a_1,\ldots,a_d}$ if and only if it can decide $B_{a_1,\ldots,a_d}$.

A BSS-machine can simulate the strongly analytic machine for any finite amount of time. At some point, it can pick a true case out of $f(a_1, \ldots, a_d) < 2/3$ and $1/3 < f(a_1, \ldots, a_d)$, and decides $0 \notin B_{a_1,\ldots,a_d}$ and $c_0 = 0$ in the former, and $0 \in B_{a_1,\ldots,a_d}$ and $c_0 = 1/3$ in the latter case.

In the next step, $f(a_1, \ldots, a_d) - c_0 \in [0, 2/3]$, and the BSS-machine can simulate the strongly analytic machine until it determines a true case amongst $f(a_1, \ldots, a_d) - c_0 < (2/3)^2$ and $\frac{2}{3^2} < f(a_1, \ldots, a_d) - c_0$, and then sets $1 \notin B_{a_1,\ldots,a_d}$ and $c_1 = c_0$ in the former, and $1 \in B_{a_1,\ldots,a_d}$ and $c_1 = c_0 + 2/9$. The BSS-machine keeps iterating this process until it determines whether or not $i \in B_{a_1,\ldots,a_d}$ for the i it needs to know about to decide membership of the input in $A_{a_1,\ldots,a_d}$. It is straight-forward calculation that $f(a_1, \ldots, a_d) = \lambda(A_{a_1,\ldots,a_d})$. ◀

In the next step, we obtain a Weihrauch upper bound for computing measures of BSS-decidable sets. This involves the Weihrauch degree Sort of the task of sorting infinite binary sequences. Here, sorting means to return the infinite binary sequence 0^n1^ω if the input sequence has exactly n zeros and 0^ω if it has infinitely many zeros. It was shown in [30] that Sort^* characterizes the strength of strongly analytic machines.

▶ **Lemma 19.** *The task* ***given a BSS-machine*** *M* ***with constants*** *$(a_1, \ldots, a_n)$* ***that decides a set*** *$A \subseteq [0,1]^d$***, compute** *$\lambda(A)$ is Weihrauch-reducible to LPO $\star$ Sort^n.*

Proof. Using LPO $\star$ Sort^n we can decide whether or not the $(a_1, \ldots, a_n)$ are algebraically independent, and if they are dependent, compute their minimal polynomial (as shown in [30]). Any test M makes on input $(x_1, \ldots, x_d)$ can be seen as asking whether a polynomial $P(a_1, \ldots, a_n, x_1, \ldots, x_d)$ is negative, positive, or 0. Knowing the minimal polynomial of $(a_1, \ldots, a_n)$ (or that they are algebraically independent) lets us decide whether P is 0 independent of the values of the x_i. If yes, we can eliminate the test for P from M. If no, then the set of $(x_1, \ldots, x_d)$ causing P to be 0 is of measure 0, hence can be safely ignored for determining $\lambda(A)$.

Once we do this procedure for all tests in M, we obtain two open sets U_1, U_2 (each as union of the open sets linked to a computation path where tests are yielding positive or negative answers) such that $U_1 \subseteq A$, $U_2 \subseteq [0,1]^d \setminus A$ and $\lambda([0,1]^d \setminus (U_1 \cup U_2)) = 0$. As the measure of open sets is lower-semicomputable, this allows us to compute $\lambda(A)$. ◀

The two preceding lemmas, together with the classification of strongly analytic machines from [30] already tell us that when allowing arbitrary use of the principle in a finite computation, then computing measures of BSS-computable sets, limits of fast converging BSS-computable sequences or solving Sort all yields the computational strength of $\mathrm{Sort}^\diamond$.

▶ **Proposition 20.** $\mathrm{Sort}^\diamond \equiv_W \Pi^0_2\mathrm{C}_\mathbb{N}$.

Proof. We find that isInfinite $\leq_\mathrm{W}$ $\mathrm{Sort} \star \mathrm{Sort}$ $\leq_\mathrm{W}$ $\mathrm{Sort}^\diamond$, and by results from [7], also $\mathrm{isInfinite}^\diamond \equiv_\mathrm{W} \Pi^0_2\mathrm{C}_\mathbb{N} \equiv_\mathrm{W} \Pi^0_2\mathrm{C}_\mathbb{N}^\diamond$. This shows that $\Pi^0_2\mathrm{C}_\mathbb{N} \leq_\mathrm{W} \mathrm{Sort}^\diamond$. For the other direction, we just point out that $\mathrm{Sort} \leq_\mathrm{W} \Pi^0_2\mathrm{C}_\mathbb{N}$ is immediate. ◀

We have now gathered all auxiliary results we need for proving the first of our theorems.

Proof of Theorem 15. **1. ⇒ 3.** It was shown in [30] that a strongly analytic machine can compute a representative of the Weihrauch degree Sort. It follows that a BSS+c-lim-machine can simulate a $\mathrm{Sort}^\diamond$-computation on any valid types. Now, $\mathrm{Sort}^\diamond \equiv_\mathrm{W} \Pi^0_2\mathrm{C}_\mathbb{N}$ by Proposition 20, so that our claim follows.

3. ⇒ 2. By Lemma 18, we can replace each c-lim-command by a λ-command.

2. ⇒ 1. By Lemma 19 a $\mathrm{Sort}^\diamond$-machine can simulate a BSS+λ-machine. Proposition 20 tells us that $\mathrm{Sort}^\diamond \equiv_\mathrm{W} \Pi^0_2\mathrm{C}_\mathbb{N}$.

1. ⇔ 4. This follows from [30, Theorem 21] and $\Pi^0_2\mathrm{C}_\mathbb{N} \equiv_\mathrm{W} \mathrm{isInfinite}^\diamond$.

1. ⇔ 5. It was observed by Nobrega [17] that being Weihrauch reducible to $\Pi^0_2\mathrm{C}_\mathbb{N}$ correspondents to a Π^0_2-cover of the domain such that all restrictions of the function to a piece are computable. Essentially, the pieces just are the inputs that lead to some number $n \in \mathbb{N}$ being a valid output of $\Pi^0_2\mathrm{C}_\mathbb{N}$ on the instance it is queried on. There is a minor issue here regarding whether we have a cover of the set of names of inputs, or directly of the inputs. For $\mathbb{R}^*$ as domain this makes no difference, as explained in [33]. ◀

3.3 Iterating measure and computable functions as supplement

We can now ask what happens if we allow to nest the λ-operator once.

▶ **Proposition 21.** *A function $f\colon \mathbb{R}^* \to \mathbb{R}^*$ is computable by a BSS+λ+λ-machine if and only if $f \leq_W \lim^\diamond$.*

Proof. A representative of the degree of lim is $\mathrm{id}\colon [0,1]_< \to [0,1]$, the identity from the lower reals in the unit interval to the unit interval. From $b \in [0,1]_<$ we can compute $[0,b) \in \mathcal{O}([0,1])$, and the characteristic function of an open set is computable relative to LPO, hence in particular relative to $\Pi^0_2\mathrm{C}_\mathbb{N}$. This establishes together with Theorem 15 that a BSS+λ+λ-machine can compute everything Weihrauch-reducible to lim, and subsequently $\lim^\diamond$.

For the other direction, assume that the characteristic function χ_A of $A \subseteq [0,1]^d$ is computable by a BSS+λ-machine. By Theorem 15 it follows that $\chi_A \leq_\mathrm{W} \Pi^0_2\mathrm{C}_\mathbb{N}$. This in turn tells us that there are Π^0_2-sets $(B_n)_{n\in\mathbb{N}}$ and $(C_n)_{n\in\mathbb{N}}$ such that $A = \bigcup_{n\in\mathbb{N}} B_n$ and $[0,1]^d \setminus A = \bigcup_{n\in\mathbb{N}} C_n$. In particular, A is a Δ^0_3-set. Using $\lim \star \lim \star \lim$ we can compute the measure of a Σ^0_3-set as a real number, and thus the claim follows. ◀

Note that for the first direction in the theorem above we only used that BSS+λ-machines can compute all computable functions, and can decide all computable open sets. This can already be facilitated by BSS+Comp-machines that can be simulated by BSS+λ-machines:

▶ **Corollary 22.** *BSS+Comp+λ- and BSS+λ+λ-machines can compute the same functions.*

The following is somewhat more general than we need for our theorem.

▶ **Proposition 23.** *The following are equivalent for $f\colon \mathbf{X} \rightrightarrows \mathbf{Y}$, where $\mathbf{X}$ is a effectively countably based space:*

1. $f \leq_W \lim^\diamond$.
2. *There is a uniform cover $\bigcup_{n\in\mathbb{N}} A_n$ of $\mathbf{X}$ where A_n is Π^0_n, and $f|_{A_n}$ is of effective Baire class n.*

Proof. Let $f \leq_{\mathrm{W}} \lim^\diamond$. We point out that using lim, we can chose a canonic name for each point in an effectively countably-based space. Thus, in the $\lim^\diamond$-computation, we can assume that all names of the same point proceed in the same way through the computation tree generated by the generalized register machine in the definition of $^\diamond$. The computation tree has countably many leaves where the computation can terminate and provide an output. If the path to a particular leaf is using n invocations of lim, then the set of inputs leading to that leaf is a Π^0_{n+1}-set, and we can obtain this uniformly. Every content of a register at that moment (thus in particular the output) can be obtained as an effectively Baire class n function. By padding with empty sets if necessary, we obtain the desired sequence from an enumeration of the leaves.

Conversely, if we have a uniform cover $\bigcup_{n\in\mathbb{N}} A_n$ of $\mathbf{X}$ where A_n is Π^0_n, then a $\lim^\diamond$-computation can on input $x \in \mathbf{X}$ find some n such that $x \in A_n$. Subsequently, a $\lim^\diamond$-computation can simulate any effective Baire class n function. ◀

Note that in particular the pre-computable Quasi-Polish space from [16] are effectively countably-based. Finally, we prove our second theorem.

Proof of Theorem 16.

1. ⇔ 2., 2. ⇔ 4. and 1. ⇔ 5. These follow from the Propositions 21 and 23 and Corollary 22 respectively.

1. ⇔ 3. As a BSS+u-lim-machine can compute all computable partial functions on $\mathbb{R}^*$, the only difference between a BSS+u-lim and a $\lim^\diamond$-machine is what are appropriate input and output types. ◀

4 Measurability, Integrability and Weihrauch degrees

Classically, any measurable function $f\colon [0,1] \to [0,1]$ is integrable. Computably, this fails to be true. Rather than dealing with all Borel measurable functions, we restrict our attention to the lowest non-trivial complexity and explore the Δ^0_2-measurable functions. For metric spaces $\mathbf{X}$ and $\mathbf{Y}$, the Jayne-Rogers Theorem gives an alternate description of the Δ^0_2-measurable functions as piece-wise continuous functions [33]. In our setting the piece-wise continuous functions can be identified with the space $\mathcal{C}(\mathbf{X}, \mathbf{Y}^\nabla)$ of $\lim_\Delta$-continuous functions. Below, we go into more detail about these spaces and also about the space $\mathrm{L}_1([0,1])$ of integrable functions. Before we do this recall that $\mathbb{R}_<$ is the space of real numbers represented as suprema of sequences of rationals and let us state the main results that we prove.

▶ **Theorem 24.** *The following maps are Weihrauch equivalent to* lim*:*

1. *Evaluating a continuous function from its description as an integrable function. That is, the inverse of the inclusion of* $\mathcal{C}([0,1],[0,1])$ *into* $\mathrm{L}_1([0,1])$*.*
2. *The Lebesgue measure as function on* $\Delta^0_2([0,1])$ *with values either in* $\mathbb{R}$ *or in* $\mathbb{R}_<$*, i.e.*

$$\lambda\colon \Delta^0_2([0,1]) \to \mathbb{R} \quad \textit{and} \quad \lambda\colon \Delta^0_2([0,1]) \to \mathbb{R}_<.$$

3. *Integration taking a piece-wise continuous function on the unit interval and returning an element of either* $\mathbb{R}$ *or* $\mathbb{R}_<$*, namely the functions*

$$\int\colon \mathcal{C}([0,1],[0,1]^\nabla) \to \mathbb{R} \quad \textit{and} \quad \int\colon \mathcal{C}([0,1],[0,1]^\nabla) \to \mathbb{R}_<.$$

4. *Translating from piece-wise continuous functions to integrable functions, i.e. the inclusion of* $\mathcal{C}([0,1],[0,1]^\nabla)$ *into* $\mathrm{L}_1([0,1])$*.*

4.1 Integrable functions, $\mathrm{L}_1([0,1])$ and continuous functions

Let us first discuss what the represented space of integrable functions looks like. Recall that strictly speaking $\mathrm{L}_1([0,1])$ is not a space of functions but instead its elements are equivalence classes of such. For $f\colon [0,1] \to \mathbb{R}$ integrable in the sense that $\int_0^1 |f|\mathrm{d}\lambda < \infty$, the collection

$$[f] := \{g\colon [0,1] \to \mathbb{R} \mid \int_0^1 |f-g|\mathrm{d}\lambda = 0\}$$

of functions that coincide with f almost everywhere is an element of $\mathrm{L}_1([0,1])$. The vector-space operations factor through the equivalence class and $\|[f]\|_1 := \int_0^1 |f|\mathrm{d}\lambda$ defines a Norm on $\mathrm{L}_1([0,1])$. For this paper equip $\mathrm{L}_1([0,1])$ with the Cauchy representation with respect to the metric induced by this norm and choose as dense set the equivalence classes of piece-wise constant functions with a finite number of rational values and breakpoints.

For more on representing spaces of integrable functions, see [36] and [37]. Integration is computable as a mapping that takes endpoints of an interval and a function and integrates it over the interval.

Let us give a name to the restriction of the assignment $f \mapsto [f]$ to continuous functions:

$$\iota\colon \mathcal{C}([0,1],[0,1]) \to \mathrm{L}_1([0,1]), \quad f \mapsto [f].$$

As continuous functions that are equal almost everywhere are already equal, ι is an injection. However, a name of $\iota(f)$ provides strictly less information than one of f. Our first lemma follows from results by Brattka [3] but we provide an elementary direct proof in the appendix.

▶ **Lemma 25** ($\lim \leq_{\mathrm{W}} \iota^{-1}$)**.** *Evaluating an integrable function that happens to be continuous is enough to compute the limit of a sequence in Baire space.*

▶ **Lemma 26** ($\iota^{-1} \leq_{\mathrm{W}} \lim$)**.** *The inverse of the inclusion of* $\mathcal{C}([0,1],[0,1])$ *into* $\mathrm{L}_1([0,1])$ *is Weihrauch reducible to* lim*.*

Proof.[3] Recall that $\lim \equiv_{\mathrm{W}} \lim^{\mathbb{N}}_{[0,1]}$, so it suffices to produce a Weihrauch reduction of ι^{-1} to $\lim^{\mathbb{N}}_{[0,1]}$. Let us assume we are given $\iota(f) \in \mathrm{L}_1([0,1])$. Fix some enumeration $(I_k)_{k\in\mathbb{N}}$ of the rational intervals with endpoints in $[0,1]$. Assume we are given $\varepsilon \in \mathbb{Q}^+$ and we want to

[3] We are grateful to a referee for sketching this simplified proof for us.

produce some rational polynomial p such that $\|f-p\|_\infty \leq \varepsilon$. Let $(p_m)_{m\in\mathbb{N}}$ be an enumeration of the rational polynomials. Note that one easily obtains $[p_m]$ as an integrable function and as we can compute integrals of integrable functions, for each m the sequence $(x_n)_{n\in\mathbb{N}}$

$$x_n := \frac{1}{\lambda(I_n)}|\int_{I_n} f - p_m \mathrm{d}\lambda|$$

is computable. Note that $x_n \leq \|f-p_m\|_\infty$ and that we can get x_n arbitrary close to $\|f-p_m\|_\infty$: As f is continuous there exists an interval where f is almost constantly almost the value whose absolute value is maximal. Thus, one instance of $\lim_{[0,1]}$ can compute $\|f-p_m\|_\infty = \sup\{x_n \mid n\in\mathbb{N}\} = \lim_{[0,1]}\big((\max\{x_k \mid k\leq n\})_{n\in\mathbb{N}}\big)$. Moreover, using $\lim_{[0,1]}^{\mathbb{N}}$ produces $(\|f-p_m\|_\infty)_{m\in\mathbb{N}}$ as a sequence. Since f is continuous, we know that there exists some p_m such that $\|f-p_m\|_\infty < \varepsilon$. As a test for strict inequality is computable when its values are taken to be from $\mathbb{S}$, we may find such p_m by dovetailing and return it. ◀

4.2 The Lebesgue measure on the Δ_2^0-subsets of the unit interval

Let us now introduce the space $\Delta_2^0(\mathbf{X})$ of Δ_2^0-subsets of a represented space $\mathbf{X}$. First recall that $\Delta_1^0(\mathbf{X}) := \mathcal{C}(\mathbf{X},\mathbb{B})$ is the space of clopen subsets and can alternatively be represented a pair of names of the characteristic function of both a set and its complement as continuous functions to Sierpiński space. The space $\Delta_2^0(\mathbf{X})$ can be introduced the same way but with Sierpiński space replaced by its lim-jump as introduced in Section 2.3. As the product of lim-jumps of two spaces is isomorphic to the lim-jump of the product, from $\Delta_1^0(\mathbf{X}) = \mathcal{C}(\mathbf{X},\mathbb{B})$ we get $\Delta_2^0(\mathbf{X}) \simeq \mathcal{C}(\mathbf{X},\mathbb{B}')$, where $\mathbb{B}'$ is the lim-jump of $\mathbb{B}$. As every convergent sequence in $\mathbb{B}$ is already eventually constant, $\mathbb{B}'$ allows for a simpler description using another jump operator. Let $\lim_\Delta(p_n) := \{p\in\mathbb{N}^\mathbb{N} \mid \exists N, \forall n\geq N, p_n = p\}$ be the limit operator with respect to the discrete topology on Baire space.

▶ **Definition 27** (The $\lim_\Delta$-jump $\mathbf{X}^\nabla$ of a space $\mathbf{X}$). *Define the* $\lim_\Delta$*-jump of a represented space* $\mathbf{X} = (X,\delta_\mathbf{X})$ *as* $\mathbf{X}^\nabla := (X, \delta_\mathbf{X}\circ\lim_\Delta \circ\delta_{(\mathbb{N}^\mathbb{N})^\mathbb{N}})$.

Note that the definition of the lim-jump of in Section 2.3 is identical, except that $\lim_\Delta$ replaces $\lim_{\mathbb{N}^\mathbb{N}}$. Now, $\mathbb{B}^\nabla$ is isomorphic to the lim-jump of $\mathbb{B}$ and $\Delta_2^0(\mathbf{X}) \simeq \mathcal{C}(\mathbf{X},\mathbb{B}^\nabla)$.

▶ **Lemma 28.** *There exists a computable function $D\colon \mathrm{dom}(\lim_{[0,1]}) \to \Delta_2^0([0,1])$ that maps each converging $\mathbf{x}\in[0,1]^\mathbb{N}$ to a set $D(\mathbf{x})$ with Lebesgue measure $\lambda(D(\mathbf{x})) = \lim_{[0,1]}(\mathbf{x})$.*

Proof. Our construction here has some similarities to the proof of Lemma 18. To argue that we can compute a Δ_2^0-set A from the sequence $(a_i)_{i\in\mathbb{N}}$ means that we can compute its characteristic function as $\chi_A\colon [0,1]\to\mathbb{B}_\Delta$ with access to $(a_i)_{i\in\mathbb{N}}$, which in turn is equivalent to a LPO$^\diamond$-machine being able to compute $\chi_A\colon[0,1]\to\mathbb{B}$. We give the algorithm for the latter as Algorithm 1 in the appendix. ◀

From this Lemma we can draw the conclusion that we need.

▶ **Corollary 29** ($\big(\lambda\colon \Delta_2^0([0,1])\to\mathbb{R}\big) \equiv_\mathrm{W} \lim$). *The Lebesgue measure as a function from $\Delta_2^0([0,1])$ to $\mathbb{R}$ is a representative of the Weihrauch degree* lim.

Proof. It suffices to prove $\lambda \leq_\mathrm{W} \lim_{\mathbb{N}^\mathbb{N}}$ and $\lim_{[0,1]} \leq_\mathrm{W} \lambda$. The latter of these follows directly from the previous Lemma 28.

To see that also $\lambda\leq_\mathrm{W}\lim_{\mathbb{N}^\mathbb{N}}$ assume that we are given $A\in\Delta_2^0([0,1])$ as input. Recall that a Δ_2^0-set is specified as a pair names of itself and its complement as Π_2^0-subsets of $[0,1]$. Now $[0,1]$ is a computable metric space and according to Lemma 10 we can thus computably obtain names of sequences $(U_n)_{n\in\mathbb{N}}, (V_n)_{n\in\mathbb{N}} \in \mathcal{O}([0,1])^{\mathbb{N}}$ so that $\bigcap_{n\in\mathbb{N}} U_n = A$ and $\bigcap_{n\in\mathbb{N}} V_n = A^c$. Thus the sequences $x_n := \lambda(\bigcap_{k\leq n} U_k)$ converges to $\lambda(A)$ from above and the sequence $y_n := 1 - \lambda(\bigcap_{k\leq n} V_k)$ from below. Note that the Lebesgue measure is computable from open sets to $\mathbb{R}_<$. However, it is known that $(\mathrm{id}\colon \mathbb{R}_< \to \mathbb{R}) \equiv_{\mathrm{W}} \lim$ and we know that $\widehat{\lim} \equiv_{\mathrm{W}} \lim$ from Lemma 7. Therefore, $\lim_{\mathbb{N}^{\mathbb{N}}}$ is sufficient to lift both of the sequences (x_n) and (y_n) from sequences in $\mathbb{R}_<$ to sequences of real numbers. These sequences approximate $\lambda(A)$ from above and below so that we can compute $\lambda(A) \in \mathbb{R}$. ◀

4.3 Piece-wise continuous functions and Δ^0_2-measurable functions

Before we talk about Δ^0_2-measurable functions let us get back to admissibility and continuous functions for illustration. For any $f \in \mathcal{C}(\mathbf{X},\mathbf{Y})$ we may consider the pre-image function $f^{-1}\colon \mathcal{O}(\mathbf{Y}) \to \mathcal{O}(\mathbf{X})$ defined by $f^{-1}(\chi) := \chi \circ f$. The above says that continuously realizable functions between represented spaces are such that the preimage of an open set is not only open but can continuously be obtained from it. Now, admissibility of the representation of $\mathbf{Y}$ guarantees that the notions coincide in that it assures that the assignment $f \mapsto f^{-1}$ can be inverted and in particular $\mathcal{C}(\mathbf{X},\mathbf{Y})$ is isomorphic to its image in $\mathcal{C}(\mathcal{O}(\mathbf{Y}),\mathcal{O}(\mathbf{X}))$ under moving to the pre-image function. A function is called Δ^0_2-measurable if its preimages of open sets are Δ^0_2-sets. We may thus ask whether there is an assumption that allows reconstruction of a Δ^0_2-measurable function from its preimage function $\mathcal{O}(\mathbf{Y}) \to \Delta^0_2(\mathbf{X})$ just like admissibility did this for continuity. Indeed, there exists such a condition and it is called $\lim_\Delta$-admissibility. It implies that $\mathcal{C}(\mathbf{X},\mathbf{Y}^\nabla)$ is isomorphic to the corresponding subspace of $\mathcal{C}(\mathcal{O}(\mathbf{Y}),\Delta^0_2(\mathbf{X}))$. Here $\mathbf{Y}^\nabla$ is the $\lim_\Delta$-jump of $\mathbf{Y}$ from Definition 27 and as its underlying set is identical to that of $\mathbf{Y}$, measurable functions are indeed functions from $\mathbf{X}$ to $\mathbf{Y}$. We point to [33, 32] for proofs that all spaces that appear as $\mathbf{Y}$ in this paper are $\lim_\Delta$-admissible.

The space $\mathcal{C}(\mathbf{X},\mathbf{Y}^\nabla)$ can often be understood as the space of piece-wise continuous functions via the Jayne-Rogers theorem. A function $f\colon \mathbf{X} \to \mathbf{Y}$ is piece-wise continuous if there is a sequence $(A_n)_{n\in\mathbb{N}} \in \mathcal{A}(\mathbf{X})^{\mathbb{N}}$ that covers $\mathbf{X}$ and such that for each $n \in \mathbb{N}$ the function $f|_{A_n}$ is continuous from A_n as a subspace of $\mathbf{X}$ to $\mathbf{Y}$. Let $\mathcal{C}_{pw}(\mathbf{X},\mathbf{Y})$ be the space of piece-wise continuous functions represented as expected.

▶ **Lemma 30** ($\mathcal{C}(\mathbf{X},\mathbf{Y}^\nabla) \simeq \mathcal{C}_{pw}(\mathbf{X},\mathbf{Y})$, proven in [33]). *Let $\mathbf{X}$ and $\mathbf{Y}$ be a computable metric spaces such that $\mathbf{Y}$ is complete. Then $\mathcal{C}(\mathbf{X},\mathbf{Y}^\nabla)$ is isomorphic to the space of piece-wise continuous functions $\mathcal{C}_{pw}(\mathbf{X},\mathbf{Y})$ as introduced above.*

Let us give a name to another restriction of the assignment $f \mapsto [f]$, namely set

$$\iota_\Delta\colon \mathcal{C}([0,1],[0,1]^\nabla) \to \mathrm{L}_1([0,1]), \quad f \mapsto [f].$$

This map is not injective: as piece-wise continuous functions characteristic functions of singletons are distinct from zero but they produce the same equivalence class in $\mathrm{L}_1([0,1])$.

▶ **Lemma 31** ($\left(\int\colon \mathcal{C}([0,1],[0,1]^\nabla) \to \mathbb{R}\right) \leq_{\mathrm{W}} \lim$). *Integrating a piece-wise continuous function over the unit interval is Weihrauch reducible to $\lim_{\mathbb{R}}$.*

Proof. According to Lemma 30, the information that we get about f is a sequence A_n of closed sets together with the restrictions $f|_{A_n}$. By Lemma 8, the information contained in a name of $A_n \in \mathcal{A}([0,1])$ is an increasing sequence $(U^i_n)_{i\in\mathbb{N}}$ of finite unions of rational intervals such that $\bigcup_{i\in\mathbb{N}} U^i_n = A^c_n$. Construct a sequence $r_n \in [0,1]$ as follows: For each $i \leq 2^n$ set $x_i := i \cdot 2^{-n}$ and pick some canonical name of $x_i \in [0,1]$. For each $k \leq n$ check whether $x_i \in U^k_n$, and if this is not the case evaluate the universal used for function spaces for n steps in attempt to obtain a 2^{-n} approximation to $f|_{A_k}(x_i)$. Let $g(x_i)$ be the first value for which this succeeds and returns something in $[-2^{-n}, 1+2^{-n}]$ and let $g(x_i)$ be zero if this never happens. Set $r_n := \sum_{i\leq 2^n} g(x_i)\cdot 2^{-n}$ and note that our description of this sequence provides a way to compute it from the input information. To see that the sequence r_n converges to the integral of f one uses the sigma-additivity of the Lebesgue measure and that, as the A_n cover $[0,1]$, it holds that $\lim_{\mathbb{R}}(\lambda(A_n))_{n\in\mathbb{N}} = 1$ and also $\lim_{\mathbb{R}}(\lambda(U^i_n))_{i\in\mathbb{N}} = 1-\lambda(A_n)$. ◀

▶ **Corollary 32** ($\iota_\Delta \equiv_{\mathrm{W}} \lim$). *The Weihrauch degree of the mapping from $\mathcal{C}([0,1],[0,1]^\nabla)$ to $\mathrm{L}_1([0,1])$ that takes f to its equivalence class is* $\lim$.

Proof. Let us first argue that $\iota_\Delta \leq_{\mathrm{W}} \lim$ and thus assume that we are given an input for ι_Δ, i.e. some $f \in \mathcal{C}([0,1],[0,1]^\nabla)$. Let $(p_k)_{k\in\mathbb{N}}$ be an enumeration of rational piece-wise constant functions similar to the one we use for defining the Cauchy representation of $\mathrm{L}_1([0,1])$ but who map to $[0,1]$. Note that the sequence $(p_k)_{k\in\mathbb{N}}$ is computable as sequence in $\mathcal{C}([0,1],[0,1]^\nabla)$ and that taking differences and the absolute value are computable operations on the piece-wise continuous functions as they can be lifted in a pointwise manner from the continuous functions. Thus, for each fixed k we can compute $\|[f]-[p_k]\|_1 = \int_0^1 |f-p_k|\mathrm{d}\lambda$ using an invocation of $\lim$ by the previous Lemma. As $\widehat{\lim} \equiv_{\mathrm{W}} \lim$ this means that we can also compute $(\|[f]-[p_k]\|_1)_{k\in\mathbb{N}}$ as a sequence by one application of $\lim$. Now given some $\varepsilon \in \mathbb{Q}^+$ we know that there exists some p_k such that $\|[f]-[p_k]\|_1 < \varepsilon$ and may thus search for it and return it.

To see that also $\lim \leq_{\mathrm{W}} \iota_\Delta$ note that $\mathbb{B}^\nabla$ is isomorphic to $\{0,1\}$ as subspace of $[0,1]^\nabla$. Thus we can computably obtain the characteristic function $\chi_A \in \mathcal{C}([0,1],[0,1]^\nabla)$ from a $A \in \Delta^0_2([0,1])$. As $\int_0^1 \chi_A \mathrm{d}\lambda = \lambda(A)$ we can thus use ι_Δ to compute the Lebesgue measure as function from $\Delta^0_2([0,1])$ to $\mathbb{R}$ and get the desired reduction from Lemma 29. ◀

References

1 Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation.* Springer, 1998.

2 Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, 1989. URL: `http://projecteuclid.org/euclid.bams/1183555121`.

3 Vasco Brattka. Computable invariance. *Theoretical Computer Science*, 210:3–20, 1999. `doi:10.1016/S0304-3975(98)00095-4`.

4 Vasco Brattka. A computable Kolmogorov superposition theorem. Informatik Berichte 272, FernUniversität Hagen, 2000.

5 Vasco Brattka. Effective Borel measurability and reducibility of functions. *Mathematical Logic Quarterly*, 51(1):19–44, 2005. `doi:10.1002/malq.200310125`.

6 Vasco Brattka, Matthew de Brecht, and Arno Pauly. Closed choice and a uniform low basis theorem. *Annals of Pure and Applied Logic*, 163(8):968–1008, 2012. `doi:10.1016/j.apal.2011.12.020`.

7 Vasco Brattka, Guido Gherardi, Rupert Hölzl, Hugo Nobrega, and Arno Pauly. Borel choice. in preparation.

8 Vasco Brattka, Guido Gherardi, and Arno Pauly. Weihrauch complexity in computable analysis, 2017. URL: `https://arxiv.org/abs/1707.03202`.

9 Vasco Brattka and Peter Hertling. Feasible real random acess machines. *Journal of Complexity*, 14:490–526, 1998. `doi:10.1006/jcom.1998.0488`.

10 Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A tutorial on computable analysis. In Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *New Computational Paradigms:*

Changing Conceptions of What is Computable, pages 425–491. Springer, 2008. URL: `https://link.springer.com/chapter/10.1007/978-0-387-68546-5_18`.

11 Vasco Brattka and Arno Pauly. On the algebraic structure of Weihrauch degrees. *Logical Methods in Computer Science*, 14(4):1–36, 2018. doi:10.23638/LMCS-14(4:4)2018.

12 Vasco Brattka and Gero Presser. Computability on subsets of metric spaces. *Theoretical Computer Science*, 305(1-3):43–76, 2003. doi:10.1016/S0304-3975(02)00693-X.

13 Peter Bürgisser and Felipe Cucker. Counting complexity classes for numeric computations ii: Algebraic and semialgebraic sets. *Journal of Complexity*, 22(2):147–191, 2006. doi:10.1016/j.jco.2005.11.001.

14 Thomas Chadzelek and Günter Hotz. Analytic machines. *Theoretical Computer Science*, 219:151–167, 1999. doi:10.1016/S0304-3975(98)00287-4.

15 Matthew de Brecht. Levels of discontinuity, limit-computability, and jump operators. In Vasco Brattka, Hannes Diener, and Dieter Spreen, editors, *Logic, Computation, Hierarchies*, pages 79–108. de Gruyter, 2014. doi:10.1515/9781614518044.79.

16 Matthew de Brecht, Arno Pauly, and Matthias Schröder. Overt choice. *Computability*, 2020. available at https://arxiv.org/abs/1902.05926. doi:10.3233/COM-190253.

17 Hugo de Holanda Cunha Nobrega. Game characterizations of function classes and Weihrauch degrees. M.Sc. thesis, University of Amsterdam, 2013. URL: `https://eprints.illc.uva.nl/905/1/MoL-2013-16.text.pdf`.

18 Tobias Gärtner and Martin Ziegler. Real analytic machines and degrees. *Logical Methods in Computer Science*, 7:1–20, 2011. doi:10.2168/LMCS-7(3:11)2011.

19 Christine Gaßner. On NP-completeness for linear machines. *Journal of Complexity*, 13:259–271, 1997. doi:10.1006/jcom.1997.0444.

20 Christine Gaßner. The P-DNP problem for infinite abelian groups. *Journal of Complexity*, 17:574–583, 2001. doi:10.1006/jcom.2001.0583.

21 Christine Gaßner. A hierarchy below the halting problem for additive machines. *Theory Computing Systems*, 17:574–583, 2008. doi:10.1007/s00224-007-9020-y.

22 Christine Gaßner. An introduction to a model of abstract computation: the BSS-RAM model. In Adrian Rezus, editor, *Contemporary Logic and Computing*, volume 1 of *Landscapes in Logic*, pages 574–603. College Publications, 2020.

23 Christine Gaßner and Pedro F. Valencia Vizcaíno. Operators for BSS RAM's. In Martin Ziegler and Akitoshi Kawamura, editors, *The Twelfth International Conference on Computability and Complexity in Analysis*, pages 24–26, 2015.

24 Vassilios Gregoriades, Tamás Kispéter, and Arno Pauly. A comparison of concepts from computable analysis and effective descriptive set theory. *Mathematical Structures in Computer Science*, 27(8):1414–1436, 2017. 2015. doi:10.1017/S0960129516000128.

25 Anders C. Hansen. On the solvability complexity index, the n-pseudospectrum and approximations of spectra of operators. *Journal of the AMS*, 24:81–124, 2011.

26 Armin Hemmerling. Computability of string functions over algebraic structures. *Mathematical Logic Quarterly*, 44(1):1–44, 1998. doi:10.1002/malq.19980440102.

27 Denis R. Hirschfeldt and Carl G. Jockusch. On notions of computability-theoretic reduction between Π^1_2-principles. *Journal of Mathematical Logic*, 16(1), 2016. 1650002:1-1650002:59. doi:10.1142/S0219061316500021.

28 Klaus Meer. Counting problems over the reals. *Theoretical Computer Science*, 242:41–58, 2000. doi:10.1016/S0304-3975(98)00190-X.

29 Yiannis N. Moschovakis. Abstract first order computability. I. *Transactions of the American Mathematical Society*, 138:427–464, 1969. doi:10.2307/1994926.

30 Eike Neumann and Arno Pauly. A topological view on algebraic computations models. *Journal of Complexity*, 44:1–22, 2018. doi:10.1016/j.jco.2017.08.003.

31 Arno Pauly. On the topological aspects of the theory of represented spaces. *Computability*, 5(2):159–180, 2016. doi:10.3233/COM-150049.

32 Arno Pauly and Matthew de Brecht. Towards synthetic descriptive set theory: An instantiation with represented spaces. http://arxiv.org/abs/1307.1850, 2013.

33 Arno Pauly and Matthew de Brecht. Non-deterministic computation and the Jayne Rogers theorem. *Electronic Proceedings in Theoretical Computer Science*, 143:87–96, 2014. DCM 2012. doi:10.4204/EPTCS.143.8.

34 Arno Pauly and Matthew de Brecht. Descriptive set theory in the category of represented spaces. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 438–449, 2015. doi:10.1109/LICS.2015.48.

35 Arno Pauly and Florian Steinberg. Comparing representations for function spaces in computable analysis. *Theory of Computing Systems*, 62(3):557–582, 2018. doi:10.1007/s00224-016-9745-6.

36 Marian Pour-El and Ian Richards. *Computability in analysis and physics.* Perspectives in Mathematical Logic. Springer, 1989.

37 Florian Steinberg. Complexity theory for spaces of integrable functions. *Logical Methods in Computer Science*, 13(3):1–39, 2017. doi:10.23638/LMCS-13(3:21)2017.

38 Nazanin Tavana and Klaus Weihrauch. Turing machines on represented sets, a model of computation for analysis. *Logical Methods in Computer Science*, 7:1–21, 2011. doi:10.2168/LMCS-7(2:19)2011.

39 John V. Tucker and Jeffrey I. Zucker. Computable functions and semicomputable sets on many-sorted algebras. In T.S.E. Maybaum S. Abramsky, D.M. Gabbay, editor, *Handbook of Logic in Computer Science*, volume 5 of *Oxford Science Publications*, pages 317–523, 2000.

40 Klaus Weihrauch. *Computable Analysis.* Springer-Verlag, 2000.

41 Linda Westrick. A note on the diamond operator. *Computability*, 202X. to appear. doi:10.3233/COM-200295.

42 Martin Ziegler. Computability and continuity on the real arithmetic hierarchy and the power of type-2 nondeterminism. In Barry S. Cooper, Benedikt Löwe, and Leen Torenvliet, editors, *Proceedings of CiE 2005*, volume 3526 of *Lecture Notes in Computer Science*, pages 562–571. Springer, 2005. URL: https://link.springer.com/chapter/10.1007/11494645_68.

A Proof of Lemma 28

That we have access to LPO is relevant in Lines 5 and 7, where LPO lets us resolve the if-statements. As we can use arbitrary computable functions, the universal quantifiers $\forall i \geq n$ are unproblematic here. Since the sequence $(a_i)_{i \in \mathbb{N}}$ is guaranteed to converge, the while-loop will terminate. If $b = 0$, then for sure $\lim_{i \to \infty} a_i \in [0, {}^2/_3]$, if $b = 1$ then $\lim_{i \to \infty} a_i \in [{}^1/_3, 1]$. In the latter case, Line 15 adds ${}^1/_3$ to the measure of A. By moving to the sequence $(a_i - {}^b/_3)_{i \in \mathbb{N}}$ we then get a sequence guaranteed to have a limit in $[0, {}^2/_3]$, and still have to determine membership in A for $x \in ({}^1/_3, 1)$. We rescale this interval up to $(0, 1)$ again, and iterate the process.

B Proof of Lemma 25

Proof. As $\lim \equiv_{\mathrm{W}} \widehat{\mathrm{LPO}}$ by Lemma 7 we may solve a countable number of instances of $\mathrm{LPO} = \chi_{\{k \mapsto 0\}}$ instead of one of lim. Thus, our input is a sequence (p_n) of elements of Baire space. Let m_n be the least natural number such that $p_n(m_n) \neq 0$ if such a number exists and ∞ otherwise. Define a sequence of functions $f_n \in \mathcal{C}([0, 1])$ by

$$f_n(x) := \begin{cases} 0 & \text{if } m_n = \infty \\ \max\{0, 1 - 2^{m_n+2}|x - \frac{3}{4}|\} & \text{otherwise,} \end{cases}$$

Algorithm 1 Computing the characteristic function of A.

```
Function Charac is
    input  : $x \in [0,1]$, converging sequence $(a_i)_{i\in\mathbb{N}}$
    output : Boolean indicating whether $x \in A$
    if $x = 0$ then return "no";
    $b := -1$; $n = 0$; while $b = -1$ do
        if $\forall i \geq n\ a_i \leq \frac{2}{3}$ then
            $b := 0$;
        else if $\forall i \geq n\ a_i \geq \frac{1}{3}$ then
            $b := 1$;
        else
            n := n + 1;
        end if
    end while
    if $x \geq \frac{2}{3}$ then
        if $b = 0$ then  return "no";
        if $b = 1$ then  return "yes";
    else
        return Charac $(\frac{3}{2}x,\ (\frac{3}{2}a_i - \frac{1}{2}b)_{i\in\mathbb{N}})$
    end if
end
```

and let f be defined by an infinite sum:

$$f(x) := \sum_{n\in\mathbb{N}} 2^{-n} f_n(2^n x).$$

This sum converges in supremum norm and f is a continuous function. To see that a sequence of piece-wise linear approximations to f in L_1-norm can be computed from p note that

$$\int 2^{-n} f_n(2^n x)dx = 2^{-2n} \int f_n(x)dx = 2^{-2(n+1)-m_n}.$$

Thus, the infinite sum may be approximated by the finite sums

$$\sum_{n \text{ s.t. } \max\{n,m_n\}\leq k} 2^{-n} f_n(2^n x)$$

up to any desired precision ε by choosing an appropriate k. Furthermore, these finite sums are computable from the input sequence p_n. An application of a solution of ι^{-1} provides with this function as a continuous function. In particular we may evaluate the values of the function in the peaks and can read the value of $\chi_{\{k\mapsto 0\}}(p_n)$ from these. ◀

C The BSS RAM model – some details

In the following, an algebraic structure $\mathcal{A}$ is a tuple $(U; c_1, c_2, \ldots; f_1, f_2, \ldots; r_1, r_2, \ldots)$ with universe $U_{\mathcal{A}} = U$, where U is any nonempty set, each c_i is an element in U, each f_i is a (partial) function of type $f_i :\subseteq U^{m_i} \to U$ $(m_i \geq 1)$, and each r_i is a relation of type $r_i \subseteq U^{k_i}$. Here, any $\mathcal{A}$-machine $\mathcal{M}$ computes a function of type $g :\subseteq U^* \to U^*$. It has

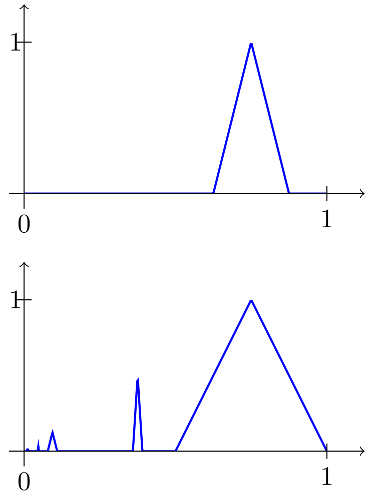


Figure 2 f_n for $m_n = 1$ and f for $m_n = (0, 3, \infty, 1, 6, \infty, \ldots)$.

registers $(Z_i)_{i \geq 1}$ and index registers $(I_j)_{j \in \{1,\ldots,k_{\mathcal{M}}\}}$ and, at any time, the content $c(Z_i)$ of any Z_i is an element of U and any I_j holds a natural number $c(I_j)$. Each program $\mathcal{P}_{\mathcal{M}}$ of any $\mathcal{M}$ is a finite list of labeled commands of the following forms: **computation instructions** $\ell:\ Z_j := f_i^{m_i}(Z_{j_1}, \ldots, Z_{j_{m_i}})$ and $\ell:\ Z_j := c_i^0$, **copy instructions** $\ell:\ Z_{I_j} := Z_{I_k}$, **branching instructions** $\ell:$ if $r_i^{k_i}(Z_{j_1}, \ldots, Z_{j_{k_i}})$ then goto ℓ_1 else goto ℓ_2, **index instructions** $\ell:$ if $I_j = I_k$ then goto ℓ_1 else goto ℓ_2, $\ell:\ I_j := 1$, and $\ell:\ I_j := I_j + 1$, a **stop instruction** $l :$ stop.

For explaining some details, let $(\vec{x} \,.\, \vec{y}) = (x_1, \ldots, x_n, y_1, \ldots, y_m) \in U^{n+m}$ and $(\vec{x} \,.\, \vec{z}) = (x_1, \ldots, x_n, z_1, z_2, \ldots) \in U^{\omega}$ for $n, m \geq 0$ and $\vec{x} = (x_1, \ldots, x_n) \in U^n$, $\vec{y} = (y_1, \ldots, y_m) \in U^m$, and $\vec{z} = (z_1, z_2, \ldots) \in U^{\omega}$. For any $\mathcal{A}$-machine $\mathcal{M}$, let $\{(\ell \,.\, \vec{\iota} \,.\, \vec{z}) \mid \ell \in \mathcal{L}_{\mathcal{M}}\ \&\ \vec{\iota} \in \mathbb{N}_+^{k_{\mathcal{M}}}\ \&\ \vec{z} \in U^{\omega}\}$ be the space of all possible configurations of $\mathcal{M}$ with the list $\mathcal{L}_{\mathcal{M}} =_{\mathrm{df}} \{1, \ldots, \ell_{\mathcal{M}}\}$ of labels. For computing partial functions of the form $f :\subseteq U^* \to U^*$, we use $\mathcal{A}$-machine $\mathcal{M}$ – so-called BSS RAM's over $\mathcal{A}$ – with at least one constant c_1 and $k_{\mathcal{M}} \geq 2$ and an input and an output procedure. Let the input procedure of $\mathcal{M}$ be determined by $\mathrm{Input}_{\mathcal{M}}(x_1, \ldots, x_n) = (1 \,.\, \vec{\iota} \,.\, (x_1, \ldots, x_n, x_n, x_n, \ldots))$ with $\vec{\iota} = (n, 1, \ldots, 1) \in \mathbb{N}_+^{k_{\mathcal{M}}}$ and $\mathrm{Input}_{\mathcal{M}}(()) = (1 \,.\, \vec{\iota} \,.\, (c_1, c_1, \ldots))$ with $\vec{\iota} = (1, 2, 1, \ldots, 1) \in \mathbb{N}_+^{k_{\mathcal{M}}}$. Let the output procedure be given by $\mathrm{Output}_{\mathcal{M}}(\ell \,.\, \vec{\iota} \,.\, \vec{z}) = (z_1, \ldots, z_{\iota_1})$ if $(\iota_1, \iota_2) \neq (1, 2)$ and $\mathrm{Output}_{\mathcal{M}}(\ell \,.\, \vec{\iota} \,.\, \vec{z}) = ()$ for $(\iota_1, \iota_2) = (1, 2)$. This means, at the start of the computation, the register I_1 contains the length of the input, all other I_j start at 1. The input is in $Z_1, \ldots, Z_n$, all other Z_i contain the constant $c_1 \in U_{\mathcal{A}}$. If the program fails to halt on some input, the computed function is undefined on these values.

For $\mathbb{R}_0 = (\mathbb{R}; 0, 1, q_1, q_2, \ldots; -, +, \times; <, =)$ with $\{q_1, q_2, \ldots\} = \mathbb{Q}$, let $\mathsf{M}_{\mathbb{R}_0}$ be the class of all BSS machines without irrational constants which can be considered to be a BSS RAM over $\mathbb{R}_0$. For a universal register machine, as inputs we use the first part of the code for encoding the program by means of Gödel numberings gn as given in [22] and the second part for the constants. For any $\mathbb{R}_0$-machine $\mathcal{M}$ with the constants in $\vec{c}^{\,(\mathcal{M})} = (c_{j_1}, \ldots, c_{j_{n_1}})$, let $\mathrm{code}(\mathcal{M}) = (\mathrm{code}(\mathcal{P}_{\mathcal{M}}) \,.\, \vec{a}^{\,(\mathcal{M},1)})$ where $\mathrm{code}(\mathcal{P}_{\mathcal{M}}) \in \{1\}^{gn(\mathcal{P}_{\mathcal{M}})}$ and $\vec{a}^{\,(\mathcal{M},a)} = (a_1, \ldots, a_{\ell_{\mathcal{M}}})$ is defined as follows. For any $\ell \leq \ell_{\mathcal{M}}$, let a_ℓ be the i^{th} component c_{j_i} in $\vec{c}^{\,(\mathcal{M})}$ if the ℓ^{th} instruction of $\mathcal{P}_{\mathcal{M}}$ is the instruction $Z_j := c_i^0$ for some j and otherwise let $a_\ell = a$. We know that there is a universal BSS RAM $\mathcal{M}_0 \in \mathsf{M}_{\mathcal{A}}$ over $\mathbb{R}_0$ satisfying $\mathcal{M}_0(\mathrm{code}(\mathcal{M}) \,.\, \vec{x}) = \mathcal{M}(\vec{x})$ for all $\vec{x} \in U_{\mathcal{A}}^{\infty}$ and any BSS RAM $\mathcal{M}$ over $\mathbb{R}_0$ ($U_{\mathcal{A}}^{\infty}$ contains all tuples/strings in $U_{\mathcal{A}}^{*}$

without the empty string). Any simple 1-tape Turing machine M computing a function $f :\subseteq \{0,1\}^* \to \{0,1\}^*$ or $f :\subseteq \mathbb{N} \to \mathbb{N}$ can be simulated by $\mathcal{A}_0$-machines $\mathcal{M}^{\mathrm{T}}(M)$ and $\mathcal{M}^{\mathrm{T}}_{\mathbb{N}}(M)$, respectively, for $\mathcal{A}_0 = (\{0,1\}; 0, 1; ; =)$ with suitable input and output procedures (for details see [22]) and thus be encoded by the Gödel number $\mathrm{code}(M) = gn(\mathcal{P}_{\mathcal{M}^{\mathrm{T}}(M)})$. By analogy, for computing a function $f :\subseteq \mathbb{R} \to \mathbb{R}$ by a type-2 machine M, we can take an $\mathcal{A}_0$-machine with modified input and output for simulating M. A **BSS+Comp machine** $\mathcal{M}$ is a generalization of a BSS machine that can additionally execute instructions of the form

$$Z_j := Comp(I_1, Z_1). \tag{1}$$

If $c(I_1)$ is the code of some type-2 machine M and M computes, on input $c(Z_1)$, the name of a real value, then, by (1), this value is assigned to Z_j and, otherwise, (1) causes that $\mathcal{M}$ does not halt.

Now, we define oracle instructions similar to instructions in Moschovakis' model using the ν-operator (cf. [29]) and use an operator $\vec{\nu}$ in order to introduce the deterministic measure operator λ and other operators (cf. [23]). A consequence is that we can get – in the same way – a precise definition of oracle instructions that can be considered as a generalization of instructions introduced in [13, p. 156] for characterizing counting complexity classes for numeric computations by using classes such as $\sharp\mathrm{P}_{\mathbb{R}}$ introduced in [28, p. 44]. For $f :\subseteq \mathbb{R}^\infty \to \mathbb{R}^*$, let the $\vec{\nu}$-operator here provide a total function $\vec{\nu}[f]$ from $\mathbb{R}^*$ into the power set $\mathcal{P}(\mathbb{R}^\infty)$ of $\mathbb{R}^\infty$. For $\vec{x} \in \mathbb{R}^*$, let $\vec{\nu}[f](\vec{x}) = \{\vec{y} \in \mathbb{R}^\infty \mid f(\vec{x} \,.\, \vec{y}) = 0\}$. We are interested in measures of such sets for a universal function $f_{\mathrm{BSS-uni}} : \mathbb{R}^\infty \to \mathbb{R}^*$ with $f_{\mathrm{BSS-uni}}(\vec{z}) = \mathcal{M}(\vec{y})$ if if there are an $\mathcal{M} \in \mathsf{M}_{\mathbb{R}_0}$, a $k \geq 1$ and a $\vec{y} \in \mathbb{R}^k$ with $\vec{z} = (\mathrm{code}(\mathcal{M}) \,.\, k \,.\, \vec{y})$ and $f_{\mathrm{BSS-uni}}(\vec{z}) = 0$ otherwise. Let $\lambda(A)$ be the Lebesgue measure of a set $A \subseteq \mathbb{R}^k$ if A is in the considered σ-algebra, and undefined otherwise. The **measure operator** $\lambda[f_{\mathrm{BSS-uni}}]$ provides, for any k, a partial function from $\mathbb{R}^*$ into the interval $[0, \infty]$. Let

$$\ell :\ Z_j := \lambda[f_{\mathrm{BSS-uni}}](I_1, Z_1, \ldots, Z_{I_2}, I_3). \tag{2}$$

If there is an $\mathcal{M} \in \mathsf{M}_{\mathbb{R}_0}$ with Gödel number $gn(\mathcal{P}_{\mathcal{M}})$ stored in I_1 and the constants stored in $Z_1, \ldots, Z_{c(I_2)}$ that decides, for $k = c(I_3)$, $A =_{\mathrm{df}} \vec{\nu}[f_{\mathrm{BSS-uni}}](code(\mathcal{M}) \,.\, k) \cap [0,1]^k$ on the interval $[0,1]^k$ and $\lambda(A)$ is defined and finite (and, thus, in $[0, \infty[$, then by (2) $\lambda(A)$ is assigned to the register Z_j. Otherwise, an instruction of the form (2) causes that a machine trying to execute the instruction and to compute the corresponding measure does not halt.

For determining the limits of sequences $(y_i)_{i \in \mathbb{N}}$ with elements in $\mathbb{R}$, two further deterministic operators, the **limit operator** lim (resp. the **u-limit operator** u-lim for uncontrolled limits) and the **c-limit operator** c-lim (for controlled limits), are available and they can be used in executing instructions of the form

$$\ell :\ Z_j := [\text{c-}]\lim[f](I_1, Z_1, \ldots, Z_{I_2}), \tag{3}$$

These operators correspond to the strongly analytic and to the weakly analytic machines going back to Hotz [14]. The differences between both instructions are partially comparable to those of weakly analytic and strongly analytic machines over $(\mathbb{R}; \mathbb{R}; +, -, \cdot, /; =, <)$ that compute the limits by producing sequences $y_0, y_1, \ldots$ weakly analytically and strongly analytically, respectively; for details see the paper [18, p. 4] on relationships between the BSS machines over real numbers (cf. [1]) and the analytic machines (cf. [14]).

For any permitted function $f :\subseteq \mathbb{R}^\infty \to \mathbb{R}^*$, $[\text{c-}]\lim[f]$ is a function of the form $g :\subseteq \mathbb{R}^* \to [-\infty, +\infty]$. For $f :\subseteq \mathbb{R}^\infty \to \mathbb{R}^*$ and $\vec{x} \in \mathbb{R}^*$, $\lim[f](\vec{x})$ is defined if and only if there are a convergent sequence $y_0, y_1, \ldots \in \mathbb{R}$ and an $\mathcal{M} \in \mathsf{M}_{\mathbb{R}_0}$ whose Gödel number $gn(\mathcal{P}_{\mathcal{M}})$

is stored in I_1 and whose constants are stored in $Z_1, \ldots, Z_{c(I_2)}$ that computes y_i on i for all $i \in \mathbb{N}$ and then $\lim[f](\vec{x})$ is the limit of this sequence; c-$\lim[f](\vec{x})$ is equal to the limit if, moreover, $|y_i - y_j| < 2^{-\min\{i,j\}}$ is satisfied for all $i, j \in \mathbb{N}$, and undefined otherwise. Let $f_{\mathrm{BSS-enu}}$ be the function f satisfying $f(\vec{z}) = 0$ if there is a BSS machine $\mathcal{M}$ satisfying $(\forall i \in \mathbb{N})(\exists y_i \in \mathbb{R})(\vec{z} = (\mathrm{code}(\mathcal{M}) \,.\, y_i)$ & $\mathcal{M}(i) = \ y_i)$ and $f(\vec{z}) = 1$ otherwise. This implies that $\{y_i \mid i \in \mathbb{N}\} = \nu[f_{\mathrm{BSS-enu}}](\mathrm{code}(\mathcal{M}))$ for any BSS machine $\mathcal{M}$. Then, the instructions (3) with $f = f_{\mathrm{BSS-enu}}$ allow to compute the corresponding finite limit if the limit of the sequence enumerating by $\mathcal{M}$ exists and is in $\mathbb{R}$ and, otherwise, a machine trying to execute such an instruction does not halt.

A **BSS+$\lambda[f_{\mathrm{BSS-uni}}]$ machine** is a generalization of a BSS machine which can additionally execute instructions of the form (2). A function g is **BSS+λ-computable** if g is computable by a BSS+$\lambda[f_{\mathrm{BSS-uni}}]$ machine. A function g is **BSS+λ+λ-computable** if g is computable by a BSS+$\lambda[f_{\mathrm{BSS+\lambda-uni}}]$ machine. In a similar way, we can define the computability by means of other generalizations of BSS machines as follows. A **BSS+[c-]** $\lim[f_{\mathrm{BSS-enu}}]$ **machine** is a generalization of a BSS machine which can additionally execute the corresponding instructions of the form (3). A function g is a **BSS+[c-]** lim**-computable** if g is computable by a BSS+[c-] $\lim[f_{\mathrm{BSS-enu}}]$ machine. A function g is **BSS+**lim+c-lim**-computable** if g is computable by a BSS+c-$\lim[f_{(\mathrm{BSS+lim})-\mathrm{enu}}]$ machine.

A Partial Metric Semantics of Higher-Order Types and Approximate Program Transformations

Guillaume Geoffroy
University of Bologna, Department of Computer Science and Engineering, Italy
guillaume.geoffroy@unibo.it

Paolo Pistone
University of Bologna, Department of Computer Science and Engineering, Italy
paolo.pistone2@unibo.it

Abstract

Program semantics is traditionally concerned with program equivalence. However, in fields like approximate, incremental and probabilistic computation, it is often useful to describe *to which extent* two programs behave in a similar, although non equivalent way. This has motivated the study of program (pseudo)metrics, which have found widespread applications, e.g. in differential privacy. In this paper we show that the standard metric on real numbers can be lifted to higher-order types in a novel way, yielding a metric semantics of the simply typed lambda-calculus in which types are interpreted as quantale-valued partial metric spaces. Using such metrics we define a class of higher-order denotational models, called diameter space models, that provide a quantitative semantics of approximate program transformations. Noticeably, the distances between objects of higher-types are elements of functional, thus non-numerical, quantales. This allows us to model contextual reasoning about arbitrary functions, thus deviating from classic metric semantics.

2012 ACM Subject Classification Theory of computation → Denotational semantics

Keywords and phrases Simply typed $\boldsymbol{\lambda}$-calculus, program metrics, approximate program transformations, partial metric spaces

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.23

Funding *Guillaume Geoffroy*: ERC CoG 818616 “DIAPASoN”, ANR 16CE250011 “REPAS”.
Paolo Pistone: ERC CoG 818616 “DIAPASoN”, ANR 16CE250011 “REPAS”.

1 Introduction

In program semantics one is usually interested in capturing notions of behavioral equivalence between programs. However, in several fields like approximate [34], incremental [10, 2] and probabilistic [13] computation, it is often more useful to be able to describe *to which extent* two programs behave in a similar, although non equivalent way, so that one can measure the change in the result produced by replacing one program by the other one.

This idea has motivated much literature on program (pseudo)metrics [4, 41, 5, 19, 6, 13, 11, 14, 21], that is, on semantics in which types are endowed with a notion of distance measuring the differences in their behaviors. This approach has found widespread applications, for example in differential privacy [35, 3, 7], where one is interested in measuring the *sensitivity* of a program, i.e. its capacity to amplify changes in its inputs, and in the study of probabilistic processes [16, 43, 11, 42].

Recent literature [44, 32] has highlighted the importance of *contextuality* to reason about program similarity: many common situations require to measure the error produced by a transformation of the form $\mathtt{C}[t] \rightsquigarrow \mathtt{C}[u]$, which replaces a program t by u *within a context* $\mathtt{C}[\ ]$, as a function of the mismatch between t and u and of the sensitivity of the context $\mathtt{C}[\ ]$ itself. For instance, the error produced by replacing the program $\lambda x.\sin(x)$ by the identity function $\lambda x.x$ in a given context $\mathtt{C}$ will be highly sensitive to how close to 0 these functions are

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 23; pp. 23:1–23:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

evaluated in C. Similar cases of contextual reasoning can be found in many areas of computer science: for example in techniques from numerical analysis (e.g. the Gauss-Newton method), in which a computationally intensive function is replaced by its Taylor's expansion around some given point, or in approximate computing techniques like *loop perforation* [38], in which a compiler can be asked to skip a certain number of iterations of a loop in a program.

The Problem of Coupling Program Metrics with Higher-Order Types. While several frameworks for contextual reasoning have been developed in recent years [35, 20, 5, 44, 32], these approaches suggest that describing program similarity for a fully higher-order language in terms of program metrics still constitutes a major challenge.

In particular, when considering higher-order languages with a type Real for real numbers, it is not clear how to *lift* the standard metric on Real to higher-order types, e.g. to Real → Real, so that the distances between higher-order programs are measured in a contextual way.

A standard solution is to take the sup-distance, that is, to let, for $f, g : \mathsf{Real} \to \mathsf{Real}$, $d(f,g) = \sup\{d(f(r), g(r)) \mid r \in \mathsf{Real}\}$. This solution works well in models in which programs are interpreted as *non-expansive* or *Lipschitz-continuous* maps [25, 5]. However such models are not *cartesian-closed*[1], so they do not account for the simply-typed lambda-calculus in its full generality, but only for linear or sub-exponential variations of it (such as Fuzz [35, 20, 5]). Also, it has been shown [13] that in a probabilistic setting the non-linearity of higher-order programs has the effect of *trivialising* metrics, that is, of forcing distances to be either 0 or 1, hence collapsing program distances onto usual notions of program equivalence. Most importantly, even if one restricts to a sub-exponential language, the sup-distance is inadequate to account for contextual transformations as the replacement of $\lambda x. \sin(x)$ by $\lambda x.x$ around 0, as the sup-distance between these two programs is infinite (see Fig. 3).

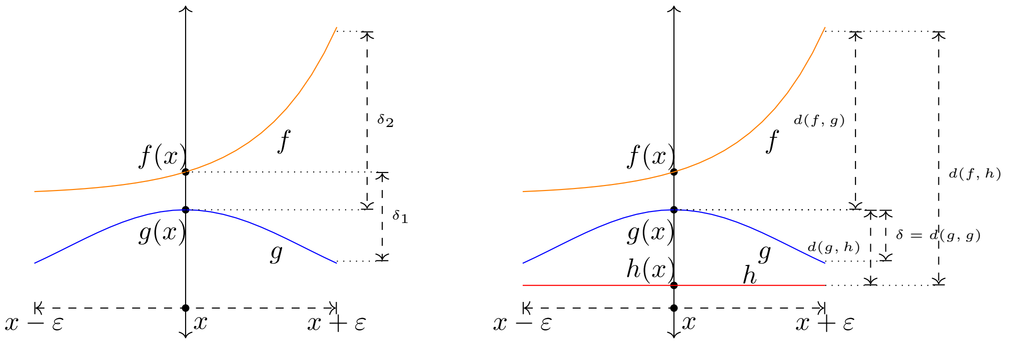
On the other side of the coin, other approaches like [44, 32] are fully contextual and higher-order, but provide, at best, only weak approximations of a standard notion of metric. Nonetheless, these approaches introduce the idea, which we retain here, that program differences must be taken as being themselves some kind of programs, relating errors in input with errors in output, and that accordingly, programs should be split in two different classes: *exact* programs, computing mappings from well-defined inputs to well-defined outputs, and *approximate* programs, mapping errors in the input to errors in the output.

Diameter Spaces. In this paper we introduce a new semantic framework to reason about program similarity and approximate program transformations based on a class of higher-order denotational models that we call *diameter space models.* Compared to existing higher-order frameworks, the main novelty of these models is that program similarities are measured by associating each simple type with a *generalized partial metric space*, yielding a lifting of the standard metric on Real to higher-order types.

Generalized partial metric spaces are a well-investigated class of metric spaces that has been widely applied in program semantics [8, 9, 33, 37, 36, 26, 23]. Such spaces generalize standard metric spaces in that distances need not be real numbers, but can be functions or any other type of object that lives in a suitable *quantale* [25], and *self-distances* $d(x,x)$ need not be 0 (which leads to a stronger triangular inequality: $d(x,y) + d(z,z) \leq d(x,z) + d(z,y)$).

In our models a higher-order type A is interpreted as a 4-tuple $(|A|, [\![A]\!], (\!|A|\!), \delta_A)$ called a *diameter space*, where $|A|$ is a set of *exact* values, $[\![A]\!] \subset \mathcal{P}(|A|)$ is a complete lattice of *approximate* values, $(\!|A|\!)$ is a quantale, and $\delta_A : [\![A]\!] \to (\!|A|\!)$ is a function, called *diameter*,

[1] In fact, cartesian closed categories of metric spaces and non-expansive functions *do* exist [19, 12], but, to our knowledge, none of these categories contains the real numbers with the standard metric.



(a) In differential logical relations the distance between two functions $f, g : \mathbb{R} \to \mathbb{R}$, computed at (x, ε) is the maximum between $\delta_1 = \max\{d(f(x), g(y));\ y \in [x-\varepsilon, x+\varepsilon]\}$ and $\delta_2 = \max\{d(g(x), f(y));\ y \in [x-\varepsilon, x+\varepsilon]\}$.

(b) The distance arising from differential logical relations is not a partial metric: the example above shows that $d(f,h) > d(f,g) + d(g,h) - d(g,g)$ (with all distances computed at (x, ε)).

Figure 1 Differential logical relations do not yield partial metrics.

which provides a quantitative measure of approximate values. The map δ_A generalizes some properties of the diameter function of the standard metric on real numbers. In particular, just like the distance between two real numbers can be described as the diameter of the smallest interval containing them, the map δ_A yields a generalized partial metric $d_A : |A| \times |A| \to \llparenthesis A \rrparenthesis$ in which the distance between two exact values of A is measured as the diameter of the smallest approximate value containing them, i.e. $d_A(x, y) = \delta_A(x \vee y)$.

Measuring Distances between Programs of Functional Type. A primary source of inspiration for our approach is the recent work by Dal Lago, Gavazzo and Yoshimizu on *differential logical relations* [32]. This is a semantical framework for higher-order languages in which a type is interpreted as a set X endowed with a kind of metric structure expressed by a ternary relation $\rho \subseteq X \times Q \times X$, where Q is an arbitrary quantale. To our knowledge, this is the first place were the idea of varying the quantales in which distances are measured is introduced as a key ingredient to obtain a cartesian closed category.

However, although such a relation ρ induces a distance function $d_\rho(x, y) = \sup\{\varepsilon \mid \rho(x, \varepsilon, y)\}$, this function is not a (partial) metric. We can show this fact with a simple example: in this model the distance between two programs $f, g : \mathsf{Real} \to \mathsf{Real}$ is taken in the quantale of functions from $\mathbb{R} \times \mathbb{R}_+^\infty$ to $\mathbb{R}_+^\infty$: intuitively, $d(f, g)$ associates a closed interval $[x-\varepsilon, x+\varepsilon]$ (corresponding to the pair (x, ε)) with the smallest distance δ such that $[f(x) - \delta, f(x) + \delta]$ and $[g(x) - \delta, g(x) + \delta]$ both contain the images of $[x - \varepsilon, x + \varepsilon]$ through g and f respectively (see Fig. 1a). Then, as shown in Fig. 1b, by letting $\delta = d(g, g)(x, \varepsilon)$, we have that $d(g, g)$ sends the interval $I = [x - \varepsilon, x + \varepsilon]$ onto the interval $[g(x) - \delta, g(x) + \delta]$, which has diameter 2δ, while the image of I has diameter δ, making the triangular law of partial metrics fail.

By contrast, in our model, the distance between two programs $f, g : \mathsf{Real} \to \mathsf{Real}$ lives in the quantale of monotone maps from *approximate values* of Real (i.e. closed intervals) to positive reals. More precisely, this distance is the function that maps a closed interval a to the diameter of the smallest interval containing *both* $f(a)$ and $g(a)$. This notion of distance does satisfy all the axioms of a partial metric, as illustrated in Fig. 2. Observe that we no longer depict the "center" of the interval $[x - \varepsilon, x + \varepsilon]$, and that the triangular inequality works because in summing $d(f, g)$ and $d(g, h)$ the self-distance $d(g, g)$ is counted twice.

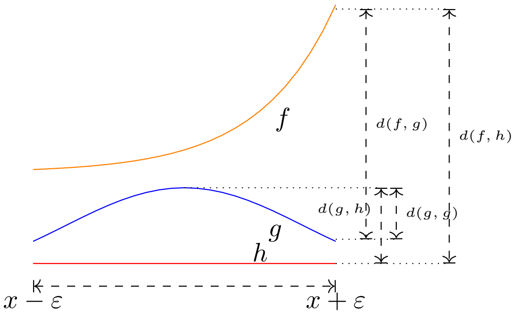


Figure 2 Our new metric is a partial metric: in the example above it can be seen that $d(f,h) \leq d(f,g) + d(g,h) - d(g,g)$ (with all distances computed in the interval $[x-\varepsilon, x+\varepsilon]$).

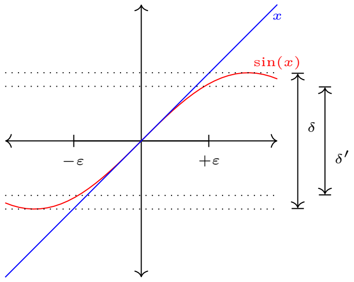


Figure 3 The self-distances δ, δ' of $\sin(x)$ and x in a small interval $[-\varepsilon, \varepsilon]$ of 0 are very close.

Note that the distance of f *from itself*, which needs not be (constantly) 0, provides a measure of the sensitivity of f, since it associates each interval a with the size of the interval $f(a)$ spanned by f on a (a similar feature is present in differential logical relations).

The use of partial metrics with functional distances yields a rich and expressive framework to reason about contextual transformations. For instance, we can express the closeness of $\lambda x.\sin(x)$ and $\lambda x.x$ around 0 by the fact that their distance, applied to a small interval $[-\varepsilon, \varepsilon]$ around 0, is very close to the self-distance of $\lambda x.\sin(x)$ on the same interval (as illustrated in Fig. 3). Moreover, the triangular inequality of partial metrics can be used to infer new bounds from previously established ones in a compositional way.

Diameter Space over a Cartesian Closed Category. Our approach was devised primarily to account for transformations in higher-order languages designed for real analysis computation (like e.g. Real PCF [18]). However, diameter spaces can be constructed starting from any higher-order programming language with a reasonable denotational semantics. In fact, for *any* cartesian closed category $\mathbb{C}$, we can construct a cartesian *lax*-closed category $\mathrm{Diam}(\mathbb{C})$, whose morphisms can be seen as approximate versions of the morphisms of $\mathbb{C}$. The "lax" preservation of the cartesian closed structure reflects the fact that, by composing approximations in a higher-order setting, also their error rates compose (typically, approximating non β-normal λ-terms will lead to higher error-rates than approximating their β-normal forms).

The generality of our construction shows in particular that our partial metric semantics requires no restrictions (e.g. Lipschitz-continuity) on morphisms, and adapts well to the model one starts with: for instance, the category $\mathrm{Diam}(\mathrm{Set})$ contains a partial metric on the set of *all* set-theoretic functions from $\mathbb{R}$ to $\mathbb{R}$, while the categories $\mathrm{Diam}(\mathrm{Eff})$ (where Eff is the *effective topos* [27]) and $\mathrm{Diam}(\mathrm{Scott})$ show that our approach scales well to a more computability-minded setting.

2 Generalized Partial Metric Spaces

Partial metric spaces were introduced in the early nineties as a variant of metric spaces in which self-distances can be non-zero. Such spaces have attracted much attention in program semantics [8, 9, 33, 37, 36, 26, 23], due to their compatibility with standard constructions from both domain theory (since their topology is T_0) and usual metric topology (e.g. Cauchy sequences, completeness, Banach-fixed point theorem) [8, 33]. *Generalized partial metric spaces*, i.e. partial metric spaces whose metric takes values over an arbitrary quantale [25], are well-investigated too [29, 28].

In this paper we will only be concerned with partial metrics taking values over a *commutative integral* quantale [25], of which we recall the definition below.

▶ **Definition 1.** *A* commutative integral quantale *is a triple* $(Q, +, \leq)$ *where:*

- $(Q, \leq)$ *is a complete lattice,*
- $(Q, +)$ *is a commutative monoid,*
- $+$ *commutes with arbitrary* $\inf$*s,*
- *the least element of* Q *is neutral for* $+$*.*

For readability, we have we have reversed the ordering with respect to the conventional definition, so that for example, $([0, \infty], +, \leq)$ is a commutative integral quantale whose least element is 0 (as opposed to "$([0, \infty], +, \geq)$ is a commutative integral quantale whose *largest* element is 0", which is what we would get with the usual definition). It is straightforward to check that for all commutative integral quantales Q, R, the product monoid $Q \times R$ equipped with the product ordering is also a commutative integral quantale. In addition, for all posets X, the set of monotone functions from X to Q, equipped with the pointwise monoid operation and the pointwise ordering, is also a commutative integral quantale. Another example of commutative integral quantale is given by the lattice of ideals of any commutative ring, with the product of ideals as the monoid operation.

We recall now the definition of a generalized partial metric space:

▶ **Definition 2.** *A* generalized partial metric space *(in short, GPMS) is the data of a set* X*, a commutative integral quantale* Q *and a function* $d : X \times X \to Q$ *such that:*

- *for all* $x, y \in X$*,* $d(x, x) \leq d(x, y)$*,*
- *for all* $x, y \in X$*, if* $d(x, x) = d(x, y) = d(y, y)$*, then* $x = y$*,*
- *for all* $x, y \in X$*,* $d(x, y) = d(y, x)$*,*
- *for all* $x, y, z \in X$*,* $d(x, z) + d(y, y) \leq d(x, y) + d(y, z)$*.*

For every metric space (X, d), the structure $(X, ([0, \infty], +, \leq), d)$ is a GPMS. As is well-known [8], any real-valued GPMS $(X, [0, \infty], d)$ induces a metric d^* by letting

$$d^*(x, y) = 2d(x, y) - d(x, x) - d(y, y) \qquad (\star)$$

For a more telling and somewhat archetypal example, take any set X and consider the set $X^{\leq\omega}$ of all sequences of elements of X indexed by an ordinal less than or equal to ω. For all such sequences s, t, let $d(s, t) = 2^{-n} \in [0, \infty]$, where n is the length of the largest common prefix to s and t: one can check that $(X^{\leq\omega}, [0, \infty], d)$ is indeed a generalized partial metric space. In fact, if we interpret the prefixes of a sequence as pieces of partial information, then we have $d(s, s) = d(s, t)$ if and only if t is a refinement of s (i.e. if it contains more information), and $d(s, s) = 0$ if and only if s is total (i.e. if it cannot be refined).

One can check that for all partial metric spaces (X, Q, d_X) and (Y, R, d_Y), $(X \times Y, Q \times R, d_{X\times Y})$ is a generalized partial metric space, where $d_{X\times Y}((x_1, y_1), (x_2, y_2)) = (d_X(x_1, x_2), d_Y(y_1, y_2))$. However, in general, it is not clear how one should define a partial metric on a function space. In Section 3.2 we introduce a construction to obtain partial metric spaces on function spaces by generalizing some properties of the standard diameter function on sets of real numbers.

3 Approximate Programs for the Simply-Typed λ-Calculus over Real

To illustrate our construction, we start from a relatively concrete example: we consider a simply-typed lambda calculus with a base type Real and primitives for real numbers, and we follow the plan outlined in the introduction, which yields for each simple type a notion of approximate value, approximate function, diameter and distance between programs. Most definitions are straightforward and intuitive: the interesting, not immediately obvious point is that our construction does yield a *partial metric* on each type.

Simple types are defined as follows: Real is a simple type; if A and B are simple types, then $A \to B$ and $A \times B$ are simple types. For all $n > 0$, we fix a set $\mathcal{F}_n$ of functions from $\mathbb{R}^n$ to $\mathbb{R}$. We consider the usual Curry-style simply-typed λ-calculus over the types defined above (the left and right projection are denoted by $\pi_L : A \times B \to A$ and $\pi_R : A \times B \to B$ respectively, and the constructor for pairs by $\langle -, - \rangle$), enriched with the following constants: for all $r \in \mathbb{R}$, a constant $r : \mathsf{Real}$; for all $n > 0$ and all $f \in \mathcal{F}_n$, a constant $f : \mathsf{Real} \to \ldots \to \mathsf{Real} \to \mathsf{Real}$. We call this calculus $\mathsf{ST\lambda C}(\mathcal{F}_n)$, and its terms are simply called *terms*. We write $t[x_1 := u_1, \ldots, x_n := u_n]$ to denote the *simultaneous* substitution of $u_1, \ldots, u_n$ for $x_1, \ldots, x_n$ in t. For all types A, we denote by Λ_A the set of closed terms of type A. The relation of β-reduction is enriched with the following rule, extended to all contexts: for all $n > 0$, $f \in \mathcal{F}_n$, and $r_1, \ldots, r_n \in \mathbb{R}$, $f r_1 \ldots r_n \to_\beta s$, where $s = f(r_1, \ldots, r_n)$. By standard arguments [1], this calculus has the properties of subject reduction, confluence and strong normalisation.

▶ Remark 3. The class of real-valued functions which can be computed in $\mathsf{ST\lambda C}(\mathcal{F}_n)$ depends on the choice we make for $\mathcal{F}_n$. With suitable choices (see for instance [40, 17, 18]) one can obtain that all programs of type $\mathsf{Real} \to \mathsf{Real}$ compute *continuous* functions[2], that all such programs are *integrable* over closed intervals, or that all such programs are *continuously differentiable.*

In addition to the usual notion of β-equivalence between terms of $\mathsf{ST\lambda C}(\mathcal{F}_n)$, we will exploit also a stronger equivalence: given two closed terms t, u of type A, we say that t and u are *observationally equivalent* and write $t \approx_A u$ if for all terms C such that $x : A \vdash C : \mathsf{Real}$ is derivable, $C[x := t]$ is β-equivalent to $C[x := u]$ (which amounts to saying that they both β-reduce to the same real number). It is clear that observational equivalence is a congruence and that two β-equivalent terms are always observationally equivalent.

3.1 Approximate Values and Approximate Programs

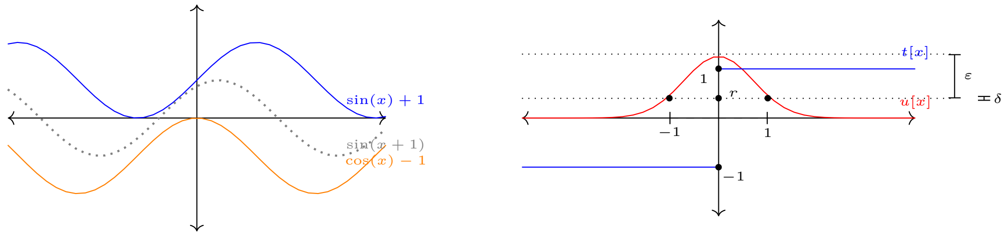
The first step of our construction for $\mathsf{ST\lambda C}(\mathcal{F}_n)$ is to associate to each simple type A a set $[\![A]\!]$ whose elements are certain sets of programs of type A that we call *approximate values of type A.* A closed term $t \in \Lambda_A$ represents a program with return type A and no parameters, so an approximate value can be thought of as a specification of a program with return type A and no parameters *up to* a certain degree of error or approximation.

For each simple type A, the set of approximate values $[\![A]\!] \subseteq \mathcal{P}(\Lambda_A)$ is defined inductively as follows:

- $[\![\mathsf{Real}]\!] = \{\{t \in \Lambda_{\mathsf{Real}} \mid \exists r \in I, t \to_\beta^* r\} \mid I \subseteq \mathbb{R} \text{ is a compact interval or } \emptyset \text{ or } \mathbb{R}\}$,
- $[\![A \times B]\!] = \{a \times b \mid a \in [\![A]\!], b \in [\![B]\!]\}$, where $a \times b = \{t \in \Lambda_{A \times B} \mid \pi_L t \in a \text{ and } \pi_R t \in b\}$,
- $[\![A \to B]\!] = \{\{t \in \Lambda_{A \to B} \mid \forall u \in \Lambda_A,\ tu \in I(u)\} \mid I : \Lambda_A \to [\![B]\!]\}$.

The approximate values of type Real are sets of closed programs of type Real which essentially coincide with the compact intervals of $\mathbb{R}$, plus the empty set and $\mathbb{R}$ itself. An approximate value in $[\![A \times B]\!]$ is a "rectangle" $a \times b$, with $a \in [\![A]\!]$ and $b \in [\![B]\!]$, while an approximate value in $[\![A \to B]\!]$ is uniquely determined by a function I from closed terms $u \in \Lambda_A$ to approximate values $I(u) \in [\![B]\!]$.

[2] Note that for this to be possible, $\mathcal{F}_n$ cannot contain the identity function over Real.



(a) $\lambda x.\sin(x+1)$ is in $[\lambda x.\sin(x)+1, \lambda x.\cos(x)+1]_{\mathsf{Real}\to\mathsf{Real}}$.

(b) $\varepsilon = (\partial(u)\circ\partial(t))([-1,1])$ is bigger than $\delta = \partial(u\circ t)([-1,1]) = [r,r]$.

Figure 4 Examples of functional approximate values and of approximate programs.

For example, any two terms $t,u \in \Lambda_{\mathsf{Real}}$ with normal forms $q,r \in \mathbb{R}$ induce an approximate value $[t,u]_{\mathsf{Real}} = \{v \in \Lambda_{\mathsf{Real}} \mid v \to_\beta^* s \wedge (q \le s \le r \vee q \ge s \ge r)\}$ of type Real. Similarly, any two terms $t,u \in \Lambda_{\mathsf{Real}\to\mathsf{Real}}$ induce an approximate value $[t,u]_{\mathsf{Real}\to\mathsf{Real}} = \{v \in \Lambda_{\mathsf{Real}\to\mathsf{Real}} \mid \forall r \in \Lambda_{\mathsf{Real}}\ vr \in [tr,ur]_{\mathsf{Real}}\}$. For instance, if $t = \lambda x.\sin(x)+1$ and $u = \lambda x.\cos(x)-1$, then $[t,u]_{\mathsf{Real}\to\mathsf{Real}}$ contains all closed terms corresponding to maps oscillating between $\sin(x)+1$ and $\cos(x)+1$ (e.g. the program $\lambda x.\sin(x+1)$, as illustrated in Fig. 4a).

For all A, the set $[\![A]\!]$ is a a subset of $\mathcal{P}(\Lambda_A)$ closed under arbitrary intersections. We deduce that $[\![A]\!]$ has arbitrary meets (given by intersections) and arbitrary joins $\bigvee_{i\in I} a_i = \bigcap\{a \in [\![A]\!] \mid \forall i \in I\ a_i \subseteq a\}$, and thus $[\![A]\!]$ is a complete lattice. In particular, for all $t \in \Lambda_A$, there is a least element of $[\![A]\!]$ that contains t, which will be denoted by $\overline{t}$. One can check that $\overline{t} = \overline{u}$ if and only if $t \approx_A u$.

Monotone functions from approximate values to approximate values represent *approximate programs*. They behave like a model of the simply-typed λ-calculus in a weak sense, namely:

- for all monotone functions $\vec{\alpha} \mapsto c[\vec{\alpha}] : [\![A_1]\!] \times \ldots \times [\![A_n]\!] \to [\![B \to C]\!]$ and $\vec{\alpha} \mapsto b[\vec{\alpha}] : [\![A_1]\!] \times \ldots \times [\![A_n]\!] \to [\![B]\!]$, we can define a monotone function $\vec{\alpha} \mapsto (c[\vec{\alpha}]\ b[\vec{\alpha}]) = \sup\{\overline{vu} \mid v \in c[\vec{\alpha}], u \in b[\vec{\alpha}]\} : [\![A_1]\!] \times \ldots \times [\![A_n]\!] \to [\![C]\!]$,
- for all monotone functions $\vec{\alpha} \mapsto c[\vec{\alpha}] : [\![A_1]\!] \times \ldots \times [\![A_n]\!] \to [\![C]\!]$ and all $i \le n$, we can define a monotone function $(\alpha_j)_{j\ne i} \mapsto (\lambda\alpha_i.\ c[\vec{\alpha}]) = \{v \in \Lambda_{A_i\to C} \mid \forall t_i \in \Lambda_{A_i},\ vt_i \in c[\alpha_1,\ldots,\overline{t_i},\ldots,\alpha_n]\} : \prod_{j\ne i}[\![A_j]\!] \to [\![A_i \to C]\!]$,

and these two constructions are weakly compatible with β-reduction and η-expansion:

▶ **Proposition 4.** *For all monotone functions $(\vec{\alpha},\beta) \mapsto c[\vec{\alpha},\beta] : [\![A_1]\!]\times\ldots\times[\![A_n]\!]\times[\![B]\!] \to [\![C]\!]$ and $\vec{\alpha} \mapsto b[\vec{\alpha}] : [\![A_1]\!] \times \ldots \times [\![A_n]\!] \to [\![B]\!]$, $(\vec{\alpha} \mapsto (\lambda\beta.\ c[\vec{\alpha},\beta])\ b[\vec{\alpha}]) \le (\vec{\alpha} \mapsto c[\vec{\alpha}, b[\vec{\alpha}]])$, and for all monotone functions $\vec{\alpha} \mapsto d[\vec{\alpha}] : [\![A_1]\!] \times \ldots \times [\![A_n]\!] \to [\![B \to C]\!]$, $(\vec{\alpha} \mapsto \lambda\beta.\ d[\vec{\alpha}]\ \beta) \ge (\vec{\alpha} \mapsto d[\vec{\alpha}])$, where functions are ordered by pointwise inclusion. In other words, on approximate programs, β-reduction and η-expansion* discard *information, and conversely β-expansion and η-reduction* recover *some information.*

Proof. Without loss of generality, we can assume $n = 0$. Let $v \in \lambda\beta.\ c[\beta]$ and $u \in b$. By definition, $tu \in c[\overline{u}]$, so $\overline{tu} \subseteq c[\overline{u}] \subseteq c[b]$. Therefore, $(\lambda\beta.\ c[\beta])\ b \subseteq b$. Let $v \in d$. For all $u \in \Lambda_B$, by definition, $vu \in d\overline{u}$. Therefore, $v \in \lambda\beta.\ d\ \beta$. ◀

Beyond theoretical aspects (which will be made clearer in Section 5) Proposition 4 is also important in practice because it implies that if we compute an approximation of a program from approximations of its parts and then simplify the resulting approximate program using β-reduction and η-expansion, what we obtain is still a valid approximation of the original program.

We can define a weak embedding from terms into approximate programs, by mapping each term to its tightest approximation: for all terms t such that $\alpha_1 : A_1, \dots, \alpha_n : A_n \vdash t : B$, we define a monotone function $\partial(t) : [\![A_1]\!] \times \cdots \times [\![A_n]\!] \to [\![B]\!]$ by $\partial(t)(a_1, \dots, a_n) = \sup\{\overline{tu_1 \dots u_n} \mid u_1 \in a_1, \dots, u_n \in a_n\}$.

▶ Remark 5. The map ∂ is constant on classes of observational equivalence, and one can check that it is is weakly compatible with the constructions of the λ-calculus, in particular:

- $\partial(\alpha_i)(a_1, \dots, a_n) = a_i$,
- $\partial(tu)(a_1, \dots, a_n) \subseteq \partial(t)(a_1, \dots, a_n)\ \partial(u)(a_1, \dots, a_n)$,
- $\partial(\lambda\beta.t)(a_1, \dots, a_n) \subseteq \lambda\beta.\ \partial(t)(\beta, a_1, \dots, a_n)$.

This map $\partial(t)$ can be taken as a measure of the *sensitivity* of t, as it maps an interval a, that is a quantifiably uncertain input, to a quantifiably uncertain output $\partial(t)(a)$. For instance, if we take the term $t[x] = \sin(x) + 1$ above, then $\partial(t) : [\![\mathsf{Real}]\!] \to [\![\mathsf{Real}]\!]$ sends the interval $[-\pi, \pi]_{\mathsf{Real}}$ into $[0, 2]_{\mathsf{Real}}$.

▶ Remark 6. When composing two maps $\partial(t)$ and $\partial(u)$, we might obtain a worse approximation than by computing $\partial(t[u/x])$ directly. For instance, let $t[x]$ and $u[x]$ be, respectively, the discontinuous and Gaussian functions illustrated in Fig. 4b. If a is the interval $[-1, +1]$, then $\partial(t)(a) = [-1, 1]$, and since $u[x := -1] = u[x := 1] \simeq_\beta r$ for some $0 < r < 1$, we deduce that $\partial(u)(\partial(t)(a)) = [-1, 1] \supsetneq [r, r] = \partial(u[t/x])(a)$.

3.2 A Partial Metric on Each Type

So far, we have associated each type A of $\mathsf{ST\lambda C}(\mathcal{F}_n)$ with a complete lattice $[\![A]\!] \subseteq \mathcal{P}(\Lambda_A)$ of approximate values of type A, and each typed program $t : A \to B$ with an approximate program $\partial(t)$ (in fact, a monotone function) from approximate values of type A to approximate values of type B. We will now exploit this structure to define, for each type A of $\mathsf{ST\lambda C}(\mathcal{F}_n)$, a generalized partial metric on the closed (exact) programs of type A.

The first step is to define, for every simple type A, a commutative integral quantale $((\!|A|\!), \leq_A, +_A)$ of *distances of type A*:

- $((\!|\mathsf{Real}|\!), \leq_{\mathsf{Real}}, +_{\mathsf{Real}}) = ([0, \infty], \leq, +)$,
- $(\!|A \times B|\!) = (\!|A|\!) \times (\!|B|\!)$,
- $(\!|A \to B|\!) = \mathrm{Poset}([\![A]\!], (\!|B|\!))$.

where, for two posets Q, R, $\mathrm{Poset}(Q, R)$ denotes the set of monotone functions from Q to R. Observe that the quantale $(\!|A \to B|\!)$ is a set of functions over the approximate values of A.

For all simple types A, we now define a *distance function* $d_A : \Lambda_A \times \Lambda_A \to (\!|A|\!)$:

- $d_{\mathsf{Real}}(t, u) = |r - s|$, where r, s are the unique elements of $\mathbb{R}$ such that $t \to_\beta^* r$ and $u \to_\beta^* s$,
- $d_{A \times B}(t, u) = (d_A(\pi_L t, \pi_L u), d_B(\pi_R t, \pi_R u))$,
- $d_{A \to B}(t, u) = a \mapsto \sup\{d_B(rv, sw) \mid r, s \in \{t, u\}, v, w \in a\}$.

It would be tempting to define $d_{A \to B}(t, u)(a)$ simply as $\sup\{d_B(tv, uw) \mid v, w \in a\}$, but then the axiom "$d_{A \to B}(t, t) \leq d_{A \to B}(t, u)$" of partial metric spaces would fail.

The maps d_A are clearly compatible with observational equivalence (i.e. if $a \approx_A a'$ and $b \approx_A b'$, then $d_A(a, b) = d_A(a', b')$).

Our objective is now to prove that $(\Lambda_A / \approx_A, (\!|A|\!), d_A)$ is a generalized partial metric space. To this end, we define for all simple types A a monotone *diameter function* $\delta_A : [\![A]\!] \to (\!|A|\!)$ by $\delta_A(a) = \sup\{d_A(t, u) \mid t, u \in a\}$. The key to our objective will be to prove that δ_A is *sub-modular* on intersecting approximate values (henceforth, *quasi-sub-modular* – see Proposition 7): this generalizes the fact that, on the (real-valued) metric space $\mathbb{R}$, the diameter is *modular* over intersecting closed intervals (see Fig. 5).

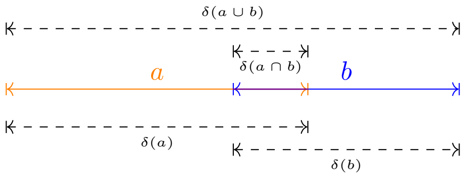


Figure 5 The diameter function is *modular* over intersecting real intervals: $\mathsf{diam}(a \cup b) + \mathsf{diam}(a \cap b) = \mathsf{diam}(a) + \mathsf{diam}(b)$ for all $a, b \in [\![\mathbb{R}]\!]$ such that $a \cap b \neq \emptyset$. This property is at the heart of our generalization of diameters. Observe that this property fails when $a \cap b$ is empty.

First, one can check that for all $t, u \in \Lambda_A$, $\delta_A\left(\overline{t} \vee \overline{u}\right) = d_A(t, u)$, and that:

- $\delta_{\mathsf{Real}}(a) = \sup\{s - r \mid s, r \in \mathbb{R} \text{ such that } s, r \in a\}$,
- $\delta_{\mathsf{A \times B}}(p) = \left(\delta_A\left(\sup\left\{\overline{\pi_L t} \mid t \in p\right\}\right) \delta_B\left(\sup\left\{\overline{\pi_R t} \mid t \in p\right\}\right)\right)$,
- $\delta_{\mathsf{A \to B}}(b) = a \mapsto \delta_B\left(\sup\left\{\overline{vt} \mid t \in a, v \in b\right\}\right)$.

This leads then to the following:

▶ **Proposition 7** (δ_A is quasi-sub-modular)**.** *For all simple types A and all $a, b \in [\![A]\!]$ such that $a \wedge b \neq \emptyset$, $\delta(a \wedge b) + \delta(a \vee b) \leq \delta(a) + \delta(b)$.*

Proof. We proceed by induction on types.

Let $a, b \in [\![\mathsf{Real}]\!]$ such that $a \wedge b \neq \emptyset$. Let $I = \{r \in \mathbb{R} \mid r \in a\}$ and $J = \{s \in \mathbb{R} \mid s \in b\}$: then I (respectively, J, $I \cap J$, $I \cup J$) is either $\mathbb{R}$ or a non-empty compact interval of $\mathbb{R}$, and its length in the usual sense is equal to $\delta_{\mathsf{Real}}(a)$ (respectively, $\delta_{\mathsf{Real}}(b)$, $\delta_{\mathsf{Real}}(a \wedge b)$, $\delta_{\mathsf{Real}}(a \vee b)$). Note that the only reason we know that $I \cup J$ is an interval is because $a \wedge b \neq \emptyset$ implies $I \cap J \neq \emptyset$. The length of an interval of $\mathbb{R}$ is equal to its Lebesgue measure, therefore $\text{length}(I \cap J) + \text{length}(I \cup J) = \text{length}(I) + \text{length}(J)$, so $\delta_{\mathsf{Real}}(a \wedge b) + \delta_{\mathsf{Real}}(a \vee b) = \delta_{\mathsf{Real}}(a) + \delta_{\mathsf{Real}}(b)$.

Let $a, b \in [\![A_L \times A_R]\!]$ such that $a \wedge b \neq \emptyset$. For all $c \in [\![A_L \times A_R]\!]$, let $c_L = \sup\{\overline{\pi_L t} \mid t \in c\}$ and $c_R = \sup\{\overline{\pi_R t} \mid t \in c\}$. One can check that $(a \wedge b)_L = a_L \wedge b_L$, $(a \wedge b)_R = a_R \wedge b_R$, $(a \vee b)_L = a_L \vee b_L$ and $(a \vee b)_R = a_R \vee b_R$, so $\delta(a \wedge b) + \delta(a \vee b) = (\delta(a_L \wedge b_L) + \delta(a_L \vee b_L), \delta(a_R \wedge b_R) + \delta(a_R \vee b_R)) \leq (\delta(a_L) + \delta(b_L), \delta(a_R) + \delta(b_R)) = \delta(a) + \delta(b)$.

Let $f, g \in [\![A \to B]\!]$ and $a \in [\![A]\!]$. For all $h \in [\![A \to B]\!]$, let $ha = \sup\{\overline{vt} \mid v \in h, t \in a\}$. One can check that $(f \wedge g)a \subseteq (fa) \wedge (ga)$ and $(f \vee g)a = (fa) \vee (ga)$. As a result, $(\delta(f \wedge g) + \delta(f \vee g))(a) \leq \delta((fa) \wedge (ga)) + \delta((fa) \vee (ga)) \leq \delta(fa) + \delta(ga) = (\delta(f) + \delta(g))(a)$. ◀

It is well-known [39] that any function $\delta : L \to [0, \infty]$ on a lattice L that is monotone and *sub-modular* induces a pseudo-metric $d : L \times L \to [0, \infty]$ by letting $d^*(a, b) = 2\delta(a \vee b) - \delta(a) - \delta(b)$. In fact, one can decompose this construction: first, one defines a partial pseudometric d on L by $d(a, b) = \delta(a \vee b)$, and then d^* is just the distance given by equation ($\star$): $d^*(a, b) = 2d(a, b) - d(a, a) - d(b, b)$. We can use this way of reasoning to establish that the maps d_A are indeed partial metrics:

▶ **Corollary 8.** *For all simple types A, $(\Lambda_A/\approx_A, (\![A]\!), d_A)$ is a generalized partial metric space, that is to say:*

1. *for all $t, u \in \Lambda_A$, $d_A(t, t) \leq d_A(t, u)$,*
2. *for all $t, u \in \Lambda_A$, if $d_A(t, t) = d_A(t, u) = d_A(u, u)$, then $t \approx_A u$,*
3. *for all $t, u \in \Lambda_A$, $d_A(t, u) = d_A(u, t)$,*
4. *for all $t, u, v \in \Lambda_A$, $d_A(t, v) + d_A(u, u) \leq d_A(t, u) + d_A(u, v)$.*

Proof. As mentioned above, for all $t, u \in \Lambda_A$, $d_A(t,u) = \delta_A(\overline{t} \vee \overline{u})$, which immediately gives point 3. Since δ_A is monotone and $\overline{t} \vee \overline{t} \leq \overline{t} \vee \overline{u}$, we also get point 1.

One can check (by induction on types) that the restriction of δ_A to the ideal generated by the $\overline{t}$ (for $t \in \Lambda_A$) is *strictly* monotone. Therefore, if $d_A(t,t) = d_A(t,u) = d_A(u,u)$, i.e. $\delta_A(\overline{t}) = \delta_A(\overline{t} \vee \overline{u}) = \delta_A(\overline{u})$, then $\overline{t} = \overline{t} \vee \overline{u} = \overline{u}$, so $t \approx_A u$.

The triangular inequality is an immediate consequence of the quasi-sub-modularity of δ_A: $d(t,v)+d(u,u) = \delta(\overline{t}\vee\overline{v})+\delta(\overline{u}) \leq \delta((\overline{t}\vee\overline{u})\vee(\overline{u}\vee\overline{v}))+\delta((\overline{t}\vee\overline{u})\wedge(\overline{u}\vee\overline{v})) \leq \delta(\overline{t}\vee\overline{u})+\delta(\overline{u}\vee\overline{v}) = d(t,u) + d(u,v)$. ◀

4 Computing Program Distances using Partial Metrics

In the previous section we showed how to associate each simple type A with a partial metric d_A over the closed terms of type A. We now illustrate through a few basic examples how the higher-order and metric features of this semantics can be used to formalize contextual reasoning about program differences.

To make our examples more realistic, we will consider some natural extensions of $\mathsf{ST}\lambda\mathsf{C}(\mathcal{F}_n)$. It is not difficult to see that all constructions from Section 3 still work if we add to $\mathsf{ST}\lambda\mathsf{C}(\mathcal{F}_n)$ some new base types. For example, we can add to our language a type Nat for natural numbers, indicating for each $n \in \mathbb{N}$, the corresponding normal forms of Nat as $\mathtt{n}$. A natural choice is to let $[\![\mathsf{Nat}]\!] = \{\{t \mid \exists n \in a\ t \rightsquigarrow \mathtt{n}\} \mid a$ finite subset of $\mathbb{N}$ or $a = \mathbb{N}\}$, $(\!|\mathsf{Nat}|\!) = [0, \infty]$ and $d_{\mathsf{Nat}}(t,u) = |n - m|$, where $t \rightarrow_\beta^* \mathtt{n}$ and $u \rightarrow_\beta^* \mathtt{m}$.

Moreover, our constructions scale well also to extensions of $\mathsf{ST}\lambda\mathsf{C}(\mathcal{F}_n)$ obtained by adding new program constructors, as soon as these do not compromise the existence and uniqueness of normal forms (since the fact that closed programs of type Real have a normal form plays an important role to define $[\![\mathsf{Real}]\!]$). For instance, if we suppose that all programs of type $\mathsf{Real} \to \mathsf{Real}$ in $\mathsf{ST}\lambda\mathsf{C}(\mathcal{F}_n)$ are either differentiable or integrable (see Remark 3), we can consider extension of $\mathsf{ST}\lambda\mathsf{C}(\mathcal{F}_n)$ with differential or integral operators, as in $\mathsf{Real\ PCF}$ [17, 18].

We start with a classical example from approximate computing that we adapt from [44].

▶ **Example 9** (Loop perforation). We work in the extension of $\mathsf{ST}\lambda\mathsf{C}(\mathcal{F}_n)$ with a type Nat. We discuss a transformation that replaces a program t which performs n iterations by a program which only performs the iterations $0, k, 2k, 3k, \dots,$ each repeated k times.

Suppose $t : (A \times A \to A) \to \mathsf{Nat} \to (A \to A) \to A$, for $n \geq 1$, is a term such that $th\mathtt{n}f$ computes the n-times iteration of h as follows: $th\mathtt{0}f = h\langle f\mathtt{0}, f\mathtt{0}\rangle$ and $th(\mathtt{n}+\mathtt{1})f = h\langle th\mathtt{n}f, f(\mathtt{n}+\mathtt{1})\rangle$. Let $\mathsf{Perf}^k(t)$, the k-th perforation of t, be the program $(\mathsf{Perf}^k(t))h\mathtt{n}f = t(\lambda x.(h^{(k)}x))\lfloor \mathtt{n} \rfloor_{\mathtt{k}}(\lambda x.f(x * \mathtt{k})$, where $\lfloor n \rfloor_k$ indicates the least $m \leq n$ such that m is divisible by k, and $x * \mathtt{k}$ is the multiplication of x by k.

To compute the distance $d_A(v_n, w_n)$ between $v_n = th\mathtt{n}f$ and its perforation $w_n = \mathsf{Perf}^k(t)h\mathtt{n}f$ we can reason as follows:

i. v_n performs n-iterations while w_n performs $k\lfloor n \rfloor_k \leq n$ iterations, and we can compute $d_A(v_n, v_{(k\lfloor n \rfloor_k)})$ as the diameter of $\partial(t)\partial(h)([k\lfloor n \rfloor_k, n]_{\mathsf{Nat}})\partial(f)$.

ii. If n is divisible by k, then for $i \leq n$, at the i-th iteration of v_n the function f is applied to $\mathtt{i}$, while at the i-th iteration of w_n, f is applied to $\lfloor i \rfloor_k$. Now, the error of replacing $f\mathtt{i}$ by $f\lfloor \mathtt{j} \rfloor_k$, with $\mathtt{i}, \mathtt{j}$ in some $a \in [\![\mathsf{Nat}]\!]$, is accounted for by the approximate program $c[y] = \partial(f)(y - k)$, where $y - k = y \vee \{u - \mathtt{k} \mid u \in y\}$. We deduce then that $d_A(v_n, w_n)$ is bounded by the diameter of $\partial(t)\partial(h)\overline{\mathtt{n}}(\lambda y.c[y])$.

iii. From the fact that $w_n = w_{(k\cdot\lfloor n \rfloor_k)}$ and the triangular inequality of the partial metric d_A we deduce $d_A(v_n, w_n) = d_A(v_n, w_{(k\cdot\lfloor n \rfloor_k)}) \leq d_A(v_n, v_{(k\cdot\lfloor n \rfloor_k)}) + d_A(v_{(k\cdot\lfloor n \rfloor_k)}, w_{(k\cdot\lfloor n \rfloor_k)}) - d_A(v_{(k\cdot\lfloor n \rfloor_k)}, v_{(k\cdot\lfloor n \rfloor_k)})$

From facts i.-iii. we deduce an explicit bound for $d_A(v_n, w_n)$ in terms of $\partial(t), \partial(f)$ and n: $d_A(v_n, w_n) \leq \delta_A(\partial(t)\partial(h)([k\lfloor n\rfloor_k, n]_{\mathsf{Nat}})\partial(f)) + \delta_A(\partial(t)\partial(h)\overline{\mathtt{n}}(\lambda y.\partial(f)(y-k))) - \delta_A(\partial(t)\partial(h)\overline{\mathtt{n}}\partial(f))$.

We now show how the partial metric semantics can be used to reason about basic approximation techniques from numerical analysis.

▶ **Example 10** (Taylor approximation)**.** We assume that all programs of type $\mathsf{Real} \to \mathsf{Real}$ in $\mathsf{ST\lambda C}(\mathcal{F}_n)$ are differentiable and that for all n, program $t : \mathsf{Real} \to \mathsf{Real}$ and real number r, we can define a term $\mathtt{T}^n(t, r) : \mathsf{Real} \to \mathsf{Real}$ computing the n-th truncated Taylor polynomial of t at r. The distance $d_{\mathsf{Real}\to\mathsf{Real}}(t, \mathtt{T}^n(t, \mathsf{0}))$ is the map associating an interval a with the diameter of the smallest interval containing the image of a under both t and $\mathtt{T}^n(t, \mathsf{0})$. This value will approximately converge to the self-distance of t when a is a small interval of 0, and will tend to diverge when a contains points which are far enough from 0.

For example, if t is the function $t = \lambda x. \sin(x)$, and a is an interval of 0, then using standard analytic reasoning we can compute a bound $d_{\mathsf{Real}\to\mathsf{Real}}(t, \mathtt{T}^n(t, \mathsf{0}))(a) \leq \frac{\delta_{\mathsf{Real}}(a)^{n+1}}{(n+1)!}$, which tends to 0 as the diameter of a tends to 0.

Observe that if, instead, we used the sup-distance $d_{\sup}(t, u) = \sup\{d_{\mathsf{Real}}(tr, ur) \mid r \in \Lambda_{\mathsf{Real}}\}$, then we could not reason as above, since the sup-distance between $\lambda x. \sin(x)$ and its truncated Taylor polynomials is infinite.

▶ **Example 11** (Integral approximation)**.** We now assume that all functions in $\mathcal{F}_n$ are integrable and that we have (see [18]) at our disposal a program $\lambda f x.\mathsf{I}_{[0,x]}(f) : (\mathsf{Real} \to \mathsf{Real}) \to \mathsf{Real} \to \mathsf{Real}$ such that $\mathsf{I}_{[0,r]}(t)$ computes (a precise enough approximation of) the definite integral $\int_0^{|r|} tx\ dx$. In many contexts we might prefer to replace the expensive computation of $\mathsf{I}_{[0,r]}(t)$ by the (more economical but less precise) computation of a finite Riemann sum $\mathsf{R}^n_{[0,r]}(t) = \sum_{i=1}^n (tx_i) \cdot |r|/n$, where $x_i = i \cdot |r|/n$.

Suppose now that, in order to approximate the integral of some computationally expensive program t on $[0, r]$, we replace t by some more efficient program u which, over $[0, r]$, is very close to t. Let $\varepsilon_t(r)$ indicate the distance between the true integral of t over $[0, r]$ and $\mathsf{R}^n_{[0,r]}(t)$, and moreover let $\eta_{t,u}(r)$ be the diameter of $\partial(t)([0, r]) \vee \partial(u)([0, r])$.

Using the metric structure of Real we can then bound the error we incur in by replacing the true integral *of* t with the Riemann sum *of* u. In fact, by standard calculation we can compute the bound $d_{\mathsf{Real}}(\mathsf{R}^n_{[0,r]}(t), \mathsf{R}^n_{[0,r]}(u)) \leq d_{\mathsf{Real}\to\mathsf{Real}}(t, u)([0, r]) \cdot |r| = \eta_{t,u}(r) \cdot |r|$. Then, using the triangular inequality of the standard metric on Real we deduce

$$\begin{aligned} d_{\mathsf{Real}}(\mathsf{I}_{[0,r]}(t), \mathsf{R}^n_{[0,r]}(u)) &\leq d_{\mathsf{Real}}(\mathsf{I}_{[0,r]}(t), \mathsf{R}^n_{[0,r]}(t)) + d_{\mathsf{Real}}(\mathsf{R}_{[0,r]}(t), \mathsf{R}^n_{[0,r]}(u)) \\ &\leq \varepsilon_t(r) + \eta_{t,u}(r) \cdot |r| \end{aligned}$$

Using the partial metric on $\mathsf{Real} \to \mathsf{Real}$, we can also derive a bound expressing how much the error above is *sensitive to changes of* r. First, using standard analytic techniques (under suitable assumptions for t and its derivatives) one can find a program $v : \mathsf{Real} \to \mathsf{Real}$ such that vr computes an upper bound for $\varepsilon_t(r)$. Then, using the triangular inequality of the partial metric on $\mathsf{Real} \to \mathsf{Real}$ we deduce, for all interval a, the following bound:

$$\begin{aligned} & d_{\mathsf{Real}\to\mathsf{Real}}(\lambda x.\mathsf{I}_{[0,x]}(t), \lambda x.\mathsf{R}^n_{[0,x]}(u))(a) \\ \leq\ & d_{\mathsf{Real}\to\mathsf{Real}}(\lambda x.\mathsf{I}_{[0,x]}(t), \lambda x.\mathsf{R}^n_{[0,x]}(t))(a) + d_{\mathsf{Real}\to\mathsf{Real}}(\lambda x.\mathsf{R}_{[0,x]}(t), \lambda x.\mathsf{R}^n_{[0,x]}(u))(a) \\ & \qquad - d_{\mathsf{Real}\to\mathsf{Real}}(\lambda x.\mathsf{R}_{[0,x]}(t), \lambda x.\mathsf{R}^n_{0,x]}(t))(a) \\ \leq\ & d_{\mathsf{Real}\to\mathsf{Real}}(v, v)(a) + \big(d_{\mathsf{Real}\to\mathsf{Real}}(t, u)(a) - d_{\mathsf{Real}\to\mathsf{Real}}(t, t)(a)\big) \cdot \delta_{\mathsf{Real}}(a) \end{aligned}$$

5 Diameter Space Models Over a Cartesian Closed Category

The examples from the last section relied on the fact that our partial metric semantics scales well to extensions of $\mathsf{ST}\lambda\mathsf{C}(\mathcal{F}_n)$ with new base types and new program constructors. In this section we justify this fact in more general terms. In fact, we show that the constructions from Section 3 can be reproduced starting from any model of the simply-typed λ-calculus.

First, we need a suitable notion of model of the simply-typed λ-calculus to start with. Traditionally, one uses *cartesian closed categories*: cartesian categories where, for all objects A, the functor $A \times -$ has a right adjoint (the *exponential* functor). However, since many usual examples are in fact *poset-enriched* categories (e.g. Scott domains and continuous functions, coherent spaces and stable functions), and since any (locally small) category can be poset-enriched by using equality as the ordering, we will consider instead *cartesian closed poset-enriched categories*. To give a counterpart to Proposition 4, we also need a notion of "weak" model of the simply-typed λ-calculus: since poset-enriched categories are a particular case of 2-categories (with a unique 2-arrow from f to g if and only if $f \leq g$), we follow Hilken [24] and consider cartesian categories where, for all objects A, the functor $A \times -$ has a lax right adjoint (the *lax-exponential* functor).

Products and exponentials, when they exist, are necessarily unique up to unique isomorphism: thus, traditionally, a cartesian closed category is defined as a category in which all finite products and exponentials exist, rather than a category *equipped* with products and exponentials (i.e. it is a category with a given *property*, rather than a category with additional *structure*). However, this is not the case for lax-exponentials, so for consistency we will adopt the "structure" picture in both cases. Adapting Hilken's definitions [24] to the simpler case of poset-enriched categories, we obtain:

▶ **Definition 12.** *Let* $(\mathbb{C}, \times, 1)$ *be a cartesian poset-enriched category. An* exponential *(respectively, a* lax-exponential*) on* $\mathbb{C}$ *is the data of a map* exp *from* $\mathrm{Ob}(\mathbb{C} \times \mathbb{C})$ *to* $\mathrm{Ob}(\mathbb{C})$ *and two families of monotone maps* $(\mathrm{ev}_{W,X,Y} : \mathbb{C}(W, \exp(X, Y)) \to \mathbb{C}(W \times X, Y))$ *and* $(\lambda_{W,X,Y} : \mathbb{C}(W \times X, Y) \to \mathbb{C}(W, \exp(X, Y)))$ *such that:*

- $\mathrm{ev}_{W,X,Y}$ *and* $\lambda_{W,X,Y}$ *are natural with respect to* W*,*
- *for all* $g \in \mathbb{C}(W \times X, Y)$*,* $\mathrm{ev}(\lambda(g)) = g$ *(respectively,* $\mathrm{ev}(\lambda(g)) \leq g$*),*
- *for all* $f \in \mathbb{C}(W, \exp(X, Y))$*,* $f = \lambda(\mathrm{ev}(f))$ *(respectively,* $f \leq \lambda(\mathrm{ev}(f))$*).*

One can check that this definition makes exp a functor (respectively, a lax-functor) from $\mathrm{Ob}(\mathbb{C}^{\mathrm{op}} \times \mathbb{C})$ to $\mathrm{Ob}(\mathbb{C})$ (with $\exp(f, g)$ defined as $\lambda(g \circ \mathrm{ev}(\mathrm{id}) \circ (\mathrm{id} \times f))$). In addition, this definition implies that ev and λ are natural, in the sense that $\mathrm{ev}(\exp(\alpha, \beta) \circ f \circ \gamma) = \beta \circ \mathrm{ev}(f) \circ (\gamma \times \alpha)$ and $\exp(\alpha, \beta) \circ \lambda(g) \circ \gamma = \lambda(\beta \circ g \circ (\gamma \times \alpha))$ (respectively, lax-natural [24], in the sense that $\mathrm{ev}(\exp(\alpha, \beta) \circ f \circ \gamma) \leq \beta \circ \mathrm{ev}(f) \circ (\gamma \times \alpha)$ and $\exp(\alpha, \beta) \circ \lambda(g) \circ \gamma \leq \lambda(\beta \circ g \circ (\gamma \times \alpha))$).

For the rest of this section, we fix a cartesian poset-enriched category $(\mathbb{C}, \times, 1)$ (we denote by $\langle -, - \rangle$ the pairing transformation and by π_L and π_R the projections) and an exponential $(\exp, \mathrm{ev}, \lambda)$ on $\mathbb{C}$. The morphisms of this category represent *exact programs*, so they play the role of the terms from Section 3.

▶ **Definition 13.** *A* $\mathbb{C}$-diameter space A *is the data of*

- *an object* $|A|$ *of* $\mathbb{C}$*. The poset* $\mathbb{C}(1, |A|)$ *will be denoted by* Λ_A*;*
- *a set* $[\![A]\!]$ *of downwards-closed subsets of* Λ_A *that is closed under arbitrary intersections. In particular,* $[\![A]\!]$ *is a complete lattice whose meet is given by intersection, and for all* $t \in \Lambda_A$*, there is a least element of* $[\![A]\!]$ *that contains* t*, which will be denoted by* $\bar{t}$*;*
- *a commutative integral quantale* $((\!|A|\!), +, \leq)$*;*

- *a monotone function* $\delta_A : \llbracket A \rrbracket \to \llparenthesis A \rrparenthesis$ *such that*

$$\forall a, b \in \llbracket A \rrbracket \text{ s.t. } a \wedge b \neq \emptyset,\ \ \delta(a \wedge b) + \delta(a \vee b) \leq \delta(a) + \delta(b),$$

and such that for all $t, u \in \Lambda_A$*, if* $\delta_A(\overline{t}) = \delta_A(\overline{t} \vee \overline{u})$*, then* $\overline{t} = \overline{t} \vee \overline{u}$*.*

The role of the condition $a \wedge b \neq \emptyset$ is illustrated by Fig. 5.

▶ **Example 14.** If $\mathbb{C}$ is the category whose objects are the simple types from Section 3 and whose morphisms are the (open) terms modulo β-equivalence, then for all simple types A, $(A, \llbracket A \rrbracket, \llparenthesis A \rrparenthesis, \delta_A)$ defines a $\mathbb{C}$-diameter space.

Following Section 3, for all $\mathbb{C}$-diameter spaces A and B, we define a $\mathbb{C}$-diameter space $A \times B$ such that $|A \times B| = |A| \times |B|$ and a $\mathbb{C}$-diameter space $\exp(A, B)$ such that $|\exp(A, B)| = \exp(|A|, |B|)$:

- $\llbracket A \times B \rrbracket = \{a \times b \mid a \in \llbracket A \rrbracket, b \in \llbracket B \rrbracket\}$, where $a \times b = \{t \in \mathbb{C}(1, |A| \times |B|) \mid \pi_L \circ t \in a \text{ and } \pi_R \circ t \in b\}$,
- $\llparenthesis A \times B \rrparenthesis = \llparenthesis A \rrparenthesis \times \llparenthesis B \rrparenthesis$,
- $\delta_{A \times B}(c) = (\delta_A(\{\pi_L \circ t \mid t \in c\}), \delta_B(\{\pi_R \circ t \mid t \in c\}))$,
- $\llbracket \exp(A, B) \rrbracket = \{\{t \in \mathbb{C}(1, \exp(|A|, |B|)) \mid \forall u \in \Lambda_A,\ \mathrm{ev}(t) \circ u \in I(u)\} \mid I \in \mathrm{Poset}(\Lambda_A, \llbracket B \rrbracket)\}$,
- $\llparenthesis \exp(A, B) \rrparenthesis = \mathrm{Poset}(\llbracket A \rrbracket, \llparenthesis B \rrparenthesis)$,
- $\delta_{\mathsf{exp(A,B)}}(c) = a \mapsto \delta_B\left(\sup\left\{\overline{\mathrm{ev}(v) \circ t} \mid t \in a, v \in c\right\}\right)$.

We need a counterpart to Proposition 4. As explained above, we obtain this by organizing the $\mathbb{C}$-diameter spaces as a cartesian poset-enriched category with a lax-exponential. First, we need to define a notion of morphisms between two $\mathbb{C}$-diameter spaces A and B (which represent *approximate programs*). By analogy with Section 3, these will be monotone functions from $\llbracket A \rrbracket$ to $\llbracket B \rrbracket$; however, in order to actually obtain a cartesian category (which was not an issue in Section 3), we will need to add an extra condition:

▶ **Definition 15.** *We denote by* $\mathrm{Diam}(\mathbb{C})$ *the poset-enriched category defined as follows:*

- *the objects of* $\mathrm{Diam}(\mathbb{C})$ *are the* $\mathbb{C}$*-diameter spaces,*
- *for all* $\mathbb{C}$*-diameter spaces* A *and* B*,* $\mathrm{Diam}(\mathbb{C})(A, B)$ *is the set of all monotone functions* $\varphi : \llbracket A \rrbracket \to \llbracket B \rrbracket$ *such that there exists* $f \in \mathbb{C}(|A|, |B|)$ *such that for all* $t \in \Lambda_A$*,* $f \circ t \in \varphi\left(\overline{t}\right)$ *(ordered by pointwise inclusion).*

One can check that the operation $- \times -$ defined above on $\mathbb{C}$-diameter spaces is a cartesian product in $\mathrm{Diam}(\mathbb{C})$. In addition, one can check that there exists in $\mathrm{Diam}(\mathbb{C})$ a terminal object $1_{\mathrm{Diam}(\mathbb{C})}$ such that $\left|1_{\mathrm{Diam}(\mathbb{C})}\right| = 1_{\mathbb{C}}$. In other words, $\mathrm{Diam}(\mathbb{C})$ is cartesian. Here too, we denote by $\langle -, - \rangle$ the pairing transformation and by π_L and π_R the projections.

Now, following Section 3, we can complete the definition of the lax-exponential: let A, B, C be $\mathbb{C}$-diameter spaces,

- for all $\varphi \in \mathrm{Diam}(\mathbb{C})(A, \exp(B, C))$, we define $\mathrm{ev}_{A,B,C}(\varphi) \in \mathrm{Diam}(\mathbb{C})(A \times B, C)$ by $\mathrm{ev}_{A,B,C}(\varphi)(p) = \sup\left\{\overline{\mathrm{ev}(v) \circ u} \mid v \in \varphi(\pi_L(p)), u \in \pi_R(p)\right\}$,
- for all $\psi \in \mathrm{Diam}(\mathbb{C})(A \times B, C)$, we define $\lambda_{A,B,C}(\psi) \in \mathrm{Diam}(\mathbb{C})(A, \exp(B, C))$ by $\lambda_{A,B,C}(\psi)(a) = \{v \in \Lambda_{\exp(B,C)} \mid \forall u \in \Lambda_B,\ \mathrm{ev}(v) \circ u \in \psi(a \times \overline{u})\}$.

▶ **Proposition 16.** *The triple* $(\exp, \mathrm{ev}, \lambda)$ *is a lax-exponential on* $\mathrm{Diam}(\mathbb{C})$*.*

Proof. Naturality with respect to A is immediate.

Let $p = a \times b \in \llbracket A \times B \rrbracket$. For all $v \in \lambda(\psi)(a)$ and and $u \in b$, by definition $\mathrm{ev}(u) \circ u \in \psi(a \times \overline{u}) \subseteq \psi(p)$. Therefore, $\mathrm{ev}(\lambda(\psi))(p) \subseteq p$.

Let $a \in \llbracket A \rrbracket$ and $v \in \varphi(a)$. For all $u \in \Lambda_B$, by definition, $\mathrm{ev}(v) \circ u \in \lambda(\varphi)(a \times \overline{u})$, so $v \in \lambda(\mathrm{ev}(\varphi))(a)$. ◀

As in Section 3, we can find a kind of weak embedding from $\mathbb{C}$ to $\mathrm{Diam}(\mathbb{C})$. Namely, for all $\mathbb{C}$-diameter spaces A and B, we define a monotone map $\partial : \mathbb{C}(|A|, |B|) \to \mathrm{Diam}(\mathbb{C})(A, B)$ by $\partial(f)(a) = \sup\{\overline{f \circ t} \mid t \in a\}$. The following compatibility result is immediate and offers a counterpart to Remark 6:

▶ **Proposition 17.** *For all $\mathbb{C}$-diameter spaces A, B, C, all $f \in \mathbb{C}(|A|, |B|)$ and all $g \in \mathbb{C}(|B|, |C|)$, $\partial(g \circ f) \leq \partial(g) \circ \partial(f)$. In addition, $\partial(\mathrm{id}_{|A|}) = \mathrm{id}_A$.*

One way to reformulate this result is that ∂ induces an oplax-functor from the category with the same objects as $\mathrm{Diam}(\mathbb{C})$ and the same morphisms as $\mathbb{C}$, to $\mathrm{Diam}(\mathbb{C})$.

One can check that ∂ preserves products, in the sense that $\partial(\langle f, g\rangle) = \langle \partial(f), \partial(g)\rangle$, $\partial(\pi_L) = \pi_L$ and $\partial(\pi_R) = \pi_R$. In addition ∂ is weakly compatible with the exponential, which corresponds to Remark 5:

▶ **Proposition 18.** *Let A, B, C be $\mathbb{C}$-diameter spaces,*

- *for all $f \in \mathbb{C}(|A|, \exp(|B|, |C|))$, $\partial(\mathrm{ev}(f)) \leq \mathrm{ev}(\partial(f))$,*
- *for all $g \in \mathbb{C}(|A| \times |B|, |C|)$, $\partial(\lambda(g)) \leq \lambda(\partial(g))$.*

Finally, following Section 3, for all $\mathbb{C}$-diameter spaces A and all $t, u \in \Lambda_A$, we write $t \approx_A u$ if $\overline{t} = \overline{u}$. In addition, we define a function $d_A : \Lambda_A \times \Lambda_A \to (\!|A|\!)$ by $d_A(t, u) = \delta_A(\overline{t} \vee \overline{u})$. Then the same arguments as in Corollary 8 show that:

▶ **Proposition 19.** *For all $\mathbb{C}$-diameter spaces A,* $(\Lambda_A / \approx_A, (\!|A|\!), d_A)$ is a generalized partial metric space.

One can check that what is described in Section 3 is indeed an instance of this construction. Here are a couple more examples:

▶ **Example 20.** We can take $\mathbb{C} = \mathrm{Set}$ (with the morphisms ordered by equality): $\mathrm{Diam}(\mathrm{Set})$ contains an object $\mathsf{Real}_{\mathrm{Set}}$ that represents the real numbers with their standard metric and the compact intervals (plus $\emptyset$ and $\mathbb{R}$) as approximate values, namely $|\mathsf{Real}_{\mathrm{Set}}| = \mathbb{R}$, $[\![\mathsf{Real}_{\mathrm{Set}}]\!] = \{\text{the compact intervals}, \emptyset, \mathbb{R}\}$, $(\!|\mathsf{Real}_{\mathrm{Set}}|\!) = [0, \infty]$ and $\delta_{\mathsf{Real}_{\mathrm{Set}}}(I) = \mathrm{length}(I)$.

In this case, $|\exp(\mathsf{Real}_{\mathrm{Set}}, \mathsf{Real}_{\mathrm{Set}})|$ is the set of all functions from $\mathbb{R}$ to $\mathbb{R}$, so $d_{\mathsf{Real}_{\mathrm{Set}}}$ defines a partial metric on all such functions.

▶ **Example 21.** We can take $\mathbb{C} = \mathrm{Eff}$, the effective topos [27]: Eff contains an object $\mathbb{R}_{\mathrm{Eff}}$ of recursive reals, and we can define an object $\mathsf{Real}_{\mathrm{Eff}}$ in $\mathrm{Diam}(\mathrm{Eff})$ by $|\mathsf{Real}_{\mathrm{Eff}}| = \mathbb{R}_{\mathrm{Eff}}$, $[\![\mathsf{Real}_{\mathrm{Eff}}]\!] = \{I \cap \mathbb{R}_{\mathrm{Eff}} \mid I \in [\![\mathsf{Real}_{\mathrm{Set}}]\!]\}$, $(\!|\mathsf{Real}_{\mathrm{Eff}}|\!) = [0, \infty]$ and $\delta_{\mathsf{Real}_{\mathrm{Eff}}}(I) = \mathrm{length}(I)$.

In this case, $|\exp(\mathsf{Real}_{\mathrm{Eff}}, \mathsf{Real}_{\mathrm{Eff}})|$ is the set of all recursive functions from $\mathsf{Real}_{\mathrm{Eff}}$ to $\mathsf{Real}_{\mathrm{Eff}}$, so $d_{\mathsf{Real}_{\mathrm{Eff}}}$ defines a partial metric on all such functions.

▶ **Example 22.** We can take $\mathbb{C} = \mathrm{Scott}$, the poset-enriched category of Scott domains and continuous functions. It contains an object representing the reals: $\mathbb{R}_{\mathrm{Scott}} = (\mathbb{R} \cup \{\bot\}, \sqsubseteq)$, with $r \sqsubseteq s$ *iff* $r = s$ or $r = \bot$. Again, we can define in $\mathrm{Diam}(\mathrm{Scott})$ an object $\mathsf{Real}_{\mathrm{Scott}}$ that represents the real numbers with their standard metric, and this defines a partial metric on $|\exp(\mathsf{Real}_{\mathrm{Scott}}, \mathsf{Real}_{\mathrm{Scott}})|$, the set of all Scott continuous functions from $\mathbb{R}_{\mathrm{Scott}}$ to $\mathbb{R}_{\mathrm{Scott}}$, which are essentially the partial functions from $\mathbb{R}$ to $\mathbb{R}$.

6 Conclusions

Related Work. As stated in the introduction, differential logical relations [32] are a primary source of inspiration for our approach. A related, but more syntactic approach to approximate program transformations is that of Westbrook and Chauduri [44], who use a System F-based type system with a type of real numbers and an explicit distinction between exact and approximate programs. Most examples of contextual reasoning from [44] can be reformulated in our framework (as the case of loop perforation discussed in Section 4).

The literature on program pseudo-metrics is vast. A major distinction can be made between those approaches in which metrics account for *extensional* aspects of programs (like ours), and approaches in which metrics are used to characterize more *intensional* aspects. To the first family belong all metric models developed for reasoning about differential privacy [35, 3, 7], probabilistic computation [13, 14] and co-inductive models [16, 43, 11, 42]. To the second class belong approaches like [19] which recovers the Scott model of PCF through a ultrametric semantics, and most models based on partial metric spaces [9, 33], which rely on a correspondence between continuous Scott domains and the T_0 topology of partial metrics.

From a more mathematical viewpoint, [12] discusses a characterization of exponentiable GPMS, showing that no such category can both be cartesian closed and contain the standard metric on $\mathbb{R}$. This result seems to add further evidence of the necessity of considering metrics over varying quantales in order to model higher-order languages. Finally, the elegant categorical approach to GPMS based on *quantaloid-enriched categories* from [26] seems to provide the relevant structure to develop explicit typing rules for our approximate programs.

Future Work. The approach we presented lends itself to further extensions and generalizations. First, we would like to investigate the interpretation of more type constructions than those of $\mathsf{ST}\lambda\mathsf{C}(\mathcal{F}_n)$ (e.g. coproducts, recursive types, effects). Moreover, we would like to explore the possibility of exploiting the structure of the category $\mathrm{Diam}(\mathbb{C})$ to construct new and more refined notions of approximations. For example (we work in $\mathrm{Diam}(\mathrm{Set})$ for simplicity), starting from the “standard” set of approximate values $\mathcal{I}$ on $\mathbb{R}^{X\times X}$ (with elements of $\mathcal{I}$ being families of compact intervals $U_{x,x'} \subseteq \mathbb{R}$ indexed by elements of X and X'), one can define a new family $\Delta^*\mathcal{I}$ of approximate values for $\mathbb{R}^X$ by “pulling back” the exact map $\Delta : \mathbb{R}^X \to \mathbb{R}^{X\times X}$ defined by $\Delta f(x,x') = f(x') - f(x)$, i.e. letting $\Delta^*\mathcal{I} = \{\Delta^{-1}(a) \mid a \in \mathcal{I}\}$. The new approximate values then correspond to sets of functions $f \in \mathbb{R}^X$ with a controlled variation, that is, such that $f(x') - f(x)$ is bounded by some family of intervals $U_{x,x'} \in \mathcal{I}$.

Another interesting research direction concerns probabilistic extensions of $\mathsf{ST}\lambda\mathsf{C}(\mathcal{F}_n)$. Probabilistic metrics [15, 30, 13, 14] have been the object of much research in recent years, due to the relevance of metric reasoning in some areas of computer science in which probabilistic computation plays a key role (e.g. in cryptography [22] and machine learning [31]). A convenient starting point seems to be the recent generalization of probabilistic (generalized) metric spaces to the partial metric case [23].

References

1 S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum. *Handbook of Logic in Computer Science: Volume 2. Background: Computational Structures*. Handbook of Logic in Computer Science. Clarendon Press, 1992. URL: `https://books.google.fr/books?id=zqkXKMNXVi0C`.

2 Mario Alvarez-Picallo and C.-H. Luke Ong. Change actions: models of generalised differentiation. In *FOSSACS 2019*, volume 11425 of *LNCS*, pages 45–61, 2019.

3 Mário S. Alvim, Miguel E. Andrés, Konstantinos Chatzikokolakis, Pierpaolo Degano, and Catuscia Palamidessi. Differential privacy: On the trade-off between utility and information leakage. In *Proceedings of the 8th International Conference on Formal Aspects of Security and Trust*, FAST–11, pages 39–54, Berlin, Heidelberg, 2011. Springer-Verlag. doi:10.1007/978-3-642-29420-4_3.

4 A. Arnold and M. Nivat. Metric interpretations of infinite trees and semantics of non deterministic recursive programs. *Theoretical Computer Science*, 11(2):181–205, 1980. doi:10.1016/0304-3975(80)90045-6.

5 Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, Shin-ya Katsumata, and Ikram Cherigui. A semantic account of metric preservation. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages - POPL 2017*. ACM Press, 2017. doi:10.1145/3009837.3009890.

6 Christel Baier and Mila E. Majster-Cederbaum. Denotational semantics in the cpo and metric approach. *Theoretical Computer Science*, 135(2):171–220, 1994. doi:10.1016/0304-3975(94)00046-8.

7 Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '12*. ACM Press, 2012. doi:10.1145/2103656.2103670.

8 Michael Bukatin, Ralph Kopperman, Steve Matthews, and Homeira Pajoohesh. Partial metric spaces. *American Mathematical Monthly - AMER MATH MON*, 116:708–718, October 2009. doi:10.4169/193009709X460831.

9 Michael A. Bukatin and Joshua S. Scott. Towards computing distances between programs via scott domains. In Sergei Adian and Anil Nerode, editors, *Logical Foundations of Computer Science*, pages 33–43, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

10 Y. Cai, P.G. Giarrusso, T. Rendel, and K. Ostermann. A theory of changes for higher-order languages: incrementalizing λ-calculi by static differentiation. *ACM SIGPLAN Not.*, 49:145–155, 2014.

11 Konstantinos Chatzikokolakis, Daniel Gebler, Catuscia Palamidessi, and Lili Xu. Generalized bisimulation metrics. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 – Concurrency Theory*, pages 32–46, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

12 Maria Manuel Clementino, Dirk Hofmann, and Isar Stubbe. Exponentiable functors between quantaloid-enriched categories. *Applied Categorical Structures*, 17(1):91–101, 2009. doi:10.1007/s10485-007-9104-5.

13 Raphaëlle Crubillé and Ugo Dal Lago. Metric reasoning about λ-terms: The affine case. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, LICS '15, page 633?644, USA, 2015. IEEE Computer Society. doi:10.1109/LICS.2015.64.

14 Raphaëlle Crubillé and Ugo Dal Lago. Metric reasoning about λ-terms: The general case. In Hongseok Yang, editor, *Programming Languages and Systems*, pages 341–367, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

15 J. Desharnais, R. Jagadeesan, V. Gupta, and P. Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 413–422, July 2002. doi:10.1109/LICS.2002.1029849.

16 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004. doi:10.1016/j.tcs.2003.09.013.

17 Pietro Di Gianantonio and Abbas Edalat. A language for differentiable functions. In Frank Pfenning, editor, *Foundations of Software Science and Computation Structures*, pages 337–352, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

18 Abbas Edalat and Martín Hötzel Escardó. Integration in real pcf. *Information and Computation*, 160(1):128–166, 2000. doi:10.1006/inco.1999.2844.

19 Martín Hötzen Escardó. A metric model of PCF. In *Workshop on Realizability Semantics and Applications*, 1999.

20 Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In *Proceedings of the 40th annual ACM*

SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '13. ACM Press, 2013. doi:10.1145/2429069.2429113.

21 Francesco Gavazzo. Quantitative behavioural reasoning for higher-order effectful programs: Applicative distances. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 452?461, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3209108.3209149.

22 Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. doi:10.1016/0022-0000(84)90070-9.

23 Jialiang He, Hongliang Lai, and Lili Shen. Towards probabilistic partial metric spaces: Diagonals between distance distributions. *Fuzzy Sets and Systems*, 370:99–119, 2019. Theme: Topology and Metric Spaces. doi:10.1016/j.fss.2018.07.011.

24 Barnaby P. Hilken. Towards a proof theory of rewriting: the simply typed 2λ-calculus. *Theoretical Computer Science*, 170(1):407–444, 1996. doi:10.1016/S0304-3975(96)80713-4.

25 Dirk Hofmann, Gavin J Seal, and W Tholen. *Monoidal Topology: a Categorical Approach to Order, Metric and Topology*. Cambridge University Press, New York, 2014.

26 Dirk Hofmann and Isar Stubbe. Topology from enrichment: the curious case of partial metrics. *Cahiers de Topologie et Géométrie DIfférentielle Catégorique, LIX*, 4:307–353, 2018.

27 J.M.E. Hyland. The effective topos. In A.S. Troelstra and D. [van Dalen], editors, *The L. E. J. Brouwer Centenary Symposium*, volume 110 of *Studies in Logic and the Foundations of Mathematics*, pages 165–216. Elsevier, 1982. doi:10.1016/S0049-237X(09)70129-6.

28 Gunther Jäger and T. M. G. Ahsanullah. Characterization of quantale-valued metric spaces and quantale-valued partial metric spaces by convergence. *Applied General Topology*, 19(1):129–144, 2018.

29 Ralph Kopperman, Steve Matthews, and Homeira Pajoohesh. Partial metrizability in value quantales. *Applied General Topology*, 5(1):115–127, 2004.

30 Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981. doi:10.1016/0022-0000(81)90036-2.

31 Andreas Krause, Brendan McMahan, Carlos Guestrin, and Anupam Gupta. Robust submodular observation selection. *Journal of Machine Learning Research (JMLR)*, 9:2761–2801, December 2008.

32 Ugo Dal Lago, Francesco Gavazzo, and Akira Yoshimizu. Differential logical relations, part I: the simply-typed case. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 111:1–111:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.111.

33 S. G. Matthews. Partial metric topology. *Annals of the New York Academy of Sciences*, 728(1):183–197, 1994. doi:10.1111/j.1749-6632.1994.tb44144.x.

34 Sparsh Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4), 2016.

35 Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. *SIGPLAN Not.*, 45(9):157–168, September 2010. doi:10.1145/1932681.1863568.

36 Bessem Samet, Calogero Vetro, and Francesca Vetro. From metric spaces to partial metric spaces. *Fixed Point Theory and Applications*, 2013(1):5, 2013. doi:10.1186/1687-1812-2013-5.

37 M. P. Schellekens. The correspondence between partial metrics and semivaluations. *Theoretical Computer Science*, 315(1):135–149, 2004.

38 Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE ?11, pages 124–134, New York, NY, USA, 2011. Association for Computing Machinery.

39 D. A. Simovici. On submodular and supermodular functions on lattices and related structures. In *2014 IEEE 44th International Symposium on Multiple-Valued Logic*, pages 202–207, May 2014. doi:10.1109/ISMVL.2014.43.

40 Paul Taylor. A lambda calculus for real analysis. *Journal of Logic and Analysis*, 2(5):1–115, 2010.

41 Franck van Breugel. An introduction to metric semantics: operational and denotational models for programming and specification languages. *Theoretical Computer Science*, 258(1):1–98, 2001. doi:10.1016/S0304-3975(00)00403-5.

42 Franck van Breugel and James Worrell. Towards quantitative verification of probabilistic transition systems. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming*, pages 421–432, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

43 Franck van Breugel and James Worrell. A behavioural pseudometric for probabilistic transition systems. *Theoretical Computer Science*, 331(1):115–142, 2005. Automata, Languages and Programming. doi:10.1016/j.tcs.2004.09.035.

44 Edwin Westbrook and Swarat Chaudhuri. A semantics for approximate program transformations, 2013. URL: https://arxiv.org/abs/1304.5531.

A Deep Quantitative Type System

Giulio Guerrieri
University of Bath, Department of Computer Science, UK

Willem B. Heijltjes
University of Bath, Department of Computer Science, UK
http://willem.heijltj.es/

Joseph W.N. Paulus
Rijksuniversiteit Groningen, The Netherlands

Abstract

We investigate intersection types and resource lambda-calculus in deep-inference proof theory. We give a unified type system that is parametric in various aspects: it encompasses resource calculi, intersection-typed lambda-calculus, and simply-typed lambda-calculus; it accommodates both idempotence and non-idempotence; it characterizes strong and weak normalization; and it does so while allowing a range of algebraic laws to determine reduction behaviour, for various quantitative effects. We give a parametric resource calculus with explicit sharing, the "collection calculus", as a Curry–Howard interpretation of the type system, that embodies these computational properties.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Lambda calculus

Keywords and phrases Lambda-calculus, Deep inference, Intersection types, Resource calculus

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.24

Funding This work was supported by EPSRC Project EP/R029121/1 *Typed lambda-calculi with sharing and unsharing.*

Acknowledgements We would like to thank Ugo Dal Lago, Delia Kesner, Luc Pelissier, Nicolas Wu, and the anonymous referees for their constructive engagement with our work.

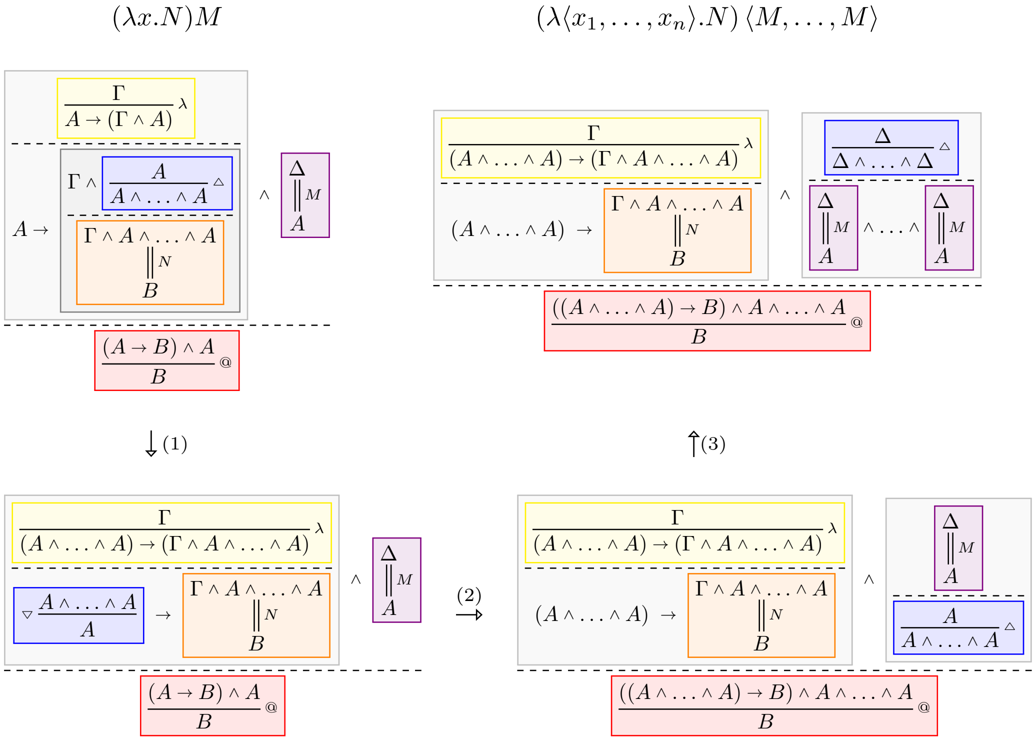
1 Introduction

Of the various qualitative and quantitative approaches to λ-calculus, which include intersection types [15, 16, 23], resource calculi [9, 30, 21], and relational models [31, 33], many are known to be related, often in deep and interesting ways. We are curious if there is a common foundation, a question that we approach through deep-inference proof theory. Here, we give a unified, structural perspective on *intersection types* and *resource calculi*, in the form of a deep quantitative proof system. It is both a simple type system for a resource calculus, the *collection calculus* that we introduce here, and an intersection type system for an explicit-substitution calculus, the *structural λ-calculus* [3, 4]. In both cases, it can be parameterized in various algebraic laws to obtain different quantitative effects.

The computational side of deep inference

Deep inference, as a family of proof formalisms, has remarkable properties: quasi-polynomial proof complexity and normalization for propositional classical logic [28, 12], non-elementary proof compression for first-order classical logic [5], and the ability to express logics for which no sequent calculus can exist [39], among others. It is a natural question if such striking features can be put to computational use. In previous work in this direction, the second author and co-authors derived two *atomic* λ-calculi, which characterize different versions of *full laziness*, from the duplication properties of intuitionistic deep inference [26, 37].

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 24; pp. 24:1–24:24
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



$(\lambda x.N)M$
$(\lambda\langle x_1,\ldots,x_n\rangle.N)\,\langle M,\ldots,M\rangle$
(1)
(2)
(3)

Figure 1 *Deriving resource calculus by proof transformations.* The derivation top left is for the lambda-term $(\lambda x.N)M$. The blue contraction rule $(\triangle)$ passes through the yellow abstraction rule (λ) in step (1), ending up as the inverted rule $(\triangledown)$ to reflect that $(\to)$ reverses "polarity" on the left; it then passes through the application rule (@) in step (2); and duplicates the argument derivation M in step (3). The resulting derivation, top right, is for an interpretation of the original term as a resource term $(\lambda\langle x_1,\ldots,x_n\rangle.N)\,\langle M,\ldots,M\rangle$, where the variables x_i represent the occurrences of x. Categorically, (λ) and (@) are the η and ϵ transformations of the adjunction between $(\to)$ and $(\wedge)$, while $(\triangle)$ is the diagonal map for $(\wedge)$; the steps (1)–(3) then reflect the (di)naturality of these maps, and the inversion of $(\triangle)$ to $(\triangledown)$ reflects the contravariance of $(\to)$ in its first argument.

In this paper, we investigate *intersection types* [15] and *resource lambda-calculi* [9] from the perspective of deep inference. We will work in the formalism *open deduction* [25]; see Section 3 for an introduction. We start by observing that in lambda-calculus and natural deduction, duplication and beta-reduction are intimately related: in an abstraction $\lambda x.N$, the bound occurrences of x in N represent a potential duplication, which can only be effected by a beta-step on $(\lambda x.N)M$. Systems like sequent calculi, proof nets, and explicit-substitution calculi may separate beta-reduction and duplication by an explicit *contraction* rule. Deep inference goes one step further: contraction rules may *pass through* other proof rules. We illustrate this in Figure 1. For a simply-typed open-deduction proof, one may carry out all latent duplication by pushing the contractions through the proof in the way of the example, until they disappear at the top or bottom of the proof. Doing so transforms a simple type derivation for a lambda-term into a (non-idempotent) intersection-type derivation, or equivalently into a simple type derivation for a corresponding resource term. The result is familiar from resource calculi, which may unfold the redex $(\lambda x.N)M$ to $(\lambda\langle x_1,\ldots,x_n\rangle.N)\langle M,\ldots,M\rangle$ where the variables x_i represent the n occurrences of x in N. Crucially, in open deduction this transformation applies not only to redexes, but to individual abstractions and applications.

In Figure 1, the conjunction ($\wedge$) has two distinct rôles: its standard rôle in the application rule, in typing NM, plus that of creating a *collection* of derivations, in typing $\langle M, \ldots, M\rangle$. We separate both rôles by introducing an intersection type operator ($+$) for collections, leaving the conjunction in its traditional rôle. We give the operator ($+$), and the rules for it to interact with ($\wedge$), below: (1) as is characteristic of connectives in open deduction, ($+$) applies to *derivations* as well as formulas, giving a derivation from $A+C$ to $B+D$; (2) the *contraction* rule ($\triangle$) is modified to transform a conjunction into an intersection; and (3) conjunction and intersection are interchanged by a (non-invertible) *medial* rule (m).

$$(1)\quad \begin{array}{c} A \\ \| \\ B \end{array} + \begin{array}{c} C \\ \| \\ D \end{array} \qquad\qquad (2)\quad \dfrac{A+B}{A\wedge B}\,\triangle \qquad\qquad (3)\quad \dfrac{(A+B)\wedge(C+D)}{(A\wedge C)+(B\wedge D)}\,\mathsf{m}$$

Our construction makes essential use of the characteristic properties of open deduction. Firstly, operators apply to *derivations* as well as *formulas*, as in (1) above. Via the Curry–Howard correspondence this gives us a natural, simultaneous treatment of collections of *terms* and collections of *types*, giving a tight correspondence between resource terms and their type derivations. Secondly, *medial*-style rules [11, 38, 6] are unique to deep-inference systems, and are at the root of many of the contributions of the theory, including the complexity results for classical logic mentioned above, and both atomic lambda-calculi.

The most salient feature of our approach is that the calculus and the type system can be *parameterized* in various algebraic laws, which captures for instance the familiar distinction between *idempotent* and *non-idempotent* intersection types. This is made possible by our structural approach: once the above constructions (1–3) are available, the proof system is in principle agnostic about the further properties of collections.

Our technical exposition starts with a system of simple types in open deduction, in Section 3, for the *Structural λ-Calculus* $\lambda\mathsf{j}$ of Accattoli and Kesner [3, 4], which we recall in Section 2 and here abbreviate SC. The work of Accattoli and Kesner derives the SC, and the related *Linear Substitution Calculus* (LSC, [1]), from an extensive search for good reduction properties in explicit-substitution calculi, inspired by linear logic. Here, we observe that the SC also arises as a natural Curry–Howard-style interpretation of intuitionistic open deduction. We view this as further support for our proof-theory based approach. A version of the SC with linear use of variables was the basis for both atomic λ-calculi [26, 37].

A precursor to the present work is the workshop paper [27]. Proofs are in the Appendix.

Related work. Intersection types, in their idempotent variant, have been studied to characterize several kinds of normalization [15, 16, 36]. The non-idempotent variant introduced in [23] is strictly related to linear logic [18, 19] and induces a well-known denotational model of the λ-calculus and linear logic: relational semantics [13, 34]. The literature about intersection types is huge, let us mention [14] for a survey and [2] for recent developments.

Resource-sensitive calculi [9, 30] can be seen as a “dynamic” counterpart of non-idempotent intersection types, often inspired by linear logic, see for instance [21, 22, 33].

Approaches to resource calculi and intersection types from a proof-theoretic perspective are uncommon; for the former, since qualitative and quantitative properties are already captured through the *term calculus*, and for the latter since the restriction that intersection types can only be formed for proofs *of the same term* is a fundamental departure from traditional logical systems; exceptions to the latter are [35, 20].

A different unified perspective, via category theory, is given in [32]; the conceptual difference is that their approach is *extensional* (characterizing qualitative systems through their properties) where ours is *intensional* (we give an underlying syntactic structure).

2 The structural λ-calculus

Our point of departure is the *structural λ-calculus* (SC) of Accattoli and Kesner [3, 4], an explicit-substitution λ-calculus where closures are evaluated by *decomposition* and *linear substitution*. This puts the dynamic behaviour of the calculus away from implicit substitution, and closer to *graph reduction*; we prefer to call it an *explicit-sharing* calculus instead.

As an interpretation of deep inference, other calculi with explicit sharing would be equally suitable, such as $\lambda\mathsf{lxr}$ of Kesner and Lengrand [29]; we choose the SC for its concise notation.

▶ **Definition 1.** *The* terms r, s, t *of the SC are defined by the grammar*

$$r, s, t \;::=\; x \;\mid\; ts \;\mid\; \lambda x.t \;\mid\; t[x \leftarrow s]$$

with from left to right: a variable*; an* application*; an* abstraction*, which binds x in t; and a* closure*, which binds x in t.*

We call $[x \leftarrow s]$ a *sharing*, abbreviated to $[\phi]$, and write $[\Phi]$ for a sequence of sharings $[x_1 \leftarrow s_1] \ldots [x_n \leftarrow s_n]$, or $t[\Phi]$ when applied as closures to a term t. We write $\{t/x\}$ for the (capture-avoiding) substitution of t for x, and $|t|_x$ for the number of free occurrences of x in t. The set of free variables of a term t is denoted by $\mathsf{fv}(t)$.

▶ **Definition 2.** *The* reduction rules *of the SC are the contextual closure of the rules below.*

$$\begin{array}{rcllr}
(\lambda x.t)[\Phi]s & \to_{\mathsf{b}} & t[x \leftarrow s][\Phi] & & \text{(beta)} \\
t\{x/y\}[x \leftarrow s] & \to_{\mathsf{c}} & t[x \leftarrow s][y \leftarrow s] & |t|_x, |t|_y \geq 1 & \text{(copy)} \\
t[x \leftarrow s] & \to_{\mathsf{e}} & t\{s/x\} & |t|_x = 1 & \text{(evaluate)} \\
t[x \leftarrow s] & \to_{\mathsf{d}} & t & |t|_x = 0 & \text{(delete)}
\end{array}$$

We set $\to_{\neg\mathsf{b}} \;=\; \to_{\mathsf{c}} \cup \to_{\mathsf{d}} \cup \to_{\mathsf{e}}$ *and* $\to_{\mathsf{sc}} \;=\; \to_{\mathsf{b}} \cup \to_{\neg\mathsf{b}}$.

The *beta*-rule includes the closures $[\Phi]$ so that these do not block the redex: it acts *at a distance*. This mimicks graph reduction and obviates the need to permute closures. In the *copy* rule, the notation $t\{x/y\}$ is used to separate the occurrences of x into two (non-empty) classes: those that occur as x in t and those that occur as y in t. The sharing $[x \leftarrow s]$ can then be duplicated and split among them. This is used to isolate a variable with one occurrence, to which the *evaluate* rule then applies, whose substitution $\{s/x\}$ is *linear.*

For a rewrite relation $\to$, we write $\twoheadrightarrow$ for its reflexive-transitive closure, and $\twoheadrightarrow\!\!\!|$ for reduction to normal form. The λ-calculus embeds into the SC without using a translation, as λ-terms are the SC-terms that do not have closures. The *unfolding* $t^\bullet$ of a term t, defined below, evaluates all closures by substitutions, which interprets SC-terms as λ-terms.

$$x^\bullet = x \qquad (\lambda x.t)^\bullet = \lambda x.t^\bullet \qquad (ts)^\bullet = t^\bullet s^\bullet \qquad (t[x \leftarrow s])^\bullet = t^\bullet\{s^\bullet/x\}.$$

▶ **Proposition 3** (Simulations). *Let t be a SC-term and s be a λ-term.*

1. From SC to λ-calculus*: If $t \to_{\mathsf{b}} t'$ then $t^\bullet \twoheadrightarrow_\beta t'^\bullet$; if $t \to_{\neg\mathsf{b}} t'$ then $t^\bullet = t'^\bullet$.*
2. From λ-calculus to SC*: If $s \to_\beta s'$ then $s \to_{\mathsf{b}} \twoheadrightarrow\!\!\!|_{\neg\mathsf{b}}\, s'$.*

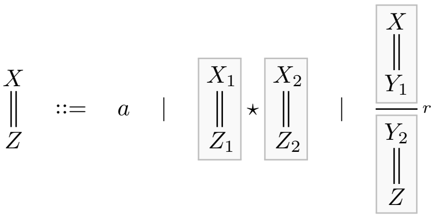
▶ **Proposition 4** (Collated results from [4])**.** *The SC has the following key properties.*

1. *The normal forms of $\to_{\neg\mathsf{b}}$ are exactly the λ-terms.*
2. *The normal forms of $\to_{\mathsf{sc}}$ are exactly the β-normal λ-terms.*
3. *For any SC-term t, one has $t \twoheadrightarrow\!\!\!|_{\neg\mathsf{b}}\, t^\bullet$; in particular, $t = t^\bullet$ for any λ-term.*
4. *The relations $\to_{\mathsf{b}}$, $\to_{\neg\mathsf{b}}$, and $\to_{\mathsf{sc}}$ are confluent; $\to_{\mathsf{b}}$ and $\to_{\neg\mathsf{b}}$ are strongly normalizing.*
5. *Preservation of strong normalization: if a λ-term t has an infinite $\to_{\mathsf{sc}}$-reduction, then it has an infinite $\to_\beta$-reduction.*

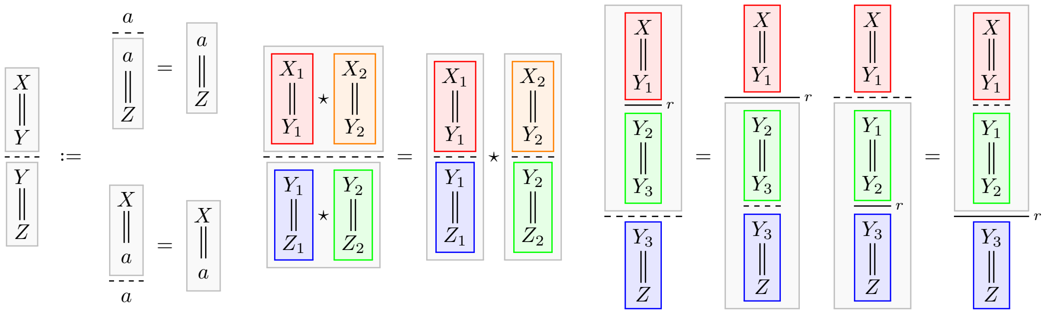
3 A deep type system

Open deduction is a dialect of deep-inference proof theory, introduced by Guglielmi, Gundersen, and Parigot [25], where proofs are constructed in two directions: *horizontally* by *connectives*, and *vertically* by *rules*. We give a brief formal introduction.

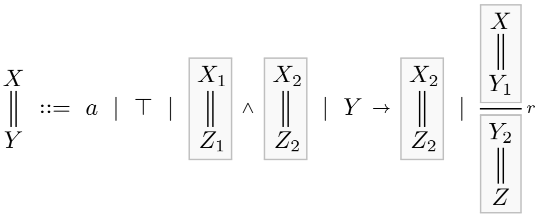
A *derivation* from a *premise* formula X to a *conclusion* formula Z is constructed inductively as below, with from left to right: a propositional atom a, where $X = Z = a$; *horizontal construction* with a connective $\star$, where $X = X_1 \star X_2$ and $Z = Z_1 \star Z_2$; and *vertical construction* with an inference rule r from Y_1 to Y_2. Boxes serve as parentheses (since derivations extend in two dimensions) and may be omitted.



Derivations are considered up to associativity of vertical construction. One may consider *formulas* as derivations that omit vertical construction. The binary $\star$ may be generalized to 0-ary, unary, and n-ary operators, and it may have *negative* arguments where a derivation becomes *inverted*, exchanging premise and conclusion, such as to the left of an implication – though we will avoid the need for these. *Composition* of a derivation from X to Y and one from Y to Z, depicted by a dashed line, is a defined operation:



We specialize the above to a proof system for conjunction-implication intuitionistic logic, similar to that of Brünnler and McKinley [10], through the grammar and inference rules below. Note the inclusion of the unit $\top$ as a 0-ary operator, and the restriction of the left subderivation of $(\to)$ to a formula, to avoid introducing inverted derivations. The rules are: *abstraction* (λ), *application* $(@)$, and n-ary *contraction* $(\triangle)$ on the left and the invertible rules for *associativity*, *symmetry*, and *unitality* of conjunction on the right. A 0-ary contraction, with conclusion $\top$, is a *weakening*. We will leave the invertible rules implicit in derivations, and consider conjunction modulo associativity and unitality.



$$\frac{X}{Y \to (X \wedge Y)}\,\lambda \qquad \frac{X \wedge (Y \wedge Z)}{(X \wedge Y) \wedge Z}\,=$$

$$\frac{(X \to Y) \wedge X}{Y}\,@ \qquad \frac{X \wedge Y}{Y \wedge X}\,=$$

$$\frac{X}{X \wedge \cdots \wedge X}\,\triangle \qquad \frac{X \wedge \top}{X}\,=$$

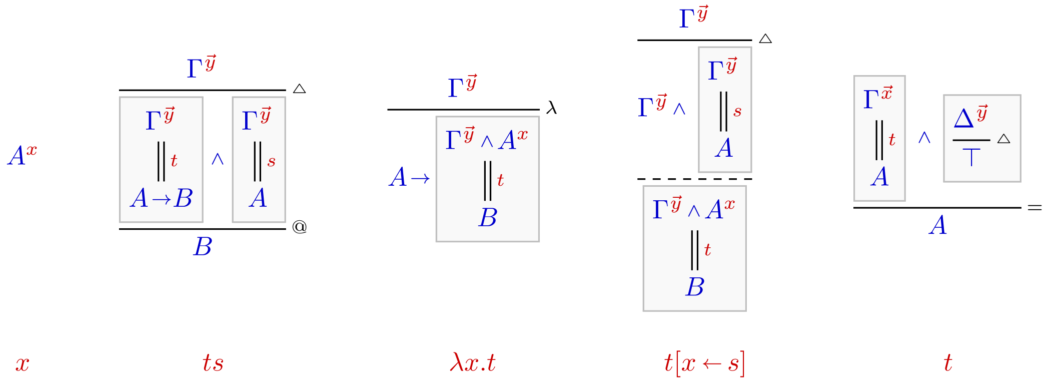
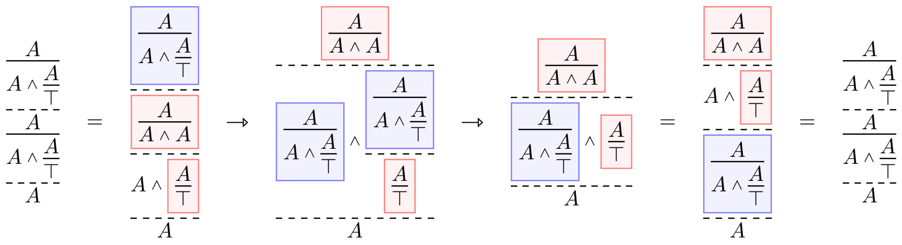


Figure 2 *An open-deduction system of simple types for the structural λ-calculus.* In ts, both t and s are given the same context $\Gamma^{\vec{y}}$ with $\vec{y} = \mathsf{fv}(t) \cup \mathsf{fv}(s)$ by applying the rules △ and = as in the rightmost construction, where $\vec{x} \cap \vec{y} = \varnothing$; and similarly for $t[x \leftarrow s]$.

Typing the structural λ-calculus

We give an open-deduction system of simple types for the SC. Not all derivations correspond to a term: the calculus picks out a subset of derivations, imposes certain equivalences, and guides reduction. The latter is essential, since naïve reduction in the proof system creates cycles of contractions duplicating each other: this example is from [10] – see there for detail.

▶ **Example 5** ([10])**.** The natural reductions for a contraction or weakening rule are to duplicate respectively to delete the derivation above it (see also Figure 3). Applied naïvely, this reduction is non-terminating by the example below (all explicit rules are contractions).



The type system is in Figure 2. A term t is typed by a derivation from Γ to A, which we indicate as below left. The *structured types* A and Γ are respectively a *basic type* and a *context type*, generated by the respective grammars below. A context type $\Gamma = A_1 \wedge \cdots \wedge A_n$ in the premise of the derivation for t types a vector of *context variables* $\vec{x} = x_1, \ldots, x_n$, which include the free variables of t, but may be expanded to include variables not occurring in t via the rightmost derivation in Figure 2. We make context variables explicit as $\Gamma^{\vec{x}} = A_1^{x_1} \wedge \cdots \wedge A_n^{x_n}$.

A derivation for t: $\begin{array}{c}\Gamma\\ \Big\|\, t\\ A\end{array}$

Basic types: $A, B, C, D \ ::= \ a \ \mid \ A \to B$

Context types: $\Gamma, \Delta, \Lambda, \Sigma \ ::= \ \top \ \mid \ A \ \mid \ \Gamma \wedge \Delta$

In the calculus, contraction is implicit via the use of variables (and made explicit in the reduction rules by considering variable *occurrences*). Correspondingly, we consider derivations modulo the equivalences ($\triangle\triangle$) and ($\overset{\triangle}{\triangle}$) below right, where in ($\triangle\triangle$) both contractions have the same width (the same number of formulas X respectively Y in the conclusion).

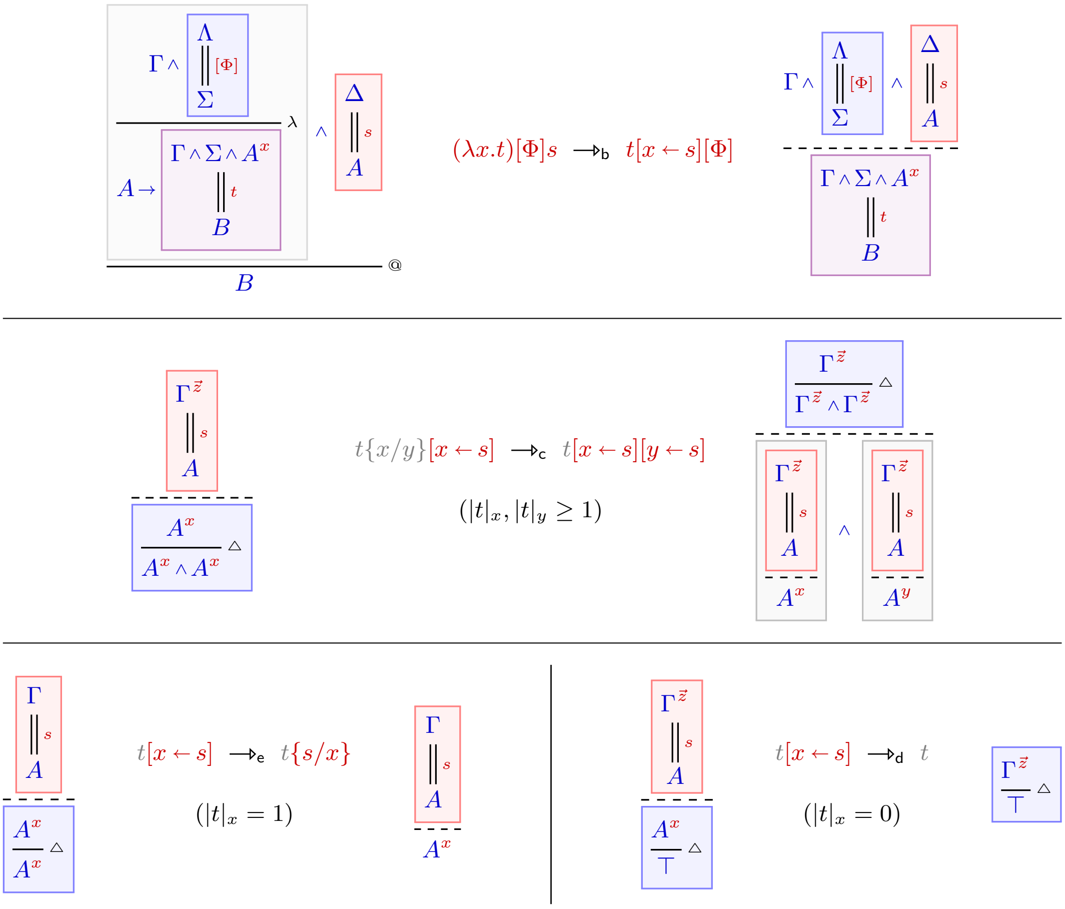

$(\lambda x.t)[\Phi]s \rightarrow_{\mathsf{b}} t[x \leftarrow s][\Phi]$
$t\{x/y\}[x \leftarrow s] \rightarrow_{\mathsf{c}} t[x \leftarrow s][y \leftarrow s]$
$(|t|_x, |t|_y \geq 1)$
$t[x \leftarrow s] \rightarrow_{\mathsf{e}} t\{s/x\}$
$(|t|_x = 1)$
$t[x \leftarrow s] \rightarrow_{\mathsf{d}} t$
$(|t|_x = 0)$

Figure 3 Subject reduction for the simply-typed structural λ-calculus.

$$\boxed{\frac{X}{X\wedge\cdots\wedge X}\,\triangle} \wedge \boxed{\frac{Y}{Y\wedge\cdots\wedge Y}\,\triangle} \quad\sim\quad \boxed{\frac{X\wedge Y}{(X\wedge Y)\wedge\cdots\wedge(X\wedge Y)}\,\triangle} \qquad (\triangle\triangle)$$

$$\frac{\boxed{\dfrac{X}{X\wedge\cdots\wedge X\wedge X}\,\triangle}}{X\wedge\cdots\wedge X\wedge \boxed{\dfrac{X}{X\wedge\cdots\wedge X}\,\triangle}} \quad\sim\quad \boxed{\frac{X}{X\wedge\cdots\wedge X\wedge X\wedge\cdots\wedge X}\,\triangle} \qquad \left(\substack{\triangle\\\triangle}\right)$$

We consider the unary contraction equivalent to an identity (below left), though we may choose to deploy it as a “marker” to differentiate between an *explicit* substitution $t[x \leftarrow s]$ where $|t|_x = 1$ (*with* unary contraction) and the *implicit* substitution $t\{s/x\}$ to which it reduces (*without*). We include a naturality equation for the abstraction (below right), to capture the equation $(\lambda x.t)\{s/z\} = \lambda x.t\{s/z\}$ (where $x \neq z$) for subsitution.

$$\frac{X}{X}\triangle \;\sim\; X \qquad\qquad \frac{\boxed{\begin{array}{c}Y\\ \|\\ Z\end{array}}}{X\to(Z\wedge X)}\lambda \;\sim\; \frac{Y}{X\to\boxed{\boxed{\begin{array}{c}Y\\ \|\\ Z\end{array}}\wedge X}}\lambda$$

Typing derivations for the reduction rules are given in Figure 3. For the rules (c, d, e) we omit the term t from the derivations, for brevity. The figure witnesses that:

▶ **Proposition 6** (Subject reduction). *SC reduction preserves typing.*

4 The collection calculus

We extend the SC with an abstract notion of *collection*, applied to terms, types, and derivations. The resulting *collection calculus* (CC) is a resource λ-calculus that is parameterized in a specific choice of collection, such as sets, multisets, or with a minor modification, sequences. We generate collections syntactically, by combining empty $\langle\rangle$ and singleton $\langle t\rangle$ collections with a binary *append* operator $+$, and then consider these modulo standard algebraic laws.

▶ **Definition 7.** *The* collection calculus *(CC) is given by the* terms *and* collection terms*:*

$$r,s,t \;::=\; x \;\mid\; t\tau \;\mid\; \lambda x.t \;\mid\; t[x\leftarrow\tau] \qquad\qquad \rho,\sigma,\tau \;::=\; \langle\rangle \;\mid\; \langle t\rangle \;\mid\; \sigma+\tau$$

where terms are as for the SC, and collection terms are empty*,* singleton*, and* append*.*

The collection calculus is *parameterized* in a *collection algebra*, a preorder $(\leq)$ over collection terms generated by a selection of algebraic equalities and inequalities, which governs the behaviour of collections under reduction. A reduction step on a closure $t[x\leftarrow\tau]$ will treat the collection τ modulo $(\leq)$: it will correspond to a (non-deterministic) syntactic reduction on $t[x\leftarrow\sigma]$ for a chosen σ such that $\tau\leq\sigma$. Conceptually, $(\leq)$ implements the *structural* aspects of reduction, such as duplication, deletion, and exchange.

The algebraic laws are the following, where $\sigma=\tau$ means that both $\sigma\leq\tau$ and $\tau\leq\sigma$. Commensurate with the intuition of $(\leq)$ as a reduction relation, it satisfies *reflexivity* ($\tau\leq\tau$), *transitivity* (if $\rho\leq\sigma$ and $\sigma\leq\tau$ then $\rho\leq\tau$), and *contextual closure* (if $\rho\leq\sigma$ then $\tau+\rho\leq\tau+\sigma$ and $\rho+\tau\leq\sigma+\tau$). The current presentation further assumes *associativity*, *unitality*, and *symmetry* (below), so that collections are *multisets*. Relaxing these laws will be discussed in Section 7. The laws of *redundancy*, *duplicability*, and *idempotence* are optional parameters.

Associativity	$\rho+(\sigma+\tau) \;=\; (\rho+\sigma)+\tau$	**Redundancy**	$\tau \;\leq\; \langle\rangle$
Unitality	$\langle\rangle+\tau \;=\; \tau \;=\; \tau+\langle\rangle$	**Duplicability**	$\tau \;\leq\; \tau+\tau$
Symmetry	$\sigma+\tau \;=\; \tau+\sigma$	**Idempotence**	$\tau \;=\; \tau+\tau$

Reduction is non-deterministic, and produces an (idempotent) formal sum of terms at the meta-level, distinct from collection terms and the append operator.

▶ **Definition 8.** *The* reduction rules *of the CC are the contextual closure of the rules below.*

$$(\lambda x.t)[\Phi]\tau \;\rightharpoonup_{\mathsf{b}}\; t[x\leftarrow\tau][\Phi] \qquad\qquad \text{(beta)}$$

$$t\{x/y\}[x\leftarrow\tau] \;\rightharpoonup_{\mathsf{c}}\; \sum_{\tau\,\leq\,\rho+\sigma} t[x\leftarrow\rho][y\leftarrow\sigma] \qquad |t|_x,|t|_y\geq 1 \qquad \text{(copy)}$$

$$t[x\leftarrow\tau] \;\rightharpoonup_{\mathsf{e}}\; \sum_{\tau\,\leq\,\langle s\rangle} t\{s/x\} \qquad |t|_x=1 \qquad \text{(evaluate)}$$

$$t[x\leftarrow\tau] \;\rightharpoonup_{\mathsf{d}}\; \sum_{\tau\,\leq\,\langle\rangle} t \qquad |t|_x=0 \qquad \text{(delete)}$$

We set $\rightharpoonup_{\neg\mathsf{b}} \;=\; \rightharpoonup_{\mathsf{c}}\cup\rightharpoonup_{\mathsf{d}}\cup\rightharpoonup_{\mathsf{e}}$*, and* $\rightharpoonup_{\mathsf{cc}} \;=\; \rightharpoonup_{\mathsf{b}}\cup\rightharpoonup_{\neg\mathsf{b}}$.

Observe that for a closure $t[x \leftarrow \tau]$, the number of occurrences $|t|_x$ determines which reduction step applies, while the collection algebra $(\leq)$ determines what reducts are obtained. For the *evaluate* and *delete* steps, the sum implements the possibility of *deadlock*: the result is either a singleton or the empty sum 0. The laws of *redundancy* and *duplicability* allow deletion respectively duplication of the terms in a collection.

▶ **Example 9.** We have the following $\neg\mathsf{b}$-reductions, writing $\langle t_1, \dots, t_n\rangle$ for $\langle t_1\rangle + \cdots + \langle t_n\rangle$, where the row (1) gives the reducts for the plain collection algebra (collections as multisets); (2) gives those with *redundancy*; (3) those with *duplicability*; and (4) those with both laws.

$$\begin{array}{llll}
x\langle x\rangle[x \leftarrow \langle s,t\rangle] \twoheadrightarrow_{\neg\mathsf{b}} & x[x \leftarrow \langle s,t\rangle] \twoheadrightarrow_{\neg\mathsf{b}} & x\langle x\rangle[x \leftarrow \langle s\rangle] \twoheadrightarrow_{\neg\mathsf{b}} & \\
\quad s\langle t\rangle + t\langle s\rangle & \quad 0 & \quad 0 & (1)\\
\quad s\langle t\rangle + t\langle s\rangle & \quad s+t & \quad 0 & (2)\\
\quad s\langle t\rangle + t\langle s\rangle & \quad 0 & \quad s\langle s\rangle & (3)\\
\quad s\langle s\rangle + s\langle t\rangle + t\langle s\rangle + t\langle t\rangle & \quad s+t & \quad s\langle s\rangle & (4)
\end{array}$$

Traditionally, intersection type systems have featured *idempotence* instead of *duplicability*, rendering collections as *sets*. While natural from an algebraic perspective, *duplicability* and *redundancy* are a closer match with the reduction behaviour of contraction and weakening rules. The missing direction is also derived through *redundancy* and *unitality*: $\tau + \tau \;\leq\; \tau + \langle\rangle \;=\; \tau$.

With *duplicability* or *idempotence*, the *copy* reduction step produces infinite sums, since the class of collections $\{\sigma + \rho \mid \tau \leq \sigma + \rho\}$ is infinite as soon as τ is non-empty. However, most duplication may safely be delayed. Writing $\sigma \subseteq \tau$ for the sub-multiset relation (i.e. each element of σ occurs at least as many times in τ), the *copy* rule may be restricted to those reducts where $\rho, \sigma \subseteq \tau$. Likewise, with *redundancy*, deletion may be delayed and relegated to *evaluate* and *delete* steps, restricting *copy* to reducts where $\tau \subseteq \rho + \sigma$. And with both laws, we only need consider $\rho, \sigma = \tau$.

▶ **Example 10.** We consider the reduction from $(\lambda z.z\langle z\rangle)\langle x,x,y\rangle$, which starts as follows.

$$(\lambda z.z\langle z\rangle)\langle x,x,y\rangle \;\rightarrow_{\mathsf{b}}\; z\langle z\rangle[z \leftarrow \langle x,x,y\rangle] \;\rightarrow_{\mathsf{c}}\; \sum_{\langle x,x,y\rangle \,\leq\, \rho+\sigma} w\langle z\rangle[w \leftarrow \rho][z \leftarrow \sigma]$$

With the plain collection algebra, this sum consists of:

$$\begin{array}{l}
\quad w\langle z\rangle[w \leftarrow \langle\rangle][z \leftarrow \langle x,x,y\rangle] \;+\; w\langle z\rangle[w \leftarrow \langle x\rangle][z \leftarrow \langle x,y\rangle] \;+\; w\langle z\rangle[w \leftarrow \langle y\rangle][z \leftarrow \langle x,x\rangle]\\
+\; w\langle z\rangle[w \leftarrow \langle x,x,y\rangle][z \leftarrow \langle\rangle] \;+\; w\langle z\rangle[w \leftarrow \langle x,y\rangle][z \leftarrow \langle x\rangle] \;+\; w\langle z\rangle[w \leftarrow \langle x,x\rangle][z \leftarrow \langle y\rangle]
\end{array}$$

This reduces to zero, since no summand has both ρ and σ as singletons. With *redundancy*, reduction continues as below left – by delaying deletion as discussed previously, the above *copy* step is not affected, and recall that the meta-level sum is idempotent. With only *idempotence*, both occurrences of x may be collapsed and reduction instead continues as below right. While we need to consider additional reducts for the *copy* rule above, such as $w\langle z\rangle[w \leftarrow \langle x,y\rangle][z \leftarrow \langle y\rangle]$, these do not add normal forms, since x and y may not be deleted. With also *redundancy*, $x\langle x\rangle$ or $y\langle y\rangle$ are added as normal forms.

$$\text{redundancy:}\;\; \ldots \twoheadrightarrow_{\mathsf{e}} x\langle x\rangle + x\langle y\rangle + y\langle x\rangle \qquad\qquad \text{idempotence:}\;\; \ldots \twoheadrightarrow_{\mathsf{e}} x\langle y\rangle + y\langle x\rangle$$

Reduction does not produce $x\langle x,y\rangle$ or $y\langle x,y\rangle$, nor does it produce $x\langle\rangle$ or $y\langle\rangle$. By contrast, given *idempotence*, $z\langle z,z\rangle[z \leftarrow \langle x,x,y\rangle]$ *does* (non-deterministically) reduce to $x\langle x,y\rangle$ and $y\langle x,y\rangle$. The meaning of the algebraic laws is not to equate terms, as idempotence would

$z\langle z\rangle$ and $z\langle z,z\rangle$, but to govern the behaviour of collections under reduction. This is standard for resource calculi, and the alternative would severely complicate reduction: evaluating a closure $t[x \leftarrow \tau]$ would require duplication within any collection σ in t where $x \in \mathsf{fv}(\sigma)$.

▶ **Proposition 11.** *The normal forms of* $\rightarrow_{\neg\mathsf{b}}$ *and* $\rightarrow_{\mathsf{CC}}$ *are (non-deterministic sums over) terms of the form* s_0 *respectively* t_0 *given as follows.*

$$s_0 ::= x \mid s_0\langle s_0,\dots,s_0\rangle \mid \lambda x.s_0 \qquad t_1 ::= x \mid t_1\langle t_0,\dots,t_0\rangle \qquad t_0 ::= t_1 \mid \lambda x.t_0$$

▶ **Definition 12.** *The* unfolding $t^\bullet$ *of a* CC*-term* t *and* substitution *for collections* $\{\tau/x\}$ *are defined as follows.*

$$\begin{aligned} x^\bullet &= x \\ (t\tau)^\bullet &= t^\bullet\tau^\bullet \\ (\lambda x.t)^\bullet &= \lambda x.t^\bullet \\ (t[x\leftarrow\tau])^\bullet &= t^\bullet\{\tau^\bullet/x\} \\ \langle t_1,\dots,t_n\rangle^\bullet &= \langle t_1^\bullet,\dots,t_n^\bullet\rangle \end{aligned}$$

$$\begin{aligned} x\{\sigma/x\} &= \textstyle\sum_{\sigma\,\le\,\langle s\rangle} s \\ x\{\sigma/y\} &= \textstyle\sum_{\sigma\,\le\,\langle\rangle} x \qquad (\textit{if } x \neq y) \\ (t\tau)\{\sigma/y\} &= \textstyle\sum_{\sigma\,\le\,\sigma_1+\sigma_2} (t\{\sigma_1/y\})(\tau\{\sigma_2/y\}) \\ (\lambda x.t)\{\sigma/y\} &= \lambda x.(t\{\sigma/y\}) \\ t[x\leftarrow\tau]\{\sigma/y\} &= \textstyle\sum_{\sigma\,\le\,\sigma_1+\sigma_2} t\{\sigma_1/y\}[x\leftarrow\tau\{\sigma_2/y\}] \\ \langle\rangle\{\sigma/y\} &= \textstyle\sum_{\sigma\,\le\,\langle\rangle} \langle\rangle \\ \langle t\rangle\{\sigma/y\} &= \langle t\{\sigma/y\}\rangle \\ (\tau_1+\tau_2)\{\sigma/y\} &= \textstyle\sum_{\sigma\,\le\,\sigma_1+\sigma_2} \tau_1\{\sigma_1/y\}+\tau_2\{\sigma_2/y\} \end{aligned}$$

A single closure $t[x\leftarrow\tau]$ is evaluated by a substitution $t\{\tau/x\}$, and the unfolding of a term evaluates all closures, commensurate with $\neg\mathsf{b}$-reduction to normal form in a way that we make precise below.

▶ **Proposition 13.** $\neg\mathsf{b}$*-Reduction of a* CC*-term* t *is strongly normalizing, and confluent in the following sense: if* $t \twoheadrightarrow_{\neg\mathsf{b}} \sum_{s\in S} s$ *and* $t^\bullet = \sum_{r\in R} r$ *then* $S \subseteq R$*.*

▶ **Proposition 14.** *Without idempotence and duplicability,* CC*-terms are strongly normalizing.*

A main purpose of resource calculi is to provide quantitative bounds on the length of reduction sequences. We will measure the length of *non-deterministic* reduction paths, which select only one term from a formal sum of reducts, by a *reduction weight* $|t|$ derived from the constructors in a term t. With the plain collection algebra, the number of *beta* steps is bounded by the number of abstractions and applications. For non-beta steps, a reduction path $t[x\leftarrow\tau] \twoheadrightarrow_{\neg\mathsf{b}} t\{\tau/x\}$ where $|t|_x = n$ consists of $2n-1$ steps ($n-1$ *copy* steps and n *evaluate* steps) if $n \geq 1$, or one *delete* step if $n=0$. This gives the constraints that variables contribute 2 to the reduction weight, weakenings contribute 1, and sharings otherwise contribute -1. Then $|t|$ is defined as follows, where $x \notin \mathsf{fv}(r)$ but $x \in \mathsf{fv}(s)$:

$$\begin{aligned} |x| &= 2 & |\lambda x.r| &= |r|+2 & |r[x\leftarrow\tau]| &= |r|+|\tau|+1 & \\ |t\tau| &= |t|+|\tau| & |\lambda x.s| &= |s| & |s[x\leftarrow\tau]| &= |s|+|\tau|-1 & |\langle t_1,\dots,t_n\rangle| = \sum_{i=1}^{n}|t_i| \end{aligned}$$

In the absence of further algebraic laws, quantitative bounds are *exact*; with *redundancy*, they are *upper bounds*; and with *duplicability*, they are *lower bounds.*

▶ **Proposition 15.** *The length of a (non-deterministic) reduction sequence* $s \twoheadrightarrow_{\mathsf{CC}} t$ *is:*

1. *without algebraic laws, exactly* $|s|-|t|$*;*
2. *with only* redundancy*, at most* $|s|-|t|$*;*
3. *with only* duplicability*, at least* $|s|-|t|$*.*

5 A deep quantitative type system

Figure 4 gives an open-deduction proof system for a logic of quantitative types. Unlike the two-sorted CC, which has separate sorts for terms and collection terms, the type system is single-sorted, and the constructors *empty* $\langle\rangle$ and *append* $+$ are included with the regular types and derivations. The inference rules include a modified n-ary *contraction* $(\vartriangle)$, with the 0-ary case given below, the $m{\times}n$-ary *medial* (m), specialized to the dimensions $0{\times}0$, $2{\times}0$, $1{\times}1$, $0{\times}2$, and $2{\times}2$ below, and the rule $(\leq)$ that implements the algebraic laws for types. The inequality $(\leq)$ as a typing rule represents a generalized *structural* rule, generalizing the *weakening*, *contraction*, and *exchange* rules familiar from sequent calculi and other proof systems. A similar algebraic rule appears in the intersection type system of [7].

$$\frac{\langle\rangle}{\top}\vartriangle \qquad\qquad \frac{\top}{\langle\rangle}\mathsf{m} \quad \frac{\langle\rangle\wedge\langle\rangle}{\langle\rangle}\mathsf{m} \quad \frac{X}{X}\mathsf{m} \quad \frac{\top}{\top+\top}\mathsf{m} \quad \frac{(W+X)\wedge(Y+Z)}{(W\wedge Y)+(X\wedge Z)}\mathsf{m}$$

The operators *append* and *empty* are parameterized by the same algebraic laws as for collection terms: we assume *associativity*, *unitality*, and *symmetry*, though these can be relaxed without fear of inconsistency, and optional are *redundancy*, *duplicability*, and *idempotence* (as usual, $(\leq)$ is reflexive, transitive and contextual, and $A=B$ means that $A\leq B$ and $B\leq A$).

Associativity	$A+(B+C) \;=\; (A+B)+C$	**Redundancy**	$A \;\leq\; \langle\rangle$
Unitality	$\langle\rangle+A \;=\; A \;=\; A+\langle\rangle$	**Duplicability**	$A \;\leq\; A+A$
Symmetry	$A+B \;=\; B+A$	**Idempotence**	$A \;=\; A+A$

Figure 5 presents a type system for the collection calculus within the open-deductive quantitative system. A term t is typed by a derivation over *structured* types, defined below, with as conclusion a *basic type* A and as premise a *context type* Γ, which is itself a conjunction over *collection types* I. The premise Γ of a derivation for t types a vector of *context variables* $\vec{x}$ that includes the free variables of t, made explicit by writing $\Gamma^{\vec{x}} = I_1^{x_1}\wedge\cdots\wedge I_n^{x_n}$.

$$\text{A derivation for } t\text{:}\quad \begin{array}{c}\Gamma^{\vec{x}}\\ \Big\Vert\, t\\ A\end{array} \qquad \begin{array}{lrcl} \text{Basic types:} & A,B,C,D & ::= & a \;\mid\; I\to A\\ \text{Collection types:} & I,J,K,L & ::= & \langle\rangle \;\mid\; A \;\mid\; I+J\\ \text{Context types:} & \Gamma,\Delta,\Lambda,\Sigma & ::= & \top \;\mid\; I \;\mid\; \Gamma\wedge\Delta \end{array}$$

In typing a collection of terms $\tau=\langle t_1,\dots,t_n\rangle$, the medial generates the type of each context variable x, by combining the types I_1 through I_n for x in each t_i into the type $I_1+\cdots+I_n$. For convenience, we capture the effect of the $n{\times}0$-medial by abbreviating contexts of empty collections by $\Gamma^{\vec{x}}_{\langle\rangle} = \langle\rangle^{x_1}\wedge\cdots\wedge\langle\rangle^{x_n}$, and that of the $n{\times}2$-medial by an operator $(\mathbin{+\!\!\!+})$:

$$\frac{\Gamma^{\vec{x}}_{\langle\rangle}}{\langle\rangle}\mathsf{m} \qquad \frac{(\Gamma\mathbin{+\!\!\!+}\Delta)^{\vec{x}}}{\Gamma^{\vec{x}}+\Delta^{\vec{x}}}\mathsf{m} \quad \text{with } (I_1\wedge\cdots\wedge I_n)\mathbin{+\!\!\!+}(J_1\wedge\cdots\wedge J_n) \;\triangleq\; (I_1+J_1)\wedge\cdots\wedge(I_n+J_n)$$

An example is the derivation (5), for the term $(\lambda z.z\langle z\rangle)\langle x,x,y\rangle$ of Example 10.

$$\dfrac{\boxed{\dfrac{\top}{((A\to B)+A)\to\boxed{\dfrac{\dfrac{((A\to B)+A)^z}{(A\to B)^z\wedge A^z}\vartriangle}{B}@}}\lambda}\;\wedge\;\boxed{\dfrac{\dfrac{\boxed{\dfrac{((A\to B)+A)^x}{((A\to B)+A+\langle\rangle)^x}\leq}\;\wedge\;\boxed{\dfrac{A^y}{(\langle\rangle+\langle\rangle+A)^y}\leq}}{\boxed{\dfrac{(A\to B)^x\wedge\boxed{\dfrac{\langle\rangle^y}{\top}\vartriangle}}{A\to B}=}+\boxed{\dfrac{A^x\wedge\boxed{\dfrac{\langle\rangle^y}{\top}\vartriangle}}{A}=}+\boxed{\dfrac{\boxed{\dfrac{\langle\rangle^x}{\top}\vartriangle}\wedge A^y}{A}=}}\mathsf{m}}{(A\to B)+A}\leq}}{B}@ \tag{5}$$

$$\begin{array}{c}X\\\Vert\\Z\end{array} \;::=\; a \;\mid\; \top \;\mid\; \boxed{\begin{array}{c}X_1\\\Vert\\Z_1\end{array}} \wedge \boxed{\begin{array}{c}X_2\\\Vert\\Z_2\end{array}} \;\mid\; Y \to \boxed{\begin{array}{c}X_2\\\Vert\\Z_2\end{array}} \;\mid\; \dfrac{\boxed{\begin{array}{c}X\\\Vert\\Y_1\end{array}}}{\boxed{\begin{array}{c}Y_2\\\Vert\\Z\end{array}}}\,r \;\mid\; \langle\rangle \;\mid\; \boxed{\begin{array}{c}X_1\\\Vert\\Z_1\end{array}} + \boxed{\begin{array}{c}X_2\\\Vert\\Z_2\end{array}}$$

$$\dfrac{X}{Y\to(X\wedge Y)}\,\lambda \qquad \dfrac{(X\to Y)\wedge X}{Y}\,@ \qquad \dfrac{X\wedge(Y\wedge Z)}{(X\wedge Y)\wedge Z}\,= \qquad \dfrac{X\wedge Y}{Y\wedge X}\,= \qquad \dfrac{X\wedge\top}{X}\,=$$

$$\dfrac{X_1+\cdots+X_n}{X_1\wedge\cdots\wedge X_n}\,\triangle \qquad \dfrac{(X_1^1+\cdots+X_n^1)\wedge\cdots\wedge(X_1^m+\cdots+X_n^m)}{(X_1^1\wedge\cdots\wedge X_1^m)+\cdots+(X_n^1\wedge\cdots\wedge X_n^m)}\,\mathsf{m} \qquad \dfrac{X}{Y}\,\leq \quad (X\leq Y)$$

Figure 4 An open-deduction system for quantitative types.

The inequality for collection terms $\tau \leq \sigma$, which expands during reduction, is captured in derivations by a permutation across the $\leq$ inference rule. We give *associativity* and *duplicability* as an example:

$$\dfrac{\boxed{\begin{array}{c}X\\\Vert\\X'\end{array}} + \boxed{\boxed{\begin{array}{c}Y\\\Vert\\Y'\end{array}} + \boxed{\begin{array}{c}Z\\\Vert\\Z'\end{array}}}}{(X'+Y')+Z'}\,\leq \quad\leq\quad \dfrac{X+(Y+Z)}{\boxed{\boxed{\begin{array}{c}X\\\Vert\\X'\end{array}} + \boxed{\begin{array}{c}Y\\\Vert\\Y'\end{array}}} + \boxed{\begin{array}{c}Z\\\Vert\\Z'\end{array}}}\,\leq \qquad\qquad \dfrac{\boxed{\begin{array}{c}X\\\Vert\\Y\end{array}}}{Y+Y}\,\leq \quad\leq\quad \dfrac{X}{\boxed{\begin{array}{c}X\\\Vert\\Y\end{array}} + \boxed{\begin{array}{c}X\\\Vert\\Y\end{array}}}\,\leq$$

Typing the collection calculus imposes several equivalences on the quantitative open-deduction system. In Figure 6 we give the equivalences due to interaction of contraction and medial, and medial with itself: the first two are the splitting and associativity of contraction $(\triangle\!\triangle)$ and $(\overset{\triangle}{\triangle})$, adjusted from the simple type system for the SC; the latter two $(\overset{\mathsf{m}}{\mathsf{m}})$ and $(\overset{\mathsf{m}}{\mathsf{m}})$ are associativity laws for the medial. We equate the 1×1-medial and the unary contraction with the identity, illustrated below left, though we may choose to leave these rules as "markers" in the derivation for the term constructs $\langle s\rangle$, respectively $t[x\leftarrow\tau]$ where $|t|_x = 1$, to eliminate both with the rewrite rule for $t[x\leftarrow\langle s\rangle]$, as in Figure 7. We assume the following two *naturality* laws, for abstraction and medial.

$$\begin{array}{l}\dfrac{X}{X}\,\mathsf{m} \sim X\\[2ex] \dfrac{X}{X}\,\triangle \sim X\end{array} \qquad \dfrac{\boxed{\begin{array}{c}Y\\\Vert\\Z\end{array}}}{X\to(Z\wedge X)}\,\lambda \;\sim\; \dfrac{Y}{X\to\boxed{\boxed{\begin{array}{c}Y\\\Vert\\Z\end{array}}\wedge X}}\,\lambda \qquad \dfrac{\boxed{\boxed{\begin{array}{c}X_1\\\Vert\\Z_1\end{array}}+X_2}\;\wedge(Y_1+Y_2)}{(Z_1\wedge Y_1)+(X_2\wedge Y_2)}\,\mathsf{m} \;\sim\; \dfrac{(X_1+X_2)\wedge(Y_1+Y_2)}{\boxed{\boxed{\begin{array}{c}X_1\\\Vert\\Z_1\end{array}}+X_2}\;(X_2\wedge Y_2)}\,\mathsf{m}$$

Figure 7 gives typing derivations for the reduction rules of the collection calculus, witnessing the following proposition.

▶ **Proposition 16** (Subject reduction). *The quantitative type system for the collection calculus satisfies subject reduction.*

Typing restricts reduction in the collection calculus: the terms in a collection may have different types, and only those with the same type as a given variable may be substituted for it. The example (6) gives the typed ¬b-reduction (omitting the first b-step) of Example 10

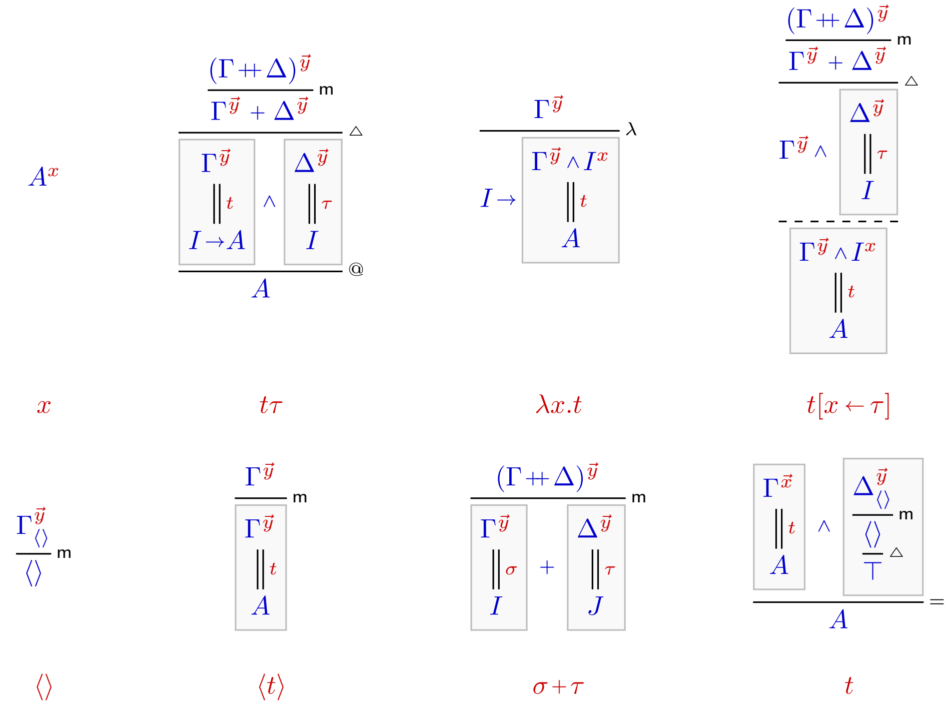
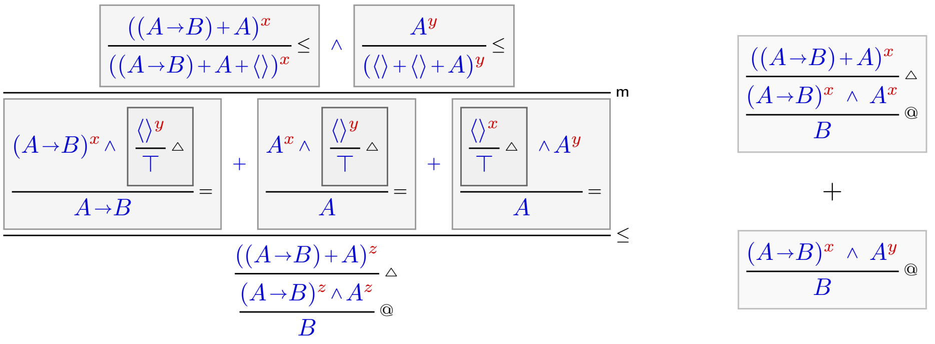


Figure 5 *Typing the collection calculus in open deduction.* In the constructions $t\tau$ and $t[x\leftarrow\tau]$ the contexts of both derivations are expanded to cover the same context variables $\vec{y}$, using the bottom-right figure. For non-empty collection derivations the context is expanded for every term, while empty collections are given for any context variables $\vec{y}$.

with *duplicability* and *redundancy*. The typed reduction results in only two of the four summands of the untyped reduction: the remaining two, $y\langle x\rangle$ and $y\langle y\rangle$, are not well-typed with respect to this reduction, since the derivation for $z\langle z\rangle[z\leftarrow\langle x,x,y\rangle]$ does not assign y the correct type to appear in function position.



$$z\langle z\rangle[z\leftarrow\langle x,x,y\rangle] \twoheadrightarrow_{\neg\mathsf{b}} x\langle x\rangle + x\langle y\rangle \qquad (6)$$

▶ **Theorem 17.** *A typed CC-term is strongly normalizing.*

$$\dfrac{\dfrac{(X_1^{1}+\cdots+X_n^{1})\wedge\cdots\wedge(X_1^{m}+\cdots+X_n^{m})}{(X_1^{1}\wedge\cdots\wedge X_1^{m})+\cdots+(X_n^{1}\wedge\cdots\wedge X_n^{m})}\,\mathsf{m}}{(X_1^{1}\wedge\cdots\wedge X_1^{m}\wedge\cdots\wedge X_n^{1}\wedge\cdots\wedge X_n^{m})}\,\triangle \quad\sim\quad \boxed{\dfrac{X_1^{1}+\cdots+X_n^{1}}{X_1^{1}+\cdots+X_n^{1}}\,\triangle}\wedge\cdots\wedge\boxed{\dfrac{X_1^{m}+\cdots+X_n^{m}}{X_1^{m}+\cdots+X_n^{m}}\,\triangle} \qquad (\triangle\triangle)$$

$$\dfrac{X_1+\cdots+X_n+Y_1+\cdots+Y_m}{X_1\wedge\cdots\wedge X_n\wedge\boxed{\dfrac{Y_1+\cdots+Y_m}{Y_1\wedge\cdots\wedge Y_m}\,\triangle}}\,\triangle \quad\sim\quad \dfrac{X_1+\cdots+X_n+Y_1+\cdots+Y_m}{X_1\wedge\cdots\wedge X_n\wedge Y_1\wedge\cdots\wedge Y_m}\,\triangle \qquad \left(\substack{\triangle\\ \triangle}\right)$$

$$\dfrac{X_1\mathbin{+\!\!+}\cdots\mathbin{+\!\!+}X_n\mathbin{+\!\!+}Y_1\mathbin{+\!\!+}\cdots\mathbin{+\!\!+}Y_m}{X_1+\cdots+X_n+\boxed{\dfrac{Y_1\mathbin{+\!\!+}\cdots\mathbin{+\!\!+}Y_m}{Y_1+\cdots+Y_m}\,\mathsf{m}}}\,\mathsf{m} \quad\sim\quad \dfrac{X_1\mathbin{+\!\!+}\cdots\mathbin{+\!\!+}X_n\mathbin{+\!\!+}Y_1\mathbin{+\!\!+}\cdots\mathbin{+\!\!+}Y_m}{X_1+\cdots+X_n+Y_1+\cdots+Y_m}\,\mathsf{m} \qquad \left(\substack{\mathsf{m}\\ \scriptscriptstyle\mathsf{m}}\right)$$

$$\dfrac{(X_1\mathbin{+\!\!+}\cdots\mathbin{+\!\!+}X_n)\wedge\boxed{\dfrac{Y_1\mathbin{+\!\!+}\cdots\mathbin{+\!\!+}Y_n}{Y_1+\cdots+Y_n}\,\mathsf{m}}}{(X_1\wedge Y_1)+\cdots+(X_n\wedge Y_n)}\,\mathsf{m} \quad\sim\quad \dfrac{(X_1\mathbin{+\!\!+}\cdots\mathbin{+\!\!+}X_n)\wedge(Y_1\mathbin{+\!\!+}\cdots\mathbin{+\!\!+}Y_n)}{(X_1\wedge Y_1)+\cdots+(X_n\wedge Y_n)}\,\mathsf{m} \qquad \left(\substack{\scriptscriptstyle\mathsf{m}\\ \mathsf{m}}\right)$$

Figure 6 Equivalences for contraction and medial.

6 Intersection types

Resource calculi aim to provide a notion of *approximation* of λ-terms, as an alternative to that given by Böhm trees. The purpose of collections in such calculi is to approximate arbitrary duplication (of a function argument) by a pre-determined, finite amount of duplication. Figure 1 in the introduction demonstrates how in deep inference, this pre-determined duplication can be implemented by rewriting. The difference with Böhm trees is exactly that the latter do not separate duplication from beta-reduction, where resource calculi do.

In our case, a CC-term may approximate an SC-term, which we formalize below by the relation $t \succ s$. Via this approximation the quantitative type system of the CC becomes an intersection type system for the SC, similar to the approach of Kfoury [30].

▶ **Definition 18.** *The* uniformity *law requires CC-terms to be* uniform, *as follows. A collection term t* flattens *to an SC-term s, and s* expands *to t, by the inductive relation $t \succ s$:*

$$\frac{}{x \succ x} \qquad \frac{t \succ s \quad \tau \succ u}{t\tau \succ su} \qquad \frac{t \succ s}{\lambda x.t \succ \lambda x.s} \qquad \frac{t \succ s \quad \tau \succ u}{t[x \leftarrow \tau] \succ s[x \leftarrow u]} \qquad \frac{}{\langle\rangle \succ u} \qquad \frac{t \succ u}{\langle t\rangle \succ u} \qquad \frac{\sigma \succ u \quad \tau \succ u}{\sigma + \tau \succ u}$$

A uniform *collection term t is one equipped with a flattening s, written as the pair $t \succ s$.*

Subterms of a uniform term receive their annotation inductively. Observe that s in $t \succ s$ is uniquely defined except at subterms of the form $\langle\rangle$ in t. During reduction, collections must be kept uniform: every term t_i in a collection $\langle t_1, \ldots, t_n\rangle \succ t$ must be reduced simultaneously, along a reduction $t \rightarrow_{\mathsf{SC}} s$. We need the following, which is an immediate induction.

▶ **Proposition 19.** *If $t \succ s \rightarrow_{\mathsf{SC}} s'$ then $t \twoheadrightarrow_{\mathsf{CC}} t' \succ s'$ for some CC-term t'.*

▶ **Definition 20.** *A* uniform reduction step $(t \succ s) \rightarrow (t' \succ s')$ *is a reduction step $s \rightarrow s'$ lifted to a corresponding reduction $t \twoheadrightarrow t'$ along the inductive definition of $\succ$.*

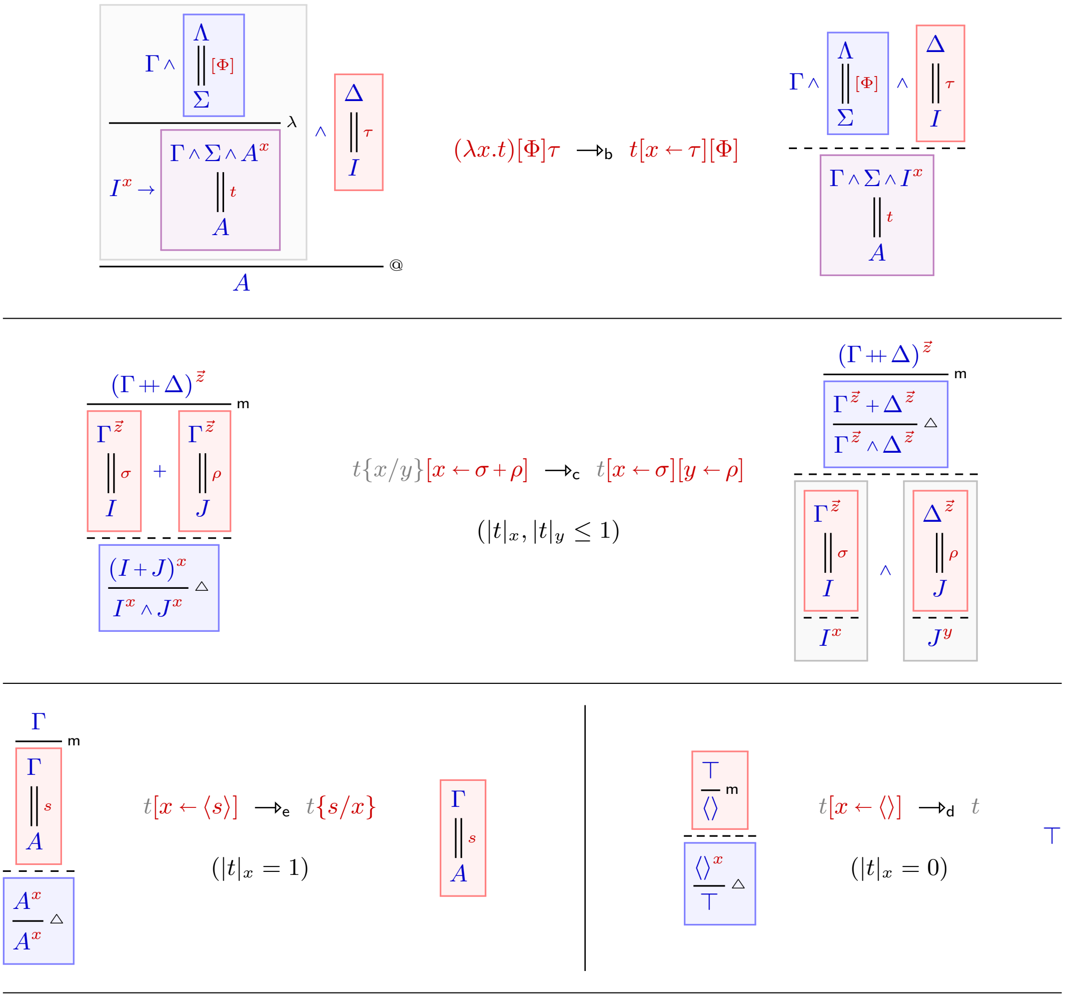
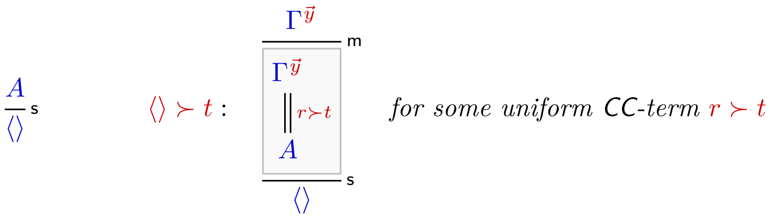

$(\lambda x.t)[\Phi]\tau \rightarrow_{\mathsf{b}} t[x \leftarrow \tau][\Phi]$
$t\{x/y\}[x \leftarrow \sigma + \rho] \rightarrow_{\mathsf{c}} t[x \leftarrow \sigma][y \leftarrow \rho]$
$(|t|_x, |t|_y \leq 1)$
$t[x \leftarrow \langle s \rangle] \rightarrow_{\mathsf{e}} t\{s/x\}$
$(|t|_x = 1)$
$t[x \leftarrow \langle\rangle] \rightarrow_{\mathsf{d}} t$
$(|t|_x = 0)$

Figure 7 Subject reduction for the typed collection calculus.

A derivation for a uniform CC-term $u \succ t$ is an intersection type derivation for the SC-term t. With *idempotence*, we have *idempotent intersection types*; without, *non-idempotent intersection types*. Both characterize weak normalization, since a collection $\langle\rangle \succ t$ may be equipped with a non-normalizing SC-term t, and nevertheless typed by the empty type $\langle\rangle$. To capture *strong* normalization, we adjust the type system to ask a typing witness for t.

▶ **Definition 21.** *The* strength *law introduces an inference rule* s *and replaces the typing law for* $\langle\rangle \succ t$ *by:*



for some uniform CC-term $r \succ t$

▶ **Theorem 22.** *A structural* λ*-term* t *is weakly [strongly] normalizing if and only if there is a typed, [strong,] uniform CC-term* $u \succ t$*.*

7 Discussion and future work

Type uniformity and simple types. Analogous to term uniformity, a *type uniformity* law may require that a quantitative type comes equipped with a flattening onto a *simple type*:

$$\frac{}{a \succ a} \qquad \frac{I \succ B \quad A \succ C}{I {\to} A \succ B {\to} C} \qquad \frac{}{\langle\rangle \succ A} \qquad \frac{I \succ A \quad J \succ A}{I {+} J \succ A} \qquad \frac{}{\top \succ \top} \qquad \frac{\Gamma \succ \Sigma \quad \Delta \succ \Lambda}{\Gamma {\wedge} \Delta \succ \Sigma {\wedge} \Lambda}$$

Without further algebraic laws, *type uniformity* gives a quantitative version of simple types. With also *redundancy* and *duplicability* quantitative types collapse to simple types, as a collection and its flattening will behave equivalently. With *type uniformity* but not *term uniformity*, opposite to intersection types, we obtain a typed *non-deterministic* calculus, where a collection $\langle t_1, \ldots, t_n \rangle$ of type $I \succ A$ represents a non-deterministic choice over terms t_i of type A. Exploring this connection more deeply is future work.

Relaxing associativity, unitality, symmetry. In the quantitative type system, the laws of *associativity*, *unitality*, and *symmetry* apply to collections, and separately to the conjunction, through the corresponding proof rules. These laws can safely be relaxed, though for that to be meaningful, they must be relaxed for the conjunction as well as for collections.

For the collections of the CC, these laws can likewise be relaxed straightforwardly. However, corresponding to the conjunction in the type system is the variable policy of the CC, where *associativity*, *unitality*, and *symmetry* are implicit. Technically, the free variables of a term form a *set*, though since their number of occurrences is significant – in particular, it drives the reduction rules – they morally form a *multiset*. To relax these laws, then, the calculus must be reformulated. Consider the following *linear* variant of the structural λ-calculus, where variables occur once but abstractions and closures bind a vector $\vec{x} = x_1 \ldots x_n$ of variables:

$$t \ ::= \ x \ \mid \ t\tau \ \mid \ \lambda\vec{x}.t \ \mid \ t[\vec{x} \leftarrow \tau]$$

Symmetry of conjunction becomes explicit in the order of variables in a vector $\vec{x}$. Relaxing symmetry of $(+)$ but not $(\wedge)$ yields a linear λ-calculus for intuitionistic multiplicative linear logic as in [8], where $(\to)$ behaves as $(\multimap)$, and $(\wedge)$ and $(+)$ together behave as $(\otimes)$ (strictly, also the distinction between terms and collection terms should be collapsed). Symmetry of conjunction is relaxed by imposing that variables are bound in the same order as they occur; i.e. the free variables of a term must become a vector, and binding restricted accordingly:

$$\begin{aligned}
\mathsf{fv}(x) &= x \\
\mathsf{fv}(t\tau) &= \vec{x}\vec{y} && \text{where } \mathsf{fv}(t) = \vec{x},\ \mathsf{fv}(\tau) = \vec{y} \\
\mathsf{fv}(\lambda\vec{x}.t) &= \vec{y} && \text{where } \mathsf{fv}(t) = \vec{y}\vec{x} \\
\mathsf{fv}(t[\vec{y}_1 \leftarrow \tau]) &= \vec{x}\vec{y}_2\vec{z} && \text{where } \mathsf{fv}(t) = \vec{x}\vec{y}_1\vec{z},\ \mathsf{fv}(\tau) = \vec{y}_2\ .
\end{aligned}$$

Such a construction significantly reduces the expressivity of the calculus. However, it is possible that such non-commutativity can model aspects of sequential (imperative) computation; investigating this is future work. Going further, *associativity* or *unitality* can be relaxed by replacing vectors $\vec{x}$ with collections over variables, akin to *pattern-matching*, though the limited expressiveness casts doubt on how useful this would be.

Further algebraic laws. The present exposition restricts itself to a sample of common algebraic laws. It is clear that there is potentially a large range of laws that would fit within the current framework, as long as the fundamental proof rules $\triangle$, m, $\leq$ are unimpeded. Establishing this range is future work. In a similar direction, connections with the exponentials of linear logic, and their *light* versions [24], are also of interest.

References

1 Beniamino Accattoli. An abstract factorization theorem for explicit substitutions. In *23rd International Conference on Rewriting Techniques and Applications, RTA 2012*, volume 15 of *LIPIcs*, pages 6–21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.RTA.2012.6.

2 Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds, fully developed. *Journal of Functional Programming*, 30:e14, 2020. doi:10.1017/S095679682000012X.

3 Beniamino Accattoli and Delia Kesner. The structural lambda-calculus. In *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference*, volume 6247 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2010. doi:10.1007/978-3-642-15205-4_30.

4 Beniamino Accattoli and Delia Kesner. Preservation of strong normalisation modulo permutations for the structural lambda-calculus. *Logical Methods in Computer Science*, 8(1), 2012. doi:10.2168/LMCS-8(1:28)2012.

5 Juan P. Aguilera and Matthias Baaz. Unsound inferences make proofs shorter. *Journal of Symbolic Logic*, 84(1):102–122, 2019. doi:10.1017/jsl.2018.51.

6 Andrea Aler Tubella and Alessio Guglielmi. Subatomic proof systems: Splittable systems. *ACM Transactions on Computational Logic (TOCL)*, 19(1):5:1–5:33, 2018. doi:10.1145/3173544.

7 Hendrik Pieter Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *The Journal of Symbolic Logic*, 48(4):931–940, 1983. doi:10.2307/2273659.

8 Nick Benton, Gavin Bierman, Valeria de Paiva, and Martin Hyland. A term calculus for intuitionistic linear logic. In *International Conference on Typed Lambda Calculi and Applications, TLCA '93*, volume 664 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 1993. doi:10.1007/BFb0037099.

9 Gérard Boudol. The lambda-calculus with multiplicities. In *CONCUR '93, 4th International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 1993. doi:10.1007/3-540-57208-2_1.

10 Kai Brünnler and Richard McKinley. An algorithmic interpretation of a deep inference system. In *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008*, volume 5330 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2008. doi:10.1007/978-3-540-89439-1_34.

11 Kai Brünnler and Alwen Tiu. A local system for classical logic. In *Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001*, volume 2250 of *Lecture Notes in Computer Science*, pages 347–361. Springer, 2001. doi:10.1007/3-540-45653-8_24.

12 Paola Bruscoli, Alessio Guglielmi, Tom Gundersen, and Michel Parigot. Quasipolynomial normalisation in deep inference via atomic flows. *Logical Methods in Computer Science*, 12(2), 2016. doi:10.2168/LMCS-12(2:5)2016.

13 Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics: the exponentials. *Annals of Pure and Applied Logic*, 109(3):205–241, 2001. doi:10.1016/S0168-0072(00)00056-7.

14 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017. doi:10.1093/jigpal/jzx018.

15 Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for lambda-terms. *Archiv für mathematische Logik und Grundlagenforschung*, 19(1):139–156, 1978. doi:10.1007/BF02011875.

16 Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980. doi:10.1305/ndjfl/1093883253.

17 Ugo Dal Lago, Giulio Guerrieri, and Willem Heijltjes. Decomposing probabilistic lambda-calculi. In *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020*, volume 12077 of *Lecture Notes in Computer Science*, pages 136–156. Springer, 2020. doi:10.1007/978-3-030-45231-5_8.

18 Daniel de Carvalho. The relational model is injective for multiplicative exponential linear logic. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016*, volume 62 of *LIPIcs*, pages 41:1–41:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.41.

19 Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Mathematical Structures in Computer Science*, 28(7):1169–1203, 2018. doi:10.1017/S0960129516000396.

20 Thomas Ehrhard. Non-idempotent intersection types in logical form. In *Foundations of Software Science and Computation Structures, FoSSaCS 2020*, volume 12077 of *Lecture Notes in Computer Science*, pages 198–216. Springer, 2020. doi:10.1007/978-3-030-45231-5_11.

21 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1–3):1–41, 2003. doi:10.1016/S0304-3975(03)00392-X.

22 Thomas Ehrhard and Laurent Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. *Theoretical Computer Science*, 403:347–372, 2008. doi:10.1016/j.tcs.2008.06.001.

23 Philippa Gardner. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software, International Conference TACS '94*, volume 789 of *Lecture Notes in Computer Science*, pages 555–574. Springer, 1994. doi:10.1007/3-540-57887-0_115.

24 Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998. doi:10.1006/inco.1998.2700.

25 Alessio Guglielmi, Tom Gundersen, and Michel Parigot. A proof calculus which reduces syntactic bureaucracy. In *21st International Conference on Rewriting Techniques and Applications, RTA 2010*, volume 6 of *LIPIcs*, pages 135–150. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.RTA.2010.135.

26 Tom Gundersen, Willem Heijltjes, and Michel Parigot. Atomic lambda-calculus: a typed lambda-calculus with explicit sharing. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013*, pages 311–320. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.37.

27 Willem Heijltjes and Joe Paulus. Deep-inference intersection types. Extended abstract, presented at the workshop Twenty Years of Deep Inference (TYDI), Oxford, 2018. Available at http://willem.heijltj.es/pdf/2018-heijltjes-paulus.pdf, 2018.

28 Emil Jeřábek. Proof complexity of the cut-free calculus of structures. *Journal of Logic and Computation*, 19(2):323–339, 2009. doi:10.1093/logcom/exn054.

29 Delia Kesner and Stéphane Lengrand. Resource operators for lambda-calculus. *Information and Computation*, 205(4):419–473, 2007. doi:10.1016/j.ic.2006.08.008.

30 Assaf J. Kfoury. A linearization of the lambda-calculus and consequences. *Journal of Logic and Computation*, 10(3):411–436, 2000. doi:10.1093/logcom/10.3.411.

31 Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. Weighted relational models of typed lambda-calculi. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013*, pages 301–310. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.36.

32 Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *Proceedings of the ACM on Programming Languages*, 2(POPL), 2018. doi:10.1145/3158094.

33 C.-H. Luke Ong. Quantitative semantics of the lambda-calculus: Some generalisations of the relational model. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005064.

34 Luca Paolini, Mauro Piccolo, and Simona Ronchi Della Rocca. Essential and relational models. *Mathematical Structures in Computer Science*, 27(5):626–650, 2017. doi:10.1017/S0960129515000316.

35 Elaine Pimentel, Simona Ronchi Della Rocca, and Luca Roversi. Intersection Types from a proof-theoretic perspective. *Fundamenta Informaticae*, 121(1-4):253–274, 2012. doi:10.3233/FI-2012-778.

36 Garrel Pottinger. A type assignment for the strongly normalizable lambda-terms. In J. Hindley and J. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, 1980.

37 David Sherratt, Willem Heijltjes, Tom Gundersen, and Michel Parigot. Spinal atomic lambda-calculus. In *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020*, volume 12077 of *Lecture Notes in Computer Science*, pages 582–601. Springer, 2020. doi:10.1007/978-3-030-45231-5_30.

38 Alwen Tiu. A local system for intuitionistic logic. In *Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference, LPAR 2006*, volume 4246 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2006. doi:10.1007/11916277_17.

39 Alwen Tiu. A system of interaction and structure II: The need for deep inference. *Logical Methods in Computer Science*, 2(2):4:1–24, 2006. doi:10.2168/LMCS-2(2:4)2006.

Appendix

A Encoding non-idempotent intersection types and resource calculi

Resource $\boldsymbol{\lambda}$-calculus. The *λ-calculus with multiplicities* by Boudol [9] features two-sorted collections P, Q with both non-duplicable and duplicable elements, the latter indicated M^∞:

$$M,N \ ::= \ x \mid NP \mid \lambda x.N \mid N[P/x] \qquad P,Q \ ::= \ 1 \mid M \mid (P|P) \mid M^\infty$$

It employs *weak head reduction*, with β-reduction occurring in head context H and closures evaluated by substituting into head variables $H\{x\}$: (borrowing the CC-notation $[\Phi]$)

$$H \ ::= \ \{\} \mid HP \mid H[P/x] \qquad \begin{array}{rcl} H\{(\lambda x.N)[\Phi]P\} & \rightarrow_{\mathsf{b}} & H\{N[P/x][\Phi]\} \\ H\{x\}[(N|P)/x] & \rightarrow_{\mathsf{s}} & H\{N\}[P/x] \end{array}$$

Note that s-reduction is non-deterministic. The two-sorted collections can be imported into the CC ad-hoc by admitting collections t^∞ and the law $t^\infty \leq \langle t\rangle + t^\infty$. b-Reduction is as in the CC, and informally translating N to t, P to τ, and H to h, s-reduction is simulated by:

$$h\{x\}[x \leftarrow \langle t\rangle + \tau] \ \rightarrow_{\mathsf{c}} \ h\{y\}[y \leftarrow \langle t\rangle][x \leftarrow \tau] \ \rightarrow_{\mathsf{e}} \ h\{t\}[x \leftarrow \tau]$$

Non-idempotent intersection types. Kfoury in [30] gives non-idempotent intersection types via a resource calculus with *uniformity.* Resource terms are given below, where a term N is *well-formed* if it flattens to a regular λ-term $|N|$. β-Reduction on N is defined along $|N|$.

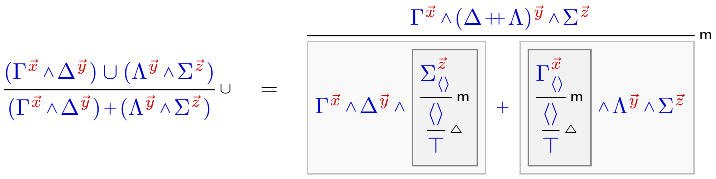
$$M,N,P \ ::= \ x \mid \lambda x.N \mid N.\,P_1 \wedge \cdots \wedge P_n \quad (n \geq 1)$$

$$\frac{}{|x| = x} \qquad \frac{|N| = M}{|\lambda x.N| = \lambda x.M} \qquad \frac{|N| = M \quad \{|P_i| = Q\}_{1 \leq i \leq n}}{|N.\,P_1 \wedge \cdots \wedge P_n| = M\,Q}$$

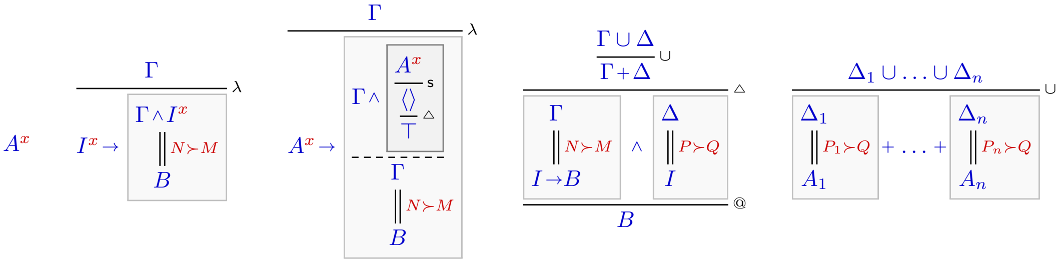
A system of simple types $\boldsymbol{\lambda}^\wedge$ for the resource calculus, given below, then generates a system of non-idempotent intersection types $\boldsymbol{\lambda}$ for the regular λ-calculus by flattening the terms in each typing rule. We use our notational conventions for types, and define the $\cup$ operator by letting $(\Gamma^{\vec{x}} \wedge \Delta^{\vec{y}}) \cup (\Lambda^{\vec{y}} \wedge \Sigma^{\vec{z}}) = \Gamma^{\vec{x}} \wedge (\Delta + \Lambda)^{\vec{y}} \wedge \Sigma^{\vec{z}}$ if $\vec{x}$ and $\vec{z}$ share no variables.

$$\frac{}{x:A\vdash x:A}\qquad \frac{\Gamma,x:I\vdash N:B}{\Gamma\vdash \lambda x.N:I\to B}\,{\scriptstyle I\neq\langle\rangle}\qquad \frac{\Gamma\vdash N:B}{\Gamma\vdash \lambda x.N:A\to B}\qquad \frac{\Gamma\vdash N:(A_1+\ldots+A_n)\to B \quad \{\Delta_i\vdash P_i:A_i\}_{1\le i\le n}}{\Gamma\cup\Delta_1\cup\cdots\cup\Delta_n\vdash N.\,P_1\wedge\cdots\wedge P_n:B}$$

Systems $\boldsymbol{\lambda}^{\wedge}$ and $\boldsymbol{\lambda}$ are effectively a restriction of the typed collection calculus to terms without closures, with the laws of *uniformity* and *strength*. Strictly, also *symmetry* is dropped, but there is no loss of expressiveness because the conjunction remains symmetric and because of *uniformity*. We introduce an admissible proof rule for the $\cup$ operator:



Systems $\boldsymbol{\lambda}^{\wedge}$ and $\boldsymbol{\lambda}$ are then encoded as follows, where $N \succ M$ is $|N| = M$, and where the constructions for application and collections are kept separate, with $\Delta = \Delta_1 \cup \cdots \cup \Delta_n$, $P = P_1 \wedge \cdots \wedge P_n$, and $I = A_1 + \cdots + A_n$.



B Omitted proofs and lemmas in Section 2

▶ Remark 23 (Free variable and translation). For every SC-term t, $\mathsf{fv}(t^\bullet) \subseteq \mathsf{fv}(t)$. The proof is by straightforward induction on t.

▶ **Lemma 24** (Substitution). *For any sharing terms t and u, we have $(t\{u/x\})^\bullet = t^\bullet\{u^\bullet/x\}$.*

Proof. By straightforward induction on t. The only interesting case is the one with sharing: if $t = r[y \leftarrow s]$ (we can suppose without loss of generality that $y \notin \mathsf{fv}(u) \cup \{x\}$), then $t\{u/x\} = r\{u/x\}\{u/x\}[y \leftarrow s\{u/x\}]$ and $t^\bullet = r^\bullet\{s^\bullet/y\}$; by induction hypothesis, $r' = (r\{u/x\})^\bullet = r^\bullet\{u^\bullet/x\}$ and $s' = (s\{u/x\})^\bullet = s^\bullet\{u^\bullet/x\} = s'$, hence

$$(t\{u/x\})^\bullet = r'^\bullet\{s'^\bullet/y\} = r^\bullet\{u^\bullet/x\}\{s^\bullet\{u^\bullet/x\}/y\} = r^\bullet\{s^\bullet/y\}\{u^\bullet/x\} = t^\bullet\{u^\bullet/x\}. \qquad \blacktriangleleft$$

▶ **Proposition 3** (Simulations). *Let t be a SC-term and s be a λ-term.*

1. From SC to λ-calculus*: If $t \to_{\mathsf{b}} t'$ then $t^\bullet \to_\beta t'^\bullet$; if $t \to_{\neg\mathsf{b}} t'$ then $t^\bullet = t'^\bullet$.*
2. From λ-calculus to SC*: If $s \to_\beta s'$ then $s \to_{\mathsf{b}}\twoheadrightarrow_{\neg\mathsf{b}} s'$.*

Proof. 1. Both proofs are by induction on the SC-term t. We omit some cases that easily follows from the induction hypothesis.

- Let us prove that if $t \to_{\mathsf{b}} t'$, then $t^\bullet \twoheadrightarrow_\beta t'^\bullet$. Cases of interest:

  - If $t = (\lambda y.s)[x_1 \leftarrow r_1] \dots [x_n \leftarrow r_n]u \to_{\mathsf{b}} s[y \leftarrow u][x_1 \leftarrow r_1] \dots [x_n \leftarrow r_n] = t'$, then we can suppose without loss of generality that $y \notin \bigcup_{i=1}^{n} \mathsf{fv}(r_i) \cup \{x\}$ and so

$$\begin{aligned} t^\bullet &= (\lambda y.s^\bullet)\{r_1^\bullet/x_1\} \dots \{r_n^\bullet/x_n\}u^\bullet = (\lambda y.s^\bullet\{r_1^\bullet/x_1\} \dots \{r_n^\bullet/x_n\})u^\bullet \\ &\to_\beta s^\bullet\{r_1^\bullet/x_1\} \dots \{r_n^\bullet/x_n\}\{u^\bullet/y\} = s^\bullet\{u^\bullet/y\}\{r_1^\bullet/x_1\} \dots \{r_n^\bullet/x_n\} \\ &\qquad = (s[y \leftarrow u])^\bullet\{r_1^\bullet/x_1\} \dots \{r_n^\bullet/x_n\} = t'^\bullet \end{aligned}$$

    where the second to last equality holds because of substitution lemma (Lemma 24).
  - If $t = u[x \leftarrow s] \to_{\mathsf{b}} u[x \leftarrow s'] = t'$ with $s \to_{\mathsf{b}} s'$ then, by induction hypothesis, $s^\bullet \to_\beta^* s'^\bullet$ and so $t^\bullet = u^\bullet\{s^\bullet/x\} \to_\beta^* u^\bullet\{s'^\bullet/x\} = t'^\bullet$.

- Let us prove that if $t \to_{\neg\mathsf{b}} t'$, then $t^\bullet = t'^\bullet$. Cases of interest:
  - *Copy*: if $t = s\{x/y\}[x \leftarrow u] \to_{\mathsf{c}} s[x \leftarrow u][y \leftarrow u] = t'$, we can suppose without loss of generality that $y \notin \mathsf{fv}(u)$. By Lemma 24, $(s\{x/y\})^\bullet = s^\bullet\{x/y\}$ and $(s[x \leftarrow u])^\bullet = s^\bullet\{u^\bullet/x\} = (s\{u/x\})^\bullet$. Thus,

$$t^\bullet = (s\{x/y\})^\bullet\{u^\bullet/x\} = s^\bullet\{x/y\}\{u^\bullet/x\} = s^\bullet\{u^\bullet/x\}\{u^\bullet/y\} = (s[x \leftarrow u])^\bullet\{u^\bullet/y\} = t'^\bullet.$$

  - *Delete*: if $t = s[x \leftarrow u] \to_{\mathsf{d}} s = t'$ then $x \notin \mathsf{fv}(s)$ and hence $x \notin \mathsf{fv}(s^\bullet)$ by Remark 23, so $t^\bullet = s^\bullet\{u^\bullet/x\} = s^\bullet = t'^\bullet$.
  - *Evaluate*: if $t = u[x \leftarrow s] \to_{\mathsf{e}} u\{s/x\} = t'$ then, by substitution lemma (Lemma 24),

$$t^\bullet = s^\bullet\{u^\bullet/x\} = (s\{u/x\})^\bullet = t'^\bullet.$$

2. See [4, proof of Lemma 2.4] and apply Proposition 4.1 below (proved independently). ◀

▶ **Proposition 4** (Collated results from [4]). *The SC has the following key properties.*

1. *The normal forms of $\to_{\neg\mathsf{b}}$ are exactly the λ-terms.*
2. *The normal forms of $\to_{\mathsf{sc}}$ are exactly the β-normal λ-terms.*
3. *For any SC-term t, one has $t \twoheadrightarrow_{\neg\mathsf{b}} t^\bullet$; in particular, $t = t^\bullet$ for any λ-term.*
4. *The relations $\to_{\mathsf{b}}$, $\to_{\neg\mathsf{b}}$, and $\to_{\mathsf{sc}}$ are confluent; $\to_{\mathsf{b}}$ and $\to_{\neg\mathsf{b}}$ are strongly normalizing.*
5. *Preservation of strong normalization: if a λ-term t has an infinite $\to_{\mathsf{sc}}$-reduction, then it has an infinite $\to_\beta$-reduction.*

Proof.

1. Clearly, every λ-term is normal for $\to_{\neg\mathsf{b}}$ because there is no sharing. Conversely, if t is a sharing term that is normal for $\to_{\neg\mathsf{b}}$ then there are no context C and no sharing terms s and u such that $t = C\langle s[x \leftarrow u]\rangle$, otherwise if $|s|_x = 0$ then t would not be normal for $\to_{\mathsf{d}}$, if $|s|_x = 1$ then t would not be normal for $\to_{\mathsf{e}}$, if $|s|_x > 1$ then t would not be normal for $\to_{\mathsf{c}}$; therefore, t has no sharings and hence is a λ-term.
2. Since $\to_{\mathsf{sc}} = \to_{\mathsf{b}} \cup \to_{\neg\mathsf{b}}$ and in SC the normal forms of $\to_{\neg\mathsf{b}}$ are exactly the λ-terms, it is enough to observe that a λ-term t is normal for $\to_{\mathsf{b}}$ if and only if there are no λ-context C and λ-terms s and u such that $t = C\langle(\lambda x.s)u\rangle$, which amounts to say that t is β-normal.
3. First, if t is a λ-term then $t^\bullet = t$ since there are no sharings in t. Now, let t be a SC-term, with $t \twoheadrightarrow_{\neg\mathsf{b}} s$ (such a s exists because $\to_{\neg\mathsf{b}}$ is strongly normalizing, [4, Lemma 2.10]); as s is a λ-term (Proposition 4.1), we have just shown that $s = s^\bullet$; by Proposition 3.1, $s^\bullet = t^\bullet$ and so $t \twoheadrightarrow_{\neg\mathsf{b}} t^\bullet$.

4. Strong normalization of $\rightarrow_{\mathsf{b}}$ is trivial (each step decreases the number of applications). Strong normalization of $\rightarrow_{\neg\mathsf{b}}$ is proved in [4, Lemma 2.10]
   Accattoli and Kesner [3, 4] already proved confluence of $\rightarrow_{\mathsf{sc}}$ (using Tait–Martin-Löf's technique based on parallel reduction) and of $\rightarrow_{\neg\mathsf{b}}$ (via Newman's lemma). Here we present a simpler, modular and more informative proof, which relies on the confluence of β reduciton.
   It is easy to check that $\rightarrow_{\mathsf{b}}$ has the diamond property and hence is confluent.
   Concerning confluence of $\rightarrow_{\neg\mathsf{b}}$, suppose $s \;{}_{\neg\mathsf{b}}\!\twoheadleftarrow t \twoheadrightarrow_{\neg\mathsf{b}} u$. By Proposition 4.3, $s \twoheadrightarrow\!\!\shortmid_{\neg\mathsf{b}} s^\bullet$ and $r \twoheadrightarrow\!\!\shortmid_{\neg\mathsf{b}} r^\bullet$. According to Proposition 3.1, $s^\bullet = t^\bullet = r^\bullet$.
   Concerning confluence of $\rightarrow_{\mathsf{sc}}$, suppose $s \;{}_{\mathsf{sc}}\!\twoheadleftarrow t \twoheadrightarrow_{\mathsf{sc}} u$. By Proposition 4.3, $s \twoheadrightarrow\!\!\shortmid_{\neg\mathsf{b}} s^\bullet$ and $r \twoheadrightarrow\!\!\shortmid_{\neg\mathsf{b}} r^\bullet$. According to Proposition 3.1, $s^\bullet \;{}_{\beta}\!\twoheadleftarrow t^\bullet \twoheadrightarrow_{\beta} r^\bullet$. By confluence of $\rightarrow_{\beta}$, $s^\bullet \twoheadrightarrow_{\beta} u \;{}_{\beta}\!\twoheadleftarrow r^\bullet$ and so $s \twoheadrightarrow_{\neg\mathsf{b}} s^\bullet \rightarrow_{\mathsf{b}}\twoheadrightarrow_{\neg\mathsf{b}} u \;{}_{\neg\mathsf{b}}\!\twoheadleftarrow{}_{\mathsf{b}}\!\leftarrow r^\bullet \;{}_{\neg\mathsf{b}}\!\twoheadleftarrow r$ by Proposition 3.2.
5. See [4, Lemma 3.5] ◀

C Omitted proofs and lemmas in Section 4

Let $|t|_x^{\mathrm{p}}$ be the maximal number of free occurrences of x that may appear in a ¬b-reduction sequence from the CC-term t. Formally:

$$|x|_x^{\mathrm{p}} = 1 \qquad |y|_x^{\mathrm{p}} = 0 \qquad |t\tau|_x^{\mathrm{p}} = |t|_x^{\mathrm{p}} + |\tau|_x^{\mathrm{p}} \qquad |\lambda y.t|_x^{\mathrm{p}} = |t|_x^{\mathrm{p}}$$

$$|t[y \leftarrow \tau]|_x^{\mathrm{p}} = |t|_x^{\mathrm{p}} + \max\{1, |t|_y^{\mathrm{p}}\} \cdot |\tau|_x^{\mathrm{p}} \qquad |\langle t_1, \dots, t_n\rangle|_x^{\mathrm{p}} = \sum_{i=1}^{n} |t_i|_x^{\mathrm{p}}$$

▶ Remark 25. For any CC-term, $|t|_x^{\mathrm{p}} = 0$ if and only if $|t|_x = 0$.

▶ **Lemma 26.** *For any CC-term t, if $|t|_x, |t|_y \geq 1$, then $|t\{x/y\}|_x^{\mathrm{p}} = |t|_x^{\mathrm{p}} + |t|_y^{\mathrm{p}}$*

Proof. By induction on t. ◀

Inspired by [4], let us define the *size* $\|t\|$ of a CC-term t the following multiset of natural numbers (well-ordered by the multiset ordering):

$$\|x\| = [\,] \qquad \|\lambda x.t\| = \|t\| \qquad \|t\tau\| = \|t\| + \|\tau\|$$

$$\|t[y \leftarrow \tau]\| = \|t\| + [|t|_x^{\mathrm{p}}] + \max\{1, |t|_x^{\mathrm{p}}\} \cdot \|\tau\| \qquad \|\langle t_1, \dots, t_n\rangle\| = \sum_{i=1}^{n} \|t_i\|$$

▶ **Lemma 27.** *Without* duplicability, *$\|\tau\| \geq \|\sigma\|$ for any collection $\tau \geq \sigma$*

Proof. Trivial. ◀

▶ **Lemma 28.** *Let u be a CC-term.*
1. *If $|u|_x = 1$ then $\|u[x \leftarrow \tau]\| > \|u\{\langle s\rangle/x\}\|$ for any CC-term s such that $\tau \geq \langle s\rangle$.*
2. $\|u\{x/y\}\| = \|u\|$.

Proof. Both are by induction on the CC-term t. ◀

▶ **Proposition 13.** ¬b-*Reduction of a CC-term t is strongly normalizing, and confluent in the following sense: if $t \twoheadrightarrow\!\!\shortmid_{\neg\mathsf{b}} \sum_{s \in S} s$ and $t^\bullet = \sum_{r \in R} r$ then $S \subseteq R$.*

Proof. Let us prove that $t \rightarrow_{\neg\mathsf{b}} \sum_{i=1}^{n} t_i \ni t'$ implies $\|t\| > \|t'\|$, by induction on t. Cases:
- *Delete*: if $t = s[x \leftarrow \tau] \rightarrow_{\mathsf{d}} \sum_{\tau \leq \langle\rangle} s \ni t'$ with $|s|_x = 0$, then $t' = s$. By Remark 25, $|s|_x^{\mathrm{p}} = 0$. So, $\|t\| = \|s\| + [|s|_x^{\mathrm{p}}] + \max\{1, |s|_x^{\mathrm{p}}\} \cdot \|\tau\| = \|s\| + [|s|_x^{\mathrm{p}}] + \|\tau\| > \|s\| = \|t'\|$.

- *Substitution*: if $t = u[x \leftarrow \tau] \to_{\mathsf{e}} \sum_{\tau \leq \langle s \rangle} u\{s/x\} \ni t'$ with $|u|_x = 1$, then $t' = u\{s/x\}$. By Lemma 28.1, $\|t\| = \|u[x \leftarrow \tau]\| > \|u\{\langle s \rangle / x\}\| = t'$.
- *Copy*: if $t = s\{x/y\}[x \leftarrow \tau] \to_{\mathsf{c}} \sum_{\tau \leq \rho + \sigma} s[x \leftarrow \rho][y \leftarrow \sigma] \ni t'$ with $|s|_x, |s|_y \geq 1$, then $t' = s[x \leftarrow \rho][y \leftarrow \sigma]$.

$$\begin{aligned}
\|t\| &= \|s\{x/y\}\| + [|s\{x/y\}|^{\mathrm{p}}_x] + \max\{1, |s\{x/y\}|^{\mathrm{p}}_x\} \cdot \|\tau\| \\
&= \|s\| + [|s|^{\mathrm{p}}_x + |s|^{\mathrm{p}}_y] + \max\{1, |s\{x/y\}|^{\mathrm{p}}_x\} \cdot \|\tau\| && \text{L. 28.2 and 26} \\
&= \|s\| + [|s|^{\mathrm{p}}_x + |s|^{\mathrm{p}}_y] + |s\{x/y\}|^{\mathrm{p}}_x \cdot \|\tau\| && \text{Rmk. 25} \\
&= \|s\| + [|s|^{\mathrm{p}}_x + |s|^{\mathrm{p}}_y] + (|s|^{\mathrm{p}}_x + |s|^{\mathrm{p}}_y) \cdot \|\tau\| && \text{L. 26} \\
&> \|s\| + [|s|^{\mathrm{p}}_x] + |s|^{\mathrm{p}}_x \cdot \|\rho\| + [|s|^{\mathrm{p}}_y + |s|^{\mathrm{p}}_x \cdot |\rho|^{\mathrm{p}}_y] + (|s|^{\mathrm{p}}_y + |s|^{\mathrm{p}}_x \cdot |\rho|^{\mathrm{p}}_y) \cdot \|\sigma\| && \text{L. 27} \\
&= \|s\| + [|s|^{\mathrm{p}}_x] + |s|^{\mathrm{p}}_x \cdot \|\rho\| + [|s|^{\mathrm{p}}_y + \max\{1, |s|^{\mathrm{p}}_x\} \cdot |\rho|^{\mathrm{p}}_y] + (|s|^{\mathrm{p}}_y + |s|^{\mathrm{p}}_x \cdot |\rho|^{\mathrm{p}}_y) \cdot \|\sigma\| && \text{Rmk. 25} \\
&= \|s\| + [|s|^{\mathrm{p}}_x] + |s|^{\mathrm{p}}_x \cdot \|\rho\| + [|s[x \leftarrow \rho]|^{\mathrm{p}}_y] + (|s|^{\mathrm{p}}_y + \max\{1, |s|^{\mathrm{p}}_x\} \cdot |\rho|^{\mathrm{p}}_y) \cdot \|\sigma\| && \text{Rmk. 25} \\
&= \|s\| + [|s|^{\mathrm{p}}_x] + \max\{1, |s|^{\mathrm{p}}_x\} \cdot \|\rho\| + [|s[x \leftarrow \rho]|^{\mathrm{p}}_y] + \max\{1, |s[x \leftarrow \rho]|^{\mathrm{p}}_y\} \cdot \|\sigma\| && \text{Rmk. 25} \\
&= \|s[x \leftarrow \rho]\| + [|s[x \leftarrow \rho]|^{\mathrm{p}}_y] + \max\{1, |s[x \leftarrow \rho]|^{\mathrm{p}}_y\} \cdot \|\sigma\| \\
&= \|t'\|
\end{aligned}$$

- The other cases smoothly follow from the induction hypothesis. ◀

▶ **Proposition 15.** *The length of a (non-deterministic) reduction sequence $s \twoheadrightarrow_{\mathsf{cc}} t$ is:*

1. *without algebraic laws, exactly* $|s| - |t|$*;*
2. *with only* redundancy*, at most* $|s| - |t|$*;*
3. *with only* duplicability*, at least* $|s| - |t|$.

Proof. Recall the definition of reduction weight $|t|$ on page 10, where $x \notin \mathsf{fv}(r)$ but $x \in \mathsf{fv}(s)$. We consider each case; for the last two, observe that $|t|$ is always positive.

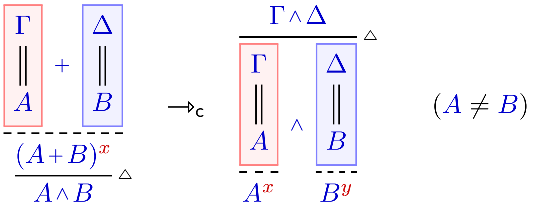
1. Each rewrite step reduces $|t|$ by exactly one: a step $(\lambda x.r)\tau \to_{\mathsf{b}} r[x \leftarrow \tau]$ where $x \in \mathsf{fv}(r)$ replaces an abstraction and application (weight zero) by a sharing (weight -1), and $(\lambda x.s)\tau \to_{\mathsf{b}} s[x \leftarrow \tau]$ where $x \in \mathsf{fv}(s)$ replaces weight 2 by weight 1; a c-step introduces a sharing of weight -1; an e-step removes a sharing (weight -1) and a variable (weight 1); and a d-step removes a weakening (weight 1).
2. By 1. above, and the observation that if $\tau \leq \sigma$ then $|\tau| \geq |\sigma|$.
3. By 1. above, and the observation that if $\tau \leq \sigma$ then $|\tau| \leq |\sigma|$. ◀

D Omitted proofs and lemmas in Section 5

▶ **Theorem 17.** *A typed CC-term is strongly normalizing.*

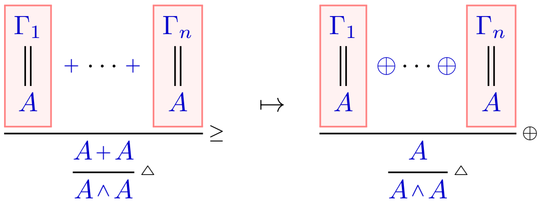
Proof sketch. We reduce the problem to typed normalization in a non-deterministic (or probabilistic) λ-calculus, which are known to terminate (see e.g. [17]).

First, the *copy* rewrite rule is *linear* for contractions on different types, since no idempotence law ($\geq$) applies. This corresponds to the following transformation on derivations:



The transformation is applied throughout a proof, through abstractions and applications (which are split, by Currying, into multiple), and is *linear* and so *terminating*. The result is a derivation for a CC-term where every variable has a uniform type. A contraction cannot be pushed past a ($\geq$) instance of idempotence on the same type. This has the form below left.

We translate the remaining derivation into one for a simply-typed non-deterministic λ-calculus, with a regular contraction rule and a type operator $\oplus$ with a co-diagonal rule from $A \oplus A$ to A. The result is below right.



This derivation is for a typed, non-deterministic λ-term, which is strongly normalizing, and simulates reduction in both CC-terms. Consequently, these are strongly normalizing. ◀

E Omitted proofs and lemmas in Section 6

▶ **Theorem 22.** *A structural λ-term t is weakly [strongly] normalizing if and only if there is a typed, [strong,] uniform* CC*-term $u \succ t$.*

Proof sketch. $\Rightarrow$ Standard: normal forms can be typed, and subject expansion holds.
$\Leftarrow$ By Theorem 17 the typed CC-term s is strongly normalizing; hence t is strongly normalizing if weakened terms are typed, and weakly normalizing if weakened terms remain untyped. ◀

Categorifying Non-Idempotent Intersection Types

Giulio Guerrieri
University of Bath, Department of Computer Science, Bath, UK

Federico Olimpieri
Institut de Mathématiques de Marseille (I2M), Aix-Marseille Université, Marseille, France

Abstract

Non-idempotent intersection types can be seen as a syntactic presentation of a well-known denotational semantics for the lambda-calculus, the category of sets and relations. Building on previous work, we present a categorification of this line of thought in the framework of the bang calculus, an untyped version of Levy's call-by-push-value. We define a bicategorical model for the bang calculus, whose syntactic counterpart is a suitable category of types. In the framework of distributors, we introduce intersection type distributors, a bicategorical proof relevant refinement of relational semantics. Finally, we prove that intersection type distributors characterize normalization at depth 0.

2012 ACM Subject Classification Theory of computation → Lambda calculus; Theory of computation → Linear logic; Theory of computation → Categorical semantics

Keywords and phrases Linear logic, bang calculus, non-idempotent intersection types, distributors, relational semantics, combinatorial species, symmetric sequences, bicategory, categorification

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.25

Funding This work is partially supported by EPSRC Project EP/R029121/1 *Typed lambda-calculi with sharing and unsharing.*

Acknowledgements The authors thank Lionel Vaux Auclair for insightful discussions and comments.

1 Introduction

Since Girard's introduction of *linear logic* [32], the notion of linearity has played a central role in the Logic-in-Computer-Science community. A program is linear when it uses its inputs only *once* during computation (inputs cannot be copied or deleted); while a non-linear program may call its inputs at will. Via the exponential modalities ! and ?, linear logic gives a logical status to the operations of erasing and copying data.

Another way to study linearity is provided by some type systems. *Intersection types* were introduced by Coppo and Dezani [14, 15] as an extension of simple types by means of the (associative, commutative and idempotent) intersection connective $a \cap b$: a term of type $a \cap b$ can be seen as a program of both type a and type b. This kind of type systems have proven to be very useful to characterize various notion of normalization in the λ-calculus [37]. If we impose *non-idempotency* to the intersection [31, 16] (i.e. $a \cap a \neq a$), we get a "resource-sensitive" intersection type system, in the sense that the arrow type encodes the *exact* number of times that a term needs its input during computation: intuitively, a term typed $a \cap a \cap b$ can be used twice as a program of type a and once as a program of type b. Non-idempotent intersection types allow *combinatorial* characterization of normalization properties and of the execution time of programs [9, 16, 4] and proof-nets [19, 20]. Also, De Carvalho's non-idempotent intersection type system $\mathcal{R}$ is a syntactic presentation of the categorical semantics of λ-calculus given in the category of sets and relations [16, 17]. There is a strong connection between linear logic and non-idempotent intersection types [18].

Inspired by [34, 41, 48, 43], we propose here a *categorification* of this kind of semantics. Roughly, categorification consists in replacing set-theoretic notions with category-theoretic ones. In general, this process gives both more fine-grained structures and general points of

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 25; pp. 25:1–25:24
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

view. Melliès and Zeilberger [42] followed this approach to present a categorical definition of what a type system is: a type system is a *functor* between a category of type derivations and a category of terms. Since we are interested in categorical semantics with an intersection type presentation, the first natural thing to do is replacing the category of sets and relations with the bicategory of *distributors* [6, 10]. Distributor-induced semantics of programming languages were already presented in [12, 27]. In particular, Fiore, Gambino, Hyland and Winskel introduced the bicategory of *generalized species of structure* [27], a very rich framework that generalizes both relational semantics and Joyal's *combinatorial species* [35, 27, 30, 48]. As shown in [12, 29], distributors can also lead to a generalization of Scott's semantics.

Mazza, Pellissier and Vial [41], inspired by [42] and Hyland's project of categorification of the theory of the λ-calculus [34], presented a general approach to intersection types rooted in the notion of multicategory. In their framework, the λ-calculus is seen as a 2-operad, where 2-cells consist of reduction paths. Intersection type systems are seen as a special kind of *fibrations.* Via a Grothendieck construction, with these fibrations they associate an *approximation presheaf* that interprets terms as *discrete distributors.* Thanks to this categorical approach, they are able to prove a parametric normalization theorem for a class of intersection type systems in a modular and elegant way. Their method relies on a Curry-Howard style correspondence between intersection type derivations and a kind of λ-terms *approximants*, the *polyadic terms.* However, their approach does not provide a denotational model and it does not support subtyping for intersection types. This latter feature is strictly linked to the fact that approximation presheaves action on types is restricted to discrete categories [41]. It is then natural to ask what happens when we take the standpoint of denotational semantics and we take into account categories with non-trivial morphisms.

Recently, Tsukada, Asada and Ong [48, 49] presented the *rigid Taylor expansion*[1] semantics for an η-expanded fragment of non-deterministic simply-typed λ-calculus with fixed point combinator, then extended to probabilistic and quantum computation: the linear approximants are still polyadic terms. They proved that this semantics is naturally isomorphic to the generalized species semantics. This time, the standpoint is the one of denotational semantics and distributors ranges over groupoids, but subtyping is not taken into account. The groupoid structure of the model gives the possibility to define an *action* of type isomorphisms on polyadic terms. A quotient induced by this action guarantees the preservation, up to isomorphism, of the semantics under reduction. Concretely one has that $[\![M]\!] \cong [\![N]\!]$ whenever $M \to N$ and the natural isomorphism is given by *reduction* of polyadic terms.

Inspired by these lines of thought, Olimpieri [43] introduced *intersection type distributors*, a categorified version of intersection type disciplines, where subtyping and denotational semantics are both taken into account. Intersection type distributors are a *syntactic presentation* of bicategorical denotational semantics for the λ-calculus given by Kleisli bicategories of distributors for suitable pseudomonads. Each pseudomonad taken into account gives rise to a notion of intersection type, with specific resource behavior[2]. The semantics obtained by this method is *proof relevant*: given a term M, a type context Δ and a type a, we set $\mathsf{T}_U(M)(\Delta, a) = \left\{ \begin{array}{c} \tilde{\pi} \\ \vdots \\ \Delta \vdash M : a \end{array} \mid \pi \text{ is a type derivation for } M \right\}$ where $\mathsf{T}_U(M)$ is the intersection type distributor that interprets M in an appropriate category U of types, and

[1] The rigid Taylor expansion is a deterministic variant of Ehrhard and Regnier's Taylor expansion [25, 26].

[2] It is worth noting that this new semantic setting is not a special case of [41], as standard polyadic terms fail dramatically subject reduction for intersection type distributors. The failure of subject reduction happens because standard polyadic terms [48, 41] cannot encode all the *qualitative* information produced by the *subtyping* feature of intersection type distributors. A counterexample is in Appendix A.

$\tilde{\pi}$ is an equivalence class of derivations. The equivalence relation on derivations is induced by the composition of distributors, which generalizes the quotient of [48]. We have that, if $M \to N$, then $\mathsf{T}_U(M) \cong \mathsf{T}_U(N)$. Categorification then allows us to pass from a semantics of *types* to a semantics of *derivations*. Note that, in our setting, the semantics of a term M associates with every type context Δ and type a the set of derivations for M with conclusion $\Delta \vdash M : a$; while more coarse-grained models such as relational semantics can only say if *there is* a type derivation for M with conclusion $\Delta \vdash M : a$.

In the present paper, we introduce *non-idempotent* intersection type distributors in an untyped call-by-push-value setting [33, 24, 39, 47], the *bang calculus*. Levy's call-by-push-value paradigm subsumes call-by-name (CbN) and call-by-value (CbV), from both the operational and denotational semantics standpoints [39, 33]. In this respect, our work is more general than [43] (which considers only the CbN λ-calculus). Moreover, inspired by linear logic, the bang calculus internalizes in the syntax the !-operator, which semantically corresponds to the monadic operator to handle resources. In this way, it is more natural to link syntax and semantics and to disentangle our investigation from the evaluation mechanism. Here we focus on a particular monadic construction (the symmetric strict monoidal completion, see Section 2) and we do not extend the more general and abstract method of [43] to the bang calculus because in this way we can avoid introducing too much categorical background.

Our categorical approach allows the introduction of a suitable *category of types*, where morphisms between types are a generalization of *subtyping*. Given a type morphism $a' \to a$, the intuition is that the type a' somehow *refines* the type a. We prove that non-idempotent intersection type distributors characterize normalization at depth 0 in the bang calculus. Normalization at depth 0 in the bang calculus is a notion that encompasses both CbN solvability [2, 37] and CbV potential valuability [45, 11]. The argument to prove this result is combinatorial and standard (similar results for the bang calculus are proved in [24, 8] using relational semantics), but thanks to the categorified setting we gain a much more fine-grained understanding of the dynamics of type derivations under reduction. Indeed, in our setting, subject reduction and expansion (Theorem 12) clearly open the possibility to define an explicit *deterministic* reduction relation on (equivalent classes) of type derivations, but the investigation of this line of thought is left to future work. We just notice that the substitution operation on type derivations is strictly linked to morphism composition, respecting the basic intuition of categorical semantics: substitution corresponds to composition.

Outline. Some preliminaries are in Section 2. Section 3 shows how the category of distributors Dist can be seen as a generalization of the categories Rel of sets and relations and Polr of preorders. In Section 4 we define a proof-relevant denotational model of the bang calculus in Dist as a generalization of non-idempotent intersection type systems and we prove a semantic characterization of depth 0 normalization in the bang calculus. Section 5 concludes. In Appendix A we recall some basic notions for bicategories and coends, we prove Lemma 11 and we show the failure of subject reduction with subtyping for polyadic terms.

2 Preliminaries

The bang calculus. The syntax and operational semantics of the *bang calculus* [33] are defined in Figure 1.[3] Terms are built up from a countably infinite set of *variables* (denoted by $x, y, z, \dots$). Terms of the form $S^!$ (resp. $\lambda x.S$; ST) are called *boxes* (resp. *abstractions*;

[3] Syntax and reduction rule of the bang calculus are presented as in [33], which are slightly different from [24]. But unlike [33] (and akin to [46]), here we do not use der as a primitive, since der and its associated rule $\mathsf{der}(S^!) \mapsto_{\mathrm{d}} S$ can be simulated in our setting by defining $\mathsf{der} = \lambda x.x$, because $(\lambda x.x)S^! \mapsto_{\mathrm{b}} S$.

$$
\begin{array}{rll}
\textit{Terms}: & S,T,U ::= x \mid \lambda x.S \mid ST \mid S^{!} & (\text{set: } !\Lambda) \\
\textit{Contexts}: & \mathtt{C} ::= [\cdot] \mid \lambda x.\mathtt{C} \mid \mathtt{C}S \mid S\mathtt{C} \mid \mathtt{C}^{!} & (\text{set: } !\Lambda_{\mathtt{C}}) \\
\textit{Ground Contexts}: & \mathtt{G} ::= [\cdot] \mid \lambda x.\mathtt{G} \mid \mathtt{G}S \mid S\mathtt{G} & (\text{set: } !\Lambda_{\mathtt{G}}) \\
\textit{Root-step}: & (\lambda x.S)T^{!} \mapsto_{\mathrm{b}} S\{T/x\} & \\
\to_{\mathrm{b}}\textit{-reduction}: & S \to_{\mathrm{b}} T \;\Leftrightarrow\; \exists\, \mathtt{C} \in !\Lambda_{\mathtt{C}},\, \exists\, S', T' \in !\Lambda : S = \mathtt{C}[S'],\, T = \mathtt{C}[T'],\, S' \mapsto_{\ell} T' & \\
\to_{\mathrm{b_g}}\textit{-reduction}: & S \to_{\mathrm{b_g}} T \;\Leftrightarrow\; \exists\, \mathtt{G} \in !\Lambda_{\mathtt{G}},\, \exists\, S', T' \in !\Lambda : S = \mathtt{G}[S'],\, T = \mathtt{G}[T'],\, S' \mapsto_{\ell} T' &
\end{array}
$$

Figure 1 The bang calculus: its syntax and reduction rules.

(linear) applications). The set of boxes is denoted by $!\Lambda_{!}$. The set of free variables of a term S, denoted by $\mathsf{fv}(S)$, is defined as expected, λ being the only binding construct. All terms are considered up to α-conversion. Given $S, T \in !\Lambda$ and a variable x, $S\{T/x\}$ denotes the term obtained by the *capture-avoiding substitution* of T for each free occurrence of x in S.

Contexts $\mathtt{C}$ and (with exactly one hole $[\cdot]$) are defined in Figure 1. We write $\mathtt{C}[S]$ for the term obtained by capture-allowing substitution of the term S for the hole $[\cdot]$ in the context $\mathtt{C}$. *Ground contexts* $\mathtt{G}$ are the restriction to contexts where the hole is not inside any !.

The *bang calculus* is the set $!\Lambda$ endowed with reduction $\to_{\mathrm{b}}$ (Figure 1), which is confluent [33]. Intuitively in the root-step $\mapsto_{\mathrm{b}}$ the box-construct ! marks the only terms that can be erased and duplicated: a *β-like redex* $(\lambda x.S)T$ can be fired only when its argument is a box, *i.e.* $T = U^{!}$: if it is so, the content U of the box T replaces any free occurrence of x in S.

Reduction $\to_{\mathrm{b_g}} \subseteq \to_{\mathrm{b}}$ is said *at depth 0* and defined as the closure of $\mapsto_{\mathrm{b}}$ under ground contexts (see Figure 1): it does not reduce inside boxes. It has the diamond-property [33].

▶ **Example 1.** Let $\Delta = \lambda x.xx^{!}$. Then $\Delta\Delta^{!} \to_{\mathrm{b_g}} \Delta\Delta^{!} \to_{\mathrm{b_g}} \dots$ (and so $\Delta\Delta^{!} \to_{\mathrm{b}} \Delta\Delta^{!} \to_{\mathrm{b}} \dots$).

▶ **Definition 2** (Clash). *A* clash *is a term of the form $S^{!}T$ or $T(\lambda x.S)$.*

Let $S \in !\Lambda$: S is clash-free *if and only if it contains no clash; S is* clash-free at depth 0 *if and only if each clash occurring in S is under the scope of a !.*

For instance, $(\lambda z.x)(x^{!}y)^{!}$ is clash-free at depth 0 but not clash-free. Roughly, a clash is a “meaningless” term that cannot inherently be typed (see [24, 8]): boxes cannot be applied, abstractions cannot be the argument of an application.

The bang calculus can be extended (see [24]) with the reduction $\to_{\sigma} = \to_{\sigma_1} \cup \to_{\sigma_2} \cup \to_{\sigma_3}$ where $\to_{\sigma_1}$, $\to_{\sigma_2}$ and $\to_{\sigma_3}$ are the contextual closure of the following rules, respectively:

$$(\lambda x.S)TU \mapsto_{\sigma_1} (\lambda x.SU)T \qquad (\lambda y.\lambda x.S)T \mapsto_{\sigma_2} \lambda x.(\lambda y.S)T \qquad U((\lambda x.S)T) \mapsto_{\sigma_3} (\lambda x.US)T$$

with $x \notin \mathsf{fv}(U)$ in $\mapsto_{\sigma_1}$ and $\mapsto_{\sigma_3}$, while $x \notin \mathsf{fv}(T) \cup \{y\}$ in $\mapsto_{\sigma_2}$. We set $\to_{\mathrm{b}\sigma} = \to_{\mathrm{b}} \cup \to_{\sigma}$ and $\to_{\mathrm{b}\sigma_{\mathrm{g}}} = \to_{\mathrm{b_g}} \cup \to_{\sigma_{\mathrm{g}}}$, where $\to_{\sigma_{\mathrm{g}}} = \to_{\sigma_{1_{\mathrm{g}}}} \cup \to_{\sigma_{2_{\mathrm{g}}}} \cup \to_{\sigma_{3_{\mathrm{g}}}}$ and $\to_{\sigma_{i_{\mathrm{g}}}}$ is the closure under ground contexts of $\mapsto_{\sigma_i}$, for $i \in \{1,2,3\}$. Reductions $\to_{\sigma}$ and $\to_{\sigma_{\mathrm{g}}}$ are strongly normalizing [24] and can “unveil” hidden b-redexes and hidden clashes. For instance,

$$((\lambda x.\Delta)x)\Delta^{!} \to_{\sigma_{1_{\mathrm{g}}}} (\lambda x.\Delta\Delta^{!})x \qquad\qquad x((\lambda y.\lambda x.z)y) \to_{\sigma_{2_{\mathrm{g}}}} x(\lambda x.(\lambda y.z)y)$$

where $((\lambda x.\Delta)x)\Delta^{!}$ is b-normal but $(\lambda x.\Delta\Delta^{!})x$ is not ($\to_{\mathrm{b_g}}$ can fire the b-redex $\Delta\Delta^{!}$), and $x((\lambda y.\lambda x.z)y)$ is clash-free but $x(\lambda x.(\lambda y.z)y)$ is not (not even at depth 0).

Integers and Permutations. For $n \in \mathbb{N}$, we set $[n] = \{1, \ldots, n\}$, so $[0] = \emptyset$. The set of permutations over $[n]$ is denoted by S_n. We define the category $\mathbb{P}$ of integers and permutations:

- the objects of $\mathbb{P}$ are $\mathrm{ob}(\mathbb{P}) = \{[n] \mid n \in \mathbb{N}\}$; the identity on $[n]$ is denoted by 1_n;
- the homset from $[n]$ to $[m]$ is $\mathbb{P}[[n],[m]] = \begin{cases} S_n & \text{if } n = m \\ \emptyset & \text{otherwise;} \end{cases}$
- the category $\mathbb{P}$ is symmetric strict monoidal, with tensor product given by addition: $[n] \oplus [m] = [n+m]$. Given $\sigma \in S_{k_1}$ and $\tau \in S_{k_2}$, we define $\sigma \oplus \tau \in S_{k_1+k_2}$ as

$$(\sigma \oplus \tau)(i) = \begin{cases} \sigma(i) & \text{if } 1 \leq i \leq k_1 \\ \tau(i-k_1)+k_1 & \text{otherwise.} \end{cases}$$

Given $k_1, \ldots, k_n \in \mathbb{N}$ and $\sigma \in S_n$, we define $\bar{\sigma} \colon [\sum_{i \in [n]} k_i] \to [\sum_{i \in [n]} k_{\sigma(i)}]$ as $\bar{\sigma}(\sum_{r=1}^{l-1} k_r + p) = \sum_{r=1}^{l-1} k_{\sigma(r)} + p$, where $l \in [n]$ and $1 \leq p \leq k_{\sigma(l)}$.

Symmetric strict monoidal completion. For a list $\vec{a} = \langle a_1, \ldots, a_k \rangle$, we set $\mathsf{len}(\vec{a}) = k$. Lists are denoted by $\vec{a}, \vec{b}, \vec{c} \ldots$, *concatenation* of two lists $\vec{a}$ and $\vec{b}$ is denoted by $\vec{a} \oplus \vec{b}$.

Let A be a small category. For each object $a \in \mathrm{ob}(A)$, the identity morphism on a is denoted by 1_a. The *symmetric strict monoidal completion* $!A$ of A is the category where:

- $\mathrm{ob}(!A) = \{\langle a_1, \ldots, a_n \rangle \mid a_i \in A \text{ and } n \in \mathbb{N}\}$;
- $!A[\langle a_1, \ldots, a_n \rangle, \langle a'_1, \ldots, a'_{n'} \rangle] = \begin{cases} \{\langle \sigma, f_1, \ldots, f_n \rangle \mid f_i : a_i \to a'_{\sigma(i)}\,,\ \sigma \in S_n\} & \text{if } n = n'; \\ \emptyset & \text{otherwise;} \end{cases}$
- for $\vec{a} = \langle a_1, \ldots, a_n \rangle \in \mathrm{ob}(!A)$, the identity on $\vec{a}$ is $1_{\vec{a}} = \langle 1_n, 1_{a_1}, \ldots, 1_{a_n} \rangle$;
- for $f = \langle \sigma, f_1, \ldots, f_n \rangle \colon \vec{a} \to \vec{b}$ and $g = \langle \tau, g_1, \ldots, g_n \rangle \colon \vec{b} \to \vec{c}$, the composition is $g \circ f = \langle \tau\sigma, g_{\sigma(1)} \circ f_1, \ldots, g_{\sigma(n)} \circ f_n \rangle$;
- the monoidal structure is given by list concatenation. The tensor product is symmetric, with symmetries given by the morphisms of the shape (where $\sigma \colon [n] \to [n]$ is a permutation)

$$\langle \sigma, \vec{1} \rangle \colon \langle a_1, \ldots, a_n \rangle \to \langle a_{\sigma(1)}, \ldots, a_{\sigma(n)} \rangle$$

Given a permutation $\sigma \colon [n] \to [n]$ and $\vec{a}_1, \ldots, \vec{a}_n \in \mathrm{ob}(!A)$ with $\mathsf{len}(\vec{a}_i) = k_i$ we define $\sigma^\star \colon \bigoplus_{i=1}^n \vec{a}_i \to \bigoplus_{i=1}^n \vec{a}_{\sigma(i)}$ as $\langle \bar{\sigma}, 1_{a_1}, \ldots, 1_{a_k} \rangle$, where $k = \sum_{i \in [n]} k_i$.

We use the following shortenings: $!A^n = (!A)^n$ and $!A^{\mathrm{op}} = (!A)^{\mathrm{op}}$.

Bicategory. We assume the reader to be familiar with bicategories [3, 6] and two-dimensional monads [5]. Some basic notions are briefly recalled in Appendix A. For a diagram $F \colon C \to D$, its colimit is denoted by $\varinjlim_{c \in C} F(c)$. Given a bicategory $\mathcal{C}$, $\mathcal{C}^{\mathrm{op}}$ is the bicategory obtained by reversing the 1-cells of $\mathcal{C}$, but not the 2-cells.

3 Rel, Polr, Dist

We sketch the structure of some categories providing denotational models of linear logic. We use linear logic notations for cartesian products, comonads modelling exponentials, etc.

Rel. A simple model of linear logic is the category Rel of sets and relations. It is a prototype of *quantitative* semantics: the interpretation of a program gives information about its resource consumption during computation. Intuitively, linear logic formulas are interpreted by sets, linear logic proofs by relations, and an element in a set represents a non-idempotent intersection type. For the bang calculus, this model has been studied in [24, 33].

Objects of Rel are sets, and morphisms of Rel are binary relations. Identities are diagonal relations. Composition of morphisms in Rel is the usual composition of relations

$$g \circ f = \{\langle x, z\rangle \mid \exists\, y \in Y : \langle x, y\rangle \in f,\ \langle y, z\rangle \in g\} \text{ for } f \subseteq X \times Y \text{ and } g \subseteq Y \times Z.$$

For $X_1, X_2 \in \mathrm{ob}(\mathrm{Rel})$, the cartesian product $X_1 \mathbin{\&} X_2$ in Rel is the disjoint union of sets $X_1 \sqcup X_2 = (\{1\} \times X_1) \cup (\{2\} \times X_2)$, where projections $\pi_i \colon X_1 \mathbin{\&} X_2 \to X_i$ (for $i \in \{1,2\}$) are injections $\{\langle\langle i, x\rangle, x\rangle \mid x \in X_i\}$, and the terminal (and initial) object $\top$ is the empty set $\emptyset$.

Rel is a *symmetric monoidal* category, where the tensor $X \otimes Y$ is the cartesian product of sets $X \times Y$ and its unit $\mathbf{1}$ is an arbitrary singleton set. It is *closed*, with $X \multimap Y = X \times Y$ and evaluation $\mathrm{ev}_{X,Y} \colon (X \multimap Y) \times X \to Y$ defined by $\{\langle\langle\langle x, y\rangle, x\rangle, y\rangle \mid x \in X, y \in Y\}$.

Rel comes with an *exponential comonad* $\langle !, \mathrm{der}, \mathrm{dig}\rangle$. The functor $!$ is given by $!X = \mathcal{M}_{\mathrm{f}}(X)$ (finite multisets over X) and, for a morphism $f \in \mathrm{Rel}[X, Y]$, $!f = \{\langle [x_1, \ldots, x_n], [y_1, \ldots, y_n]\rangle \mid n \in \mathbb{N},\ \langle x_1, y_1\rangle, \ldots, \langle x_n, y_n\rangle \in f\}$. Dereliction $\mathrm{der}_X \in \mathrm{Rel}[!X, X]$ is $\{\langle [x], x\rangle \mid x \in X\}$, and digging $\mathrm{dig}_X \in \mathrm{Rel}[!X, !!X]$ is $\{\langle m_1 + \cdots + m_k, [m_1, \ldots, m_k]\rangle \mid m_1, \ldots, m_k \in !X\}$ (for two finite multisets $\bar{a} = [a_1, \ldots, a_k]$ and $\bar{b} = [b_1, \ldots, b_n]$, we set $\bar{a} + \bar{b} = [a_1, \ldots, a_k, b_1, \ldots, b_n]$).

Polr. To work within a more informative setting, providing not only *quantitative*, but also *qualitative* information, consider the category Polr of preordered sets and monotonic relations [21, 23]. Intuitively, given two types a and b, if $a \leq b$ then a is an approximant of b. All the constructions in Polr are a refinement and generalization of the ones for Rel.

In Polr, objects are preordered sets; a morphism f from $\mathcal{X} = \langle |\mathcal{X}|, \leq_{\mathcal{X}}\rangle$ to $\mathcal{Y} = \langle |\mathcal{Y}|, \leq_{\mathcal{Y}}\rangle$ is a *monotonic* relation[4] from $|\mathcal{X}|$ to $|\mathcal{Y}|$, *i.e.*, if $\langle x, y\rangle \in f$ with $x' \leq_{\mathcal{X}} x$ and $y \leq_{\mathcal{Y}} y'$ then $\langle x', y'\rangle \in f$. The identity at $\mathcal{X}$ is $\{\langle x, x'\rangle \mid x \leq_{\mathcal{X}} x'\}$. Composition preserves monotonicity.

In Polr the cartesian product $\mathcal{X}_1 \mathbin{\&} \mathcal{X}_2$ is the disjoint union of sets $|\mathcal{X}_1| \sqcup |\mathcal{X}_2|$ with the preorder $\leq_{\mathcal{X}_1} \sqcup \leq_{\mathcal{X}_2}$ defined as $\langle i, x\rangle \leq_{\mathcal{X}_1 \& \mathcal{X}_2} \langle j, y\rangle$ if $i = j$ and $x \leq_{\mathcal{X}_i} y$. The terminal object $\top$ is $\emptyset$ with the empty order. Projections $\pi_i \colon \mathcal{X}_1 \mathbin{\&} \mathcal{X}_2 \to \mathcal{X}_i$ are $\pi_i = \{\langle\langle i, x\rangle, x'\rangle \mid x \leq_{\mathcal{X}_i} x'\}$.

Polr has a symmetric monoidal structure. The tensor $\mathcal{X}_1 \otimes \mathcal{X}_2$ is the cartesian product of sets with the product order. The endofunctor $\mathcal{X} \otimes \_$ admits a right adjoint $\_ \multimap \mathcal{Y}$ defined as follows: $|\mathcal{X} \multimap \mathcal{Y}| = |\mathcal{X}| \times |\mathcal{Y}|$ and $\langle x, y\rangle \leq_{\mathcal{X} \multimap \mathcal{Y}} \langle x', y'\rangle$ if $x' \leq_{\mathcal{X}} x$ and $y \leq_{\mathcal{Y}} y'$. The evaluation morphism $\mathrm{ev}_{\mathcal{X}_1, \mathcal{X}_2} \colon (\mathcal{X}_1 \multimap \mathcal{X}_2) \mathbin{\&} \mathcal{X}_1 \to \mathcal{X}_2$ is $\{\langle\langle\langle x, y\rangle, x'\rangle, y'\rangle \mid x' \leq x,\ y \leq y'\}$.

Polr has exponential comonad $\langle !, \mathrm{der}, \mathrm{dig}\rangle$.[5] The endofunctor $! \colon \mathrm{Polr} \to \mathrm{Polr}$ is given by $!\mathcal{X} = \langle \mathcal{M}_{\mathrm{f}}(|\mathcal{X}|), \leq_{\mathcal{X}}\rangle$ with $[x_1, \ldots, x_n] \leq_{!\mathcal{X}} [x'_1, \ldots, x'_{n'}]$ if $n = n'$ and there is $\sigma \in S_n$ such that $x_i \leq x'_{\sigma(i)}$ for all $1 \leq i \leq n$; for $f \in \mathrm{Polr}[\mathcal{X}, \mathcal{Y}]$, we set $!f = \{\langle [x_1, \ldots, x_n], [y_1, \ldots, y_k]\rangle \mid \langle x_i, y_i\rangle \in f,\ k \in \mathbb{N}\}$. Dereliction $\mathrm{der}_{\mathcal{X}} \colon !\mathcal{X} \to \mathcal{X}$ is $\{\langle [x], x'\rangle \mid x \leq_{\mathcal{X}} x'\}$, and digging $\mathrm{dig}_{\mathcal{X}} \colon !\mathcal{X} \to !!\mathcal{X}$ is $\{\langle m, [m_1, \ldots, m_k]\rangle \mid m \leq_{!\mathcal{X}} m_1 + \cdots + m_k\}$.

Rel is the full subcategory of Polr where objects are sets equipped with the discrete order.

Polr as a model of the bang calculus. A categorical model of the bang calculus [23, 24] consists of a $\star$-autonomous category $(A, \otimes, I, \multimap, (-)^{\perp})$, cartesian with product & and terminal object $\top$ (and, by $\star$-autonomy, cocartesian with coproduct $\oplus$ and initial object 0), endowed with a comonad $(!, \mathrm{der}, \mathrm{dig})$ with suitable Seely isomorphisms [23, 33]. Also, we

[4] In [21, 23], monotonicity is slightly different, so that the type system generated by the model is covariant on the left of $\vdash$ and contravariant on the right of $\vdash$. With our definition, the type system generated by the model is contravariant on the left of $\vdash$ and covariant on the right of $\vdash$, in accordance with [1].

[5] Akin to [21] and unlike [23], our exponential comonad is based on finite multiset construction. But our preorder on $!\mathcal{X}$ is different from [21]: there $[a] \leq_{!\mathcal{X}} [a, a]$ (idempotency is a sort of approximation), here $[a]$ and $[a, a]$ are incomparable, so that approximation is completely independent from idempotence.

Types:

$$a := x \in \mathcal{X} \mid [a_1, \ldots, a_k] \multimap a \mid [a_1, \ldots, a_k]$$

Preorder $\leq_U$ in U:

$$\frac{x \leq_{\mathcal{X}} x'}{x \leq_U x'} \qquad \frac{m' \leq_U m \qquad a \leq_U a'}{(m \multimap a) \leq_U (m' \multimap a')}$$

$$\frac{\sigma \in S_k \quad a_1 \leq_U a'_{\sigma(1)} \quad \overset{k \in \mathbb{N}}{\cdots} \quad a_k \leq_U a'_{\sigma(k)}}{[a_1, \ldots, a_k] \leq_U [a'_1, \ldots, a'_k]}$$

Derivation rules:

$$\frac{a' \leq_U a}{x_1 : [\,], \ldots, x_i : [a'], \ldots x_n : [\,] \vdash x_i : a}$$

$$\frac{\Gamma \vdash S : m \multimap a \qquad \Gamma' \vdash T : m \qquad \Delta \leq_{U^n} \Gamma \otimes \Gamma'}{\Delta \vdash ST : a}$$

$$\frac{\Gamma_1 \vdash S : a_1 \quad \overset{k \in \mathbb{N}}{\cdots} \quad \Gamma_k \vdash S : a_k \quad \Delta \leq_{U^n} \bigotimes_{i=1}^k \Gamma_i}{\bigotimes_{i=1}^k \Gamma_i \vdash S^! : [a_1, \ldots, a_k]}$$

$$\frac{\Delta, x : m \vdash S : a}{\Delta \vdash \lambda x.S : m \multimap a}$$

Figure 2 Non-idempotent intersection type system $\mathcal{R}_\leq$ associated with the preorder U in Polr.

require that $0 \cong \top$. An *extensional model* of the bang calculus is then an object $U \in \mathrm{ob}(A)$ such that $U \cong\, !U \,\&\, (!U \multimap U)$. To have a *non-extensional model* for the bang calculus a retraction $!U \,\&\, (!U \multimap U) \lhd U$ is enough.

We build a retraction in the category Polr. We define a family of preoders as follows:

$$U_0 = \mathcal{X} \text{ (any preorder)} \qquad U_{n+1} = \,!U_n \sqcup ((!U_n \multimap U_n) \sqcup \mathcal{X}) \tag{1}$$

We define a family of canonical inclusions $(\iota_n : U_n \hookrightarrow U_{n+1})_{n \in \mathbb{N}}$ as $\iota_0 = \iota_{\mathcal{X}}$ (the inclusion $\mathcal{X} \hookrightarrow\, !\mathcal{X} \sqcup ((!\mathcal{X} \multimap \mathcal{X}) \sqcup \mathcal{X})$) and $\iota_{n+1} = \,!\iota_n \sqcup ((!\iota_n \multimap \iota_n) \sqcup 1_{\mathcal{X}})$, so the preorder U_n is just the restriction to the elements of U_n of the preorder U_{n+1}. We set $U = \varinjlim_{n \in \mathbb{N}} U_n$, that is a directed colimit of the directed diagram $\langle \iota_i \rangle_{i \in \mathbb{N}}$. It is easy to check that there exists a canonical inclusion $\iota : !U \sqcup (!U \multimap U) \hookrightarrow U$ and that we have a retraction $!U \,\&\, (!U \multimap U) \lhd U$.

We can define the interpretation of the terms of the bang calculus in Polr. Let $S \in\, !\Lambda$ and $\mathsf{fv}(S) \subseteq \vec{x} = \langle x_1, \ldots, x_n \rangle$ with the x_i's pairwise distinct. The *semantics* (or *denotation*) of S is a monotonic relation $[\![S]\!]_{\vec{x}} : !U^{\otimes n} \to U$ defined by induction as follows:

- $[\![x_i]\!]_{\vec{x}} = \{\langle\langle [\,], \ldots, [a'], \ldots, [\,] \rangle, a\rangle \mid a' \leq a\}$ ($[a']$ is in the i^{th} position in $\langle [\,], \ldots, [a'], \ldots, [\,] \rangle$);
- $[\![\lambda y.T]\!]_{\vec{x}} = \{\langle \Delta, \iota(\langle m, a \rangle)\rangle \mid \langle \Delta \oplus \langle m \rangle, a \rangle \in [\![T]\!]_{\vec{x} \oplus \langle y \rangle}\}$, where $y \notin \vec{x}$;
- $[\![ST]\!]_{\vec{x}} = \bigcup_{m \in !U} \bigcup_{\Gamma, \Gamma' \in U^n} \{\langle \Delta, a \rangle \mid \langle \Gamma, \iota(\langle m, a \rangle)\rangle \in [\![S]\!]_{\vec{x}}\,,\ \langle \Gamma', \iota(m) \rangle \in [\![T]\!]_{\vec{x}}$ and $\Delta \leq_{U^n} \Gamma \otimes \Gamma'\}$;
- $[\![T^!]\!]_{\vec{x}} = \bigcup_{k \in \mathbb{N}} \bigcup_{\Gamma_1, \ldots, \Gamma_k \in U^n} \{\langle \Delta, [a_1, \ldots, a_k] \rangle \mid \langle \Gamma_i, a_i \rangle \in [\![T]\!]_{\vec{x}}$ and $\Delta \leq_{U^n} \bigotimes_{i=1}^k \Gamma_i\}$ where if $\Gamma = \langle m_1, \ldots, m_n \rangle$ and $\Gamma' = \langle m'_1, \ldots, m'_n \rangle$ then $\Gamma \otimes \Gamma' = \langle m_1 + m'_1, \ldots, m_n + m'_n \rangle$.

Ehrhard [23] showed this is a denotational semantics. By setting $m \multimap a = \langle m, a \rangle \in\, !U \times U$, we can give a type-theoretic description of the preorder U as in Figure 2. Such a type system $\mathcal{R}_\leq$ is similar to de Carvalho's non-idempotent intersection type system $\mathcal{R}$ [16, 17]. The main difference is that in $\mathcal{R}_\leq$ types are elements of a *preorder* U (an object of Polr), while in $\mathcal{R}$ types are elements of a *set* U (an object of Rel). The additional information provided by the preorder accounts for *approximation*: if $a \leq_U b$ then the type a approximates the type b. This is evident in the rule for the variable in Figure 2: a' can be seen as a *subtype* of a.

By easy inspection of the definition, $\langle \Delta, a \rangle \in [\![S]\!]_{\vec{x}}$ if and only if $\Delta \vdash S : a$. In other words, the semantics of a term S is the set of conclusions of the type derivations for S. The semantics is then a *semantics of types* in the non-idempotent intersection type system $\mathcal{R}_\leq$.

We now try to shift our standpoint. In system $\mathcal{R}_\leq$, let us try to define a *semantics of proofs*. Given a term S, a context Δ and type a, we set $[\![S]\!]_{\vec{x}}(\Delta, a) = \left\{ \begin{array}{c} \pi \\ \vdots \\ \Delta \vdash S : a \end{array} \middle| \, \pi \in \mathcal{R}_\leq \right\}$.

It is easy to see that this proof-relevant structure is not a denotational semantics (not even up to isomorphism). Indeed, reduction over type derivations in system $\mathcal{R}_{\leq}$ is non-deterministic, since it deals with multisets. Take $(\lambda z.(yz^!)z^!)S^! \to_{\mathrm{b_g}} (yS^!)S^!$ and the following type derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{y\colon [[a]\multimap[a]\multimap c] \vdash y\colon [a]\multimap[a]\multimap c} \quad \dfrac{\overline{z\colon[a]\vdash z\colon a}\,z_1}{z\colon[a]\vdash z^!\colon[a]}}{y\colon [[a]\multimap[a]\multimap c], z\colon[a]\vdash yz^!\colon[a]\multimap c} \quad \dfrac{\overline{z\colon[a]\vdash z\colon a}\,z_2}{z\colon[a]\vdash z^!\colon[a]}}{y\colon [[a]\multimap[a]\multimap c], z\colon[a,a]\vdash (yz^!)z^!\colon c}}{y\colon [[a]\multimap[a]\multimap c]\vdash \lambda z.(yz^!)z^!\colon[a,a]\multimap c} \quad \dfrac{\begin{matrix}\pi_1\\ \vdots\\ \Gamma_1\vdash S\colon a\end{matrix} \quad \begin{matrix}\pi_2\\ \vdots\\ \Gamma_2\vdash S\colon a\end{matrix}}{\Gamma_1\otimes\Gamma_2\vdash S^!\colon[a,a]}}{\Gamma_1\otimes\Gamma_2, y\colon [[a]\multimap[a]\multimap c]\vdash (\lambda z.(yz^!)z^!)S^!\colon c}$$

Suppose y is not free in S and $\pi_1 \neq \pi_2$ (*e.g.* take $S = wx^!$). Then if we consider the reduct $(yS^!)S^!$ we have two possible choices for the typing, $\pi\{\pi_1/z_1, \pi_2/z_2\}$ or $\pi\{\pi_2/z_1, \pi_1/z_2\}$. This non-determinism stems from the multiset structure, but we shall see that simply passing to a list-oriented framework does not solve the problem. A natural way to make this kind of structure a denotational semantics is the lifting to Set enriched distributors.

From Rel and Polr to Dist. We recall a basic but pivotal fact: a relation $f \subseteq X \times Y$ can be identified with its characteristic function $\chi_f\colon X \times Y \to \mathbf{2}$ where $\mathbf{2} = \{0,1\}$ is the two-element boolean algebra with sum (join) and product (meet). Composition is then defined as

$$\chi_{g\circ f}(x,z) = \sum_{y\in Y} \chi_g(y,z)\cdot\chi_f(x,y) \qquad \text{where } \chi_f\colon X\times Y\to\mathbf{2} \text{ and } \chi_g\colon Y\times Z\to\mathbf{2}\,. \tag{2}$$

All the constructions in Rel and Polr can be reformulated in this characteristic function perspective. For instance, in Rel, the identity at X becomes the characteristic function of X.

In Polr, a monotonic relation f from $\mathcal{X} = \langle|\mathcal{X}|, \leq_{\mathcal{X}}\rangle$ to $\mathcal{Y} = \langle|\mathcal{Y}|, \leq_{\mathcal{Y}}\rangle$ can be seen as a monotonic characteristic function $\chi_f\colon \mathcal{X}^{\mathrm{op}} \times \mathcal{Y} \to \mathbf{2}$, where $\mathcal{X}^{\mathrm{op}} = \langle|\mathcal{X}|, \geq_{\mathcal{X}}\rangle$ and $\mathbf{2}$ is endowed with the boolean order. Any preorder $\mathcal{X} = \langle|\mathcal{X}|, \leq_{\mathcal{X}}\rangle$ forms a category where $\mathrm{ob}(\mathcal{X}) = |\mathcal{X}|$ and $\mathcal{X}[x,x']$ is a singleton (if $x \leq_{\mathcal{X}} x'$) or the empty set (otherwise), so $\mathcal{X}^{\mathrm{op}}$ is the opposite category of $\mathcal{X}$. Thus, $\chi_f\colon \mathcal{X}^{\mathrm{op}} \times \mathcal{Y} \to \mathbf{2}$ is a bifunctor, contravariant in $\mathcal{X}$ and covariant in $\mathcal{Y}$. The semantics of a term S is then a Polr morphism $[\![S]\!]_{\vec{x}} : (!U^{\otimes n})^{\mathrm{op}} \times U \to \mathbf{2}$.

It is then natural to generalize the characteristic function viewpoint to generic categories, which gives rise to the notion of *distributor* (also known as *profunctors*).

Dist. For two small categories A, B, a *distributor* $F\colon A \nrightarrow B$ is a functor $F : A^{\mathrm{op}} \times B \to \mathrm{Set}$. Composition of distributors relies on the notion of *coend*, a kind of colimit (a coequalizer).

▶ **Definition 3** (Coend, [40]). *Let $F\colon C^{\mathrm{op}} \times C \to D$ be a functor. A* cowedge *for F is an object $T \in D$ together with a family of morphisms $w_c\colon F(c,c) \to T$ such that diagram* (3) *below commutes, for $f\colon c \to c'$. A* coend *for F, denoted by $\int^{c\in C} F(c,c)$, is a universal cowedge.*

$$\begin{tikzcd} F(c',c) \arrow[r, "F(f,1)"] \arrow[d, "F(1,f)"] & F(c,c) \arrow[d, "w_c"] \\ F(c',c') \arrow[r, "w_{c'}"] & T \end{tikzcd} \tag{3}$$

We now define the bicategory Dist of *distributors*. For a proper presentation of the structure of this bicategory we refer to [10, 12, 27, 30].

- *0-cells* are small categories $A, B, C \ldots$; *1-cells* $F\colon A \nrightarrow B$ are distributors, *i.e.* functors $F\colon A^{\mathrm{op}} \times B \to \mathrm{Set}$; *2-cells* $\alpha\colon F \Rightarrow G$ are natural transformations.
- Given any 0-cells A and B, 1-cells and 2-cells are organized as a category $\mathrm{Dist}(A, B)$. Composition $\alpha \star \beta$ in $\mathrm{Dist}(A, B)$ is called *vertical composition*. We define the *zero distributor* $\emptyset_{A,B} \in \mathrm{ob}(\mathrm{Dist}(A, B))$ as $\emptyset_{A,B}(a, b) = \emptyset$ for all $a \in \mathrm{ob}(A)$ and $b \in \mathrm{ob}(B)$.
- For $A \in \mathrm{Dist}$, the identity $1_A\colon A \nrightarrow A$ is Yoneda's embedding $1_A(a', a) = A[a', a]$.
- For 1-cells $F\colon A \nrightarrow B$ and $G\colon B \nrightarrow C$, their composition is given by

  $$(G \circ F)(a, c) = \int^{b \in B} G(b, c) \times F(a, b)$$

  Note the analogy with (2). Composition is only associative up to canonical isomorphisms. For this reason Dist is a bicategory [6].
- The cartesian product $A \,\&\, B$ is the disjoint union $A \sqcup B$ of categories. The terminal object $\top$ is given by the empty category. The bicategory Dist admits also coproducts, with $A \oplus B = A \sqcup B$ (the canonical inclusions are denoted by ι_A and ι_B) and $0 = \top$.
- There is a symmetric monoidal structure on Dist given by the cartesian product of categories: $A \otimes B = A \times B$, with any one-object category as a unit. The bicategory of distributors is monoidal closed, with linear exponential object $A \multimap B = A^{\mathrm{op}} \times B$.

The symmetric strict monoidal completion of a small category A (Section 2) lifts to an endofunctor in Cat, by setting $!F(\langle a_1, \ldots, a_n\rangle) = \langle F(a_1), \ldots, F(a_n)\rangle$ for any functor $F\colon A \to B$. The endofunctor $!$ can be extended to Dist, determining a pseudocomonad $(!, \mathrm{dig}_A, \mathrm{der}_A)$ on Dist [27, 30]. The two components of the pseudocomonad are defined as follows: $\mathrm{dig}_A(\vec{a}, \langle \vec{a_1}, \ldots, \vec{a_n}\rangle) = !A[\vec{a}, \bigoplus_{i=1}^{n} \vec{a_i}]$ and $\mathrm{der}_A(\vec{a}, a) = !A[\vec{a}, \langle a\rangle]$. The Kleisli bicategory $Kl(!)(\mathrm{Dist})$ is the bicategory of *categorical symmetric sequences* [30], biequivalent to the bicategory of *generalized species of structure* [27, 28]. There are Seely equivalences $!(A \,\&\, B) \simeq !A \times !B$ and $!\top \simeq 1$, pseudonatural in both A and B [27].

4 A Type-Theoretic Non-Extensional Model for the Bang Calculus

Distributors-Induced Model for the Bang Calculus. The bicategory of distributors fulfills a bicategorical generalization of the categorical model of the bang calculus shown in Section 3.[6] However, we leave the proper development of a general notion of bicategorical model for the bang calculus to future work, since the notion of symmetric monoidal bicategory is highly non-trivial. For our purpose, it is enough to present a denotational model inside a particular bicategory, *i.e.*, the bicategory of distributors. A denotational model in this setting will be an interpretation of bang terms as suitable 1-cells, such that $[\![S]\!]_{\vec{x}} \cong [\![T]\!]_{\vec{x}}$ if $S \to_\ell T$. In particular, we want $[\![S]\!]_{\vec{x}}\colon (!U^{\otimes n})^{\mathrm{op}} \times U \to \mathrm{Set}$ (for $\mathsf{len}(\vec{x}) = n$), with $!U \,\&\, (!U \multimap U) \lhd U$. The intuition is that, in Dist, 0-cells represent types (and in our untyped setting, they satisfy a retraction), 1-cells represent type derivations and 2-cells represent reduction on derivations.

We build the retraction in Dist, in analogy with the construction (1) in Polr. Indeed, they are both special cases of the free-algebra construction for an (unpointed) endofunctor [36]. We recall that, in Dist, $A \,\&\, B = A \sqcup B$, $A \otimes B = A \times B$ (so $A^{\otimes n} = A^n$) and $A \multimap B = A^{\mathrm{op}} \times B$.

[6] The only delicate point is the $\star$-autonomy of the bicategory, since it does not exist in the literature a notion of $\star$-autonomous bicategory. However it is possible to equip distributors with a dualizing pseudo-endofunctor, as shown for example in [12, 27].

Types:

$$a := x \in A \mid \langle a_1, \dots, a_k \rangle \Rightarrow a \mid \langle a_1, \dots, a_k \rangle$$

Morphisms in U:

$$\frac{f \in A[x, x']}{f \in U[x, x']} \qquad \frac{\langle \sigma, \vec{f} \rangle \colon \vec{a}' \to \vec{a} \qquad f \colon a \to a'}{\langle \sigma, \vec{f} \rangle \Rightarrow f \colon (\vec{a} \Rightarrow a) \to (\vec{a}' \Rightarrow a')}$$

$$\frac{\sigma \in S_n \quad f_1 \colon a_1 \to a'_{\sigma(1)} \quad \cdots \quad f_n \colon a_n \to a'_{\sigma(n)}}{\langle \sigma, f_1, \dots, f_n \rangle \colon \langle a_1, \dots, a_n \rangle \to \langle a'_1, \dots, a'_n \rangle}$$

Derivation rules:

$$\frac{f \colon a' \to a}{x_1 : \langle\rangle, \dots, x_i : \langle a' \rangle, \dots x_n : \langle\rangle \vdash x_i : a}$$

$$\frac{\Gamma \vdash S : \vec{a} \Rightarrow a \qquad \Gamma' \vdash T : \vec{a} \qquad \eta \colon \Delta \to \Gamma \otimes \Gamma'}{\Delta \vdash ST : a}$$

$$\frac{\Gamma_1 \vdash S : a_1 \overset{k \in \mathbb{N}}{\dots} \Gamma_k \vdash S : a_k \quad \eta \colon \Delta \to \bigotimes_{i=1}^{k} \Gamma_i}{\Delta \vdash S^! : \langle a_1, \dots, a_k \rangle}$$

$$\frac{\Delta, x : \vec{a} \vdash S : a}{\Delta \vdash \lambda x.S : \vec{a} \Rightarrow a}$$

Figure 3 Non-idempotent intersection type system $\mathcal{R}_{\to}$ associated with the 0-cell U in Dist.

▶ **Definition 4.** *Let A be a small category. We define a family of small categories $(U_n)_{n \in \mathbb{N}}$ by:*

$$U_0 = A \qquad U_{n+1} = !U_n \sqcup ((!U_n^{\mathrm{op}} \times U_n) \sqcup A)$$

We define a family of inclusions $(\iota_n : U_n \hookrightarrow U_{n+1})_{n \in \mathbb{N}}$ in the canonical way:

$$\iota_0 = \iota_A \qquad \iota_{n+1} = !(\iota_n) \sqcup ((!(\iota_n)^{\mathrm{op}} \times \iota_n) \sqcup 1_A)$$

Then we set $U_A = \varinjlim_{n \in \mathbb{N}} U_n$. From now on, the 0-cell U_A will be simply denoted by U, keeping the parameter A implicit. We denote by $\xi_n \colon !U_n \sqcup (!U_n^{\mathrm{op}} \times U_n) \hookrightarrow U_n$ the canonical inclusions.

▶ **Lemma 5** (Inclusion). *There exists a canonical inclusion $\iota : !U \sqcup (!U^{\mathrm{op}} \times U) \hookrightarrow U$.*

Proof. Since U is a filtered colimit, we have $!U \sqcup (!U^{\mathrm{op}} \times U) \cong \varinjlim_{n \in \mathbb{N}} !U_n \sqcup (\varinjlim_{n \in \mathbb{N}} !U_n^{\mathrm{op}} \times \varinjlim_{n \in \mathbb{N}} U_n)$, and so we can explicitly define the inclusion functor as $\iota(a) = y_{j+1}(\xi_j(a))$ where $j = \min\{n \in \mathbb{N} \mid a \in U_n \sqcup (!U_n^{\mathrm{op}} \times U_n)\}$ and $y_{j+1} \colon U_{j+1} \to U$ is the canonical injection of U_{j+1}. ◀

▶ **Theorem 6** (Retraction). *We have that $!U \,\&\, (!U \multimap U) \lhd U$ in* Dist.

So, the 0-cell U is a (non-extensional) denotational model of the bang calculus. By seeing the objects of A (resp. U) as the *atomic types* (resp. *types*) and setting $\vec{a} \Rightarrow a = \langle \vec{a}, a \rangle \in !U \times U$, we give in Figure 3 a type-theoretic description of the 0-cell U. This *non-idempotent intersection type system*, called $\mathcal{R}_{\to}$, is the generalization in Dist of the system $\mathcal{R}_{\leq}$ in Figure 2 associated with Polr. A morphism $f \colon a \to b$ in Figure 3 can be seen as a witness in Dist of the subtyping relation between a and b, generalizing $a \leq_U b$ of Polr.

Semantics of Bang Terms. We now present the *semantics* (or denotation) of bang terms as distributors in the bicategory Dist. We recall that $\iota \colon !U \,\&\, (!U^{\mathrm{op}} \times U) \hookrightarrow U$. Let $\Gamma = \langle \vec{b}_1, \dots, \vec{b}_n \rangle$, $\Delta = \langle \vec{b}'_1, \dots, \vec{b}'_n \rangle \in !U^n$. A morphism $\eta \colon \Gamma \to \Delta$ is a list of morphisms $\eta = \langle \langle \sigma_1, \vec{f}_1 \rangle, \dots, \langle \sigma_n, \vec{f}_n \rangle \rangle \colon \Gamma \to \Delta$ where $\langle \sigma_i, \vec{f}_i \rangle \colon \vec{b}_i \to \vec{b}'_i$. We set $\Gamma \otimes \Delta = \langle \vec{b}_1 \oplus \vec{b'}_1, \dots, \vec{b}_n \oplus \vec{b'}_n \rangle$. This tensor product inherits the relevant structure from $\oplus$. In particular, the symmetries $\vec{\sigma} \colon \bigotimes_{i=1}^{k} \Gamma_i \to \bigotimes_{i=1}^{k} \Gamma_{\sigma(i)}$ are built from the $\sigma^{\star}$ construction presented in Section 2.

▶ **Definition 7** (Semantics). *Let $S \in !\Lambda$ and $\mathsf{fv}(S) \subseteq \vec{x} = \langle x_1, \dots, x_n \rangle$, with the x_i's pairwise distinct. The* semantics *$[\![S]\!]_{\vec{x}} \colon !U^{\otimes n} \nrightarrow U$ of S with respect to $\vec{x}$ is defined by induction on S:*

- $[\![x_i]\!]_{\vec{x}}(\Delta, a) = !U^n[\Delta, \langle \langle\rangle, \dots, \langle a \rangle, \dots \langle\rangle \rangle]$ *($\langle a \rangle$ is in the i^{th} position in $\langle \langle\rangle, \dots, \langle a \rangle, \dots, \langle\rangle \rangle$);*

$$[g\colon a \to b]\left(\frac{f\colon a' \to a}{x_1 : \langle\rangle, \dots, x_i : \langle a' \rangle, \dots, x_n : \langle\rangle \vdash x_i : a}\right) \quad = \quad \frac{g \circ f\colon a' \to b}{x_1 : \langle\rangle, \dots, x_i : \langle a' \rangle, \dots, x_n : \langle\rangle \vdash x_i : b}$$

$$[\langle \sigma, \vec{g} \rangle \Rightarrow g\colon (\vec{a} \Rightarrow a) \to (\vec{b} \Rightarrow b)]\left(\frac{\begin{array}{c}\vdots\, \pi \\ \Delta, x : \vec{a} \vdash S : a\end{array}}{\Delta \vdash \lambda x.S : \vec{a} \Rightarrow a}\right) \quad = \quad \frac{\begin{array}{c}\vdots\, [g]\pi\{\langle 1, \langle \sigma, \vec{g} \rangle\rangle\} \\ \Delta, x : \vec{b} \vdash S : b\end{array}}{\Delta \vdash \lambda x.S : \vec{b} \Rightarrow b}$$

$$[g\colon a \to b]\left(\frac{\begin{array}{c}\vdots\, \pi_1 \\ \Gamma_1 \vdash S\colon \vec{a} \Rightarrow a\end{array} \quad \begin{array}{c}\vdots\, \pi_2 \\ \Gamma_2 \vdash T : \vec{a}\end{array} \quad \eta\colon \Delta \to \Gamma_1 \otimes \Gamma_2}{\Delta \vdash ST\colon a}\right) = \frac{\begin{array}{c}\vdots\, [1 \Rightarrow g]\pi_1 \\ \Gamma_1 \vdash S : \vec{a} \Rightarrow b\end{array} \quad \begin{array}{c}\vdots\, \pi_2 \\ \Gamma_2 \vdash T : \vec{a}\end{array} \quad \eta\colon \Delta \to \Gamma_1 \otimes \Gamma_2}{\Delta \vdash ST : b}$$

$$[(\sigma, \vec{g})\colon \vec{a} \to \vec{b}]\left(\frac{\left(\begin{array}{c}\vdots\, \pi_i \\ \Gamma_i \vdash S : a_i\end{array}\right)_{i=1}^{k} \quad \eta : \Delta \to \bigotimes_{i=1}^{k} \Gamma_i}{\Delta \vdash S^! : \vec{a} = \langle a_1, \dots, a_k \rangle}\right) = \frac{\left(\begin{array}{c}\vdots\, \pi_i' \\ \Gamma_{\sigma^{-1}(i)} \vdash S : a_{\sigma^{-1}(i)}\end{array}\right)_{i=1}^{k} \quad \vec{\sigma}^{-1} \circ \eta : \Delta \to \bigotimes_{i=1}^{k} \Gamma_{\sigma^{-1}(i)}}{\Delta \vdash S^! : \vec{b} = \langle b_1, \dots, b_k \rangle}$$

Figure 4 Left action on derivations. In the last identity, on the right, $\pi_i' = [g_{\sigma^{-1}(i)}]\pi_{\sigma^{-1}(i)}$.

- $[\![\lambda y.S]\!]_{\vec{x}}(\Delta, a) = \begin{cases} [\![S]\!]_{\vec{x} \oplus \langle y \rangle}(\Delta \oplus \langle \vec{a} \rangle, a') & \textit{if } a = \iota(\langle \vec{a}, a' \rangle) \\ \emptyset & \textit{otherwise.} \end{cases}$, *where* $y \notin \vec{x}$;
- $[\![ST]\!]_{\vec{x}}(\Delta, a) = \displaystyle\int^{\vec{a} \in !U} \int^{\Gamma_1, \Gamma_2 \in !U^n} [\![S]\!]_{\vec{x}}(\Gamma_1, \iota(\langle \vec{a}, a \rangle)) \times [\![T]\!]_{\vec{x}}(\Gamma_2, \iota(\vec{a})) \times (!U^n)(\Delta, \Gamma_1 \otimes \Gamma_2)$;
- $[\![S^!]\!]_{\vec{x}}(\Delta, a) = \begin{cases} \displaystyle\int^{\Gamma_1, \dots, \Gamma_k \in !U^n} \prod_{i=1}^{k} [\![S]\!]_{\vec{x}}(\Gamma_i, a_i) \times (!U^n)(\Delta, \bigotimes_{i=1}^{k} \Gamma_i) & \textit{if } a = \iota(\langle a_1, \dots, a_k \rangle) \\ \emptyset & \textit{otherwise.} \end{cases}$

Given $\langle \Delta, a \rangle \in !U^n \times U$ we call *points* the elements of $[\![S]\!]_{\vec{x}}(\Delta, a)$. From now on, when we write $[\![S]\!]_{\vec{x}}$ we always assume that $\mathsf{fv}(S) \subseteq \vec{x} = \langle x_1, \dots, x_n \rangle$ and the x_i's are pairwise distinct.

The semantics of a term S is a functor $[\![S]\!]_{\vec{x}}\colon (!U^n)^{\mathrm{op}} \times U \to \mathrm{Set}$. As such, it must be defined on the objects of the category $(!U^n)^{\mathrm{op}} \times U$ (as done in Definition 7) *and* on the morphisms of the category $(!U^n)^{\mathrm{op}} \times U$. The action on morphisms (omitted in Definition 7) is given by induction on S and, in the application and bang cases, also by the universal property of the coend construction. The variable case is just the hom-functor. An explicit definition of the application and bang cases can be given by considering coends as coequalizers [44].

Non-idempotent Intersection Type Distributors. We aim to define the *non-idempotent intersection type distributor* $\mathsf{T}_U(S)_{\vec{x}}$ for any term S. Let π be a type derivation in system $\mathcal{R}_{\to}$, as defined in Figure 3. The *left* and *right actions* of morphisms on π are defined in Figures 4 and 5, respectively (by induction on π). Given $f\colon a \to a'$ and $\theta\colon \Delta' \to \Delta$, the left and right actions may change the conclusion of a type derivation:

$$\text{left:} \quad [f]\left(\begin{array}{c}\pi \\ \vdots \\ \Delta \vdash S : a\end{array}\right) \rightsquigarrow \begin{array}{c}[f]\pi \\ \vdots \\ \Delta \vdash S : a'\end{array} \qquad\qquad \text{right:} \quad \left(\begin{array}{c}\pi \\ \vdots \\ \Delta \vdash S : a\end{array}\right)\{\theta\} \rightsquigarrow \begin{array}{c}\pi\{\theta\} \\ \vdots \\ \Delta' \vdash S : a\end{array}$$

Notice the contravariance of the right action, and that $[f](\pi\{\theta\}) = ([f]\pi)\{\theta\}$.

We define $\sim$ as the smallest congruence on type derivations generated by the rules in Figure 6. We denote by $\tilde{\pi}$ the equivalence class of π modulo $\sim$. Note that $[f]\tilde{\pi}\{\theta\} = \widetilde{[f]\pi\{\theta\}}$.

▶ **Example 8.** We give a couple of examples of the equivalence $\sim$ between type derivations in system $\mathcal{R}_{\to}$. The intuition is that $\sim$ equalizes type derivations for the same term and with the same conclusion, where the "same" permutations are performed at different moments.

$$\dfrac{f : a' \to a}{x_1 : \langle\rangle, \dots, x_i : \langle a'\rangle, \dots, x_n : \langle\rangle \vdash x_i : a}\{\langle g : b \to a'\rangle\} = \dfrac{f \circ g : b \to a}{x_1 : \langle\rangle, \dots, x_i : \langle b\rangle, \dots, x_n : \langle\rangle \vdash x_i : a}$$

$$\left(\dfrac{\begin{array}{c}\pi\\ \vdots\\ \Delta, x : \vec{a} \vdash S : a\end{array}}{\Delta \vdash \lambda x.S : \vec{a} \Rightarrow a}\right)\{\theta\} = \dfrac{\begin{array}{c}\pi\{\theta \oplus \langle 1\rangle\}\\ \vdots\\ \Delta', x : \vec{a} \vdash S : a\end{array}}{\Delta' \vdash \lambda x.S : \vec{a} \Rightarrow a}$$

$$\left(\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma_1 \vdash S \colon \vec{a} \Rightarrow a\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma_2 \vdash T : \vec{a}\end{array} \quad \eta : \Delta \to \Gamma_1 \otimes \Gamma_2}{\Delta \vdash ST \colon a}\right)\{\theta\} = \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma_1 \vdash S \colon \vec{a} \Rightarrow a\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma_2 \vdash T : \vec{a}\end{array} \quad \eta \circ \theta \colon \Delta' \to \Gamma_1 \otimes \Gamma_2}{\Delta' \vdash ST : a}$$

$$\left(\dfrac{\left(\begin{array}{c}\pi_i\\ \vdots\\ \Gamma_i \vdash S : a_i\end{array}\right)_{i=1}^{k} \quad \eta : \Delta \to \bigotimes_{i=1}^{k} \Gamma_i}{\Delta \vdash S^! : \langle a_1, \dots, a_k\rangle}\right)\{\theta\} = \dfrac{\left(\begin{array}{c}\pi_i\\ \vdots\\ \Gamma_i \vdash S : a_i\end{array}\right)_{i=1}^{k} \quad \eta \circ \theta \colon \Delta' \to \bigotimes_{i=1}^{k} \Gamma_i}{\Delta' \vdash S^! : \langle a_1, \dots, a_k\rangle}$$

Figure 5 Right action on derivations, where $\theta \colon \Delta' \to \Delta$.

Let $f \colon a' \to a$ be a morphism between types a' and a. One can think of them as, *e.g.* $a = \langle *, \langle * \rangle \Rightarrow * \rangle$ and $a' = \langle \langle * \rangle \Rightarrow *, * \rangle$ with $f = \sigma \Rightarrow 1$ being the obvious permutation.

1. Let us type the term $xx^!$ with the following type derivation π (where $A = \langle \langle a \rangle \Rightarrow a, a \rangle$, $A' = \langle a', \langle a \rangle \Rightarrow a \rangle$ and $(1\,2) \in S_2$ is the swap permutation on $\{1, 2\}$):

$$\dfrac{\dfrac{1_{\langle a\rangle\Rightarrow a} \colon (\langle a\rangle \Rightarrow a) \to (\langle a\rangle \Rightarrow a)}{x : \langle\langle a\rangle \Rightarrow a\rangle \vdash x : \langle a\rangle \Rightarrow a} \quad \dfrac{\dfrac{1_a \colon a \to a}{x : \langle a\rangle \vdash x : a} \quad 1_{\langle a\rangle} \colon \langle a\rangle \to \langle a\rangle}{x : \langle a\rangle \vdash x^! : \langle a\rangle} \quad \langle (1\,2), f, 1_{\langle a\rangle\Rightarrow a}\rangle \colon A' \to A}{x : \langle a', \langle a\rangle \Rightarrow a\rangle \vdash xx^! : a}$$

Now consider the following type derivation π' (with $A'' = \langle \langle a \rangle \Rightarrow a, a' \rangle$)

$$\dfrac{\dfrac{1_{\langle a\rangle\Rightarrow a} \colon (\langle a\rangle \Rightarrow a) \to (\langle a\rangle \Rightarrow a)}{x : \langle\langle a\rangle \Rightarrow a\rangle \vdash x : \langle a\rangle \Rightarrow a} \quad \dfrac{\dfrac{f \colon a' \to a}{x : \langle a'\rangle \vdash x : a} \quad 1_{\langle a'\rangle} \colon \langle a'\rangle \to \langle a'\rangle}{x : \langle a'\rangle \vdash x^! : \langle a\rangle} \quad \langle (1\,2), 1_{a'}, 1_{\langle a\rangle\Rightarrow a}\rangle \colon A' \to A''}{x : \langle a', \langle a\rangle \Rightarrow a\rangle \vdash xx^! : a}$$

Compared to π, π' brings forward the morphism f. By the second rule in Figure 6, $\pi \sim \pi'$.

2. Let us type the term $(\lambda x.x)z^!$ (we omit the index on the identity morphisms 1):

$$\pi = \dfrac{\dfrac{\dfrac{f \colon a' \to a}{x : \langle a'\rangle \vdash x : a}}{\vdash \lambda x.x : \langle a'\rangle \Rightarrow a} \quad \dfrac{\dfrac{1 \colon a' \to a'}{z : \langle a'\rangle \vdash z : a'} \quad 1 \colon \langle a'\rangle \to \langle a'\rangle}{z : \langle a'\rangle \vdash z^! : \langle a'\rangle} \quad 1 \colon \langle a'\rangle \to \langle a'\rangle}{z : \langle a'\rangle \vdash (\lambda x.x)z^! : a}$$

Now consider the following derivation (note the different position of f with respect to π)

$$\pi' = \dfrac{\dfrac{\dfrac{1 \colon a \to a}{x : \langle a\rangle \vdash x : a}}{\vdash \lambda x.x : \langle a\rangle \Rightarrow a} \quad \dfrac{\dfrac{f \colon a' \to a}{z : \langle a'\rangle \vdash z : a} \quad 1 \colon \langle a'\rangle \to \langle a'\rangle}{z : \langle a'\rangle \vdash z^! : \langle a\rangle} \quad 1 \colon \langle a'\rangle \to \langle a'\rangle}{z : \langle a'\rangle \vdash (\lambda x.x)z^! : a}$$

According to the first rule in Figure 6, $\pi \sim \pi'$.

$$
\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma_1 \vdash S : \vec{b} \Rightarrow a\end{array} \quad \begin{array}{c}[\langle\sigma,\vec{f}\rangle]\pi_2\\ \vdots\\ \Gamma_2 \vdash T : \vec{b}\end{array} \quad \eta : \Delta \to \Gamma_1 \otimes \Gamma_2}{\Delta \vdash ST : a}
\sim
\dfrac{\begin{array}{c}[\langle\sigma,\vec{f}\rangle \Rightarrow 1]\pi_1\\ \vdots\\ \Gamma_1 \vdash S : \vec{a} \Rightarrow a\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma_2 \vdash T : \vec{a}\end{array} \quad \eta : \Delta \to \Gamma_1 \otimes \Gamma_2}{\Delta \vdash ST : a}
$$

$$
\dfrac{\begin{array}{c}\pi_1\{\theta_1\}\\ \vdots\\ \Gamma_1 \vdash S : \vec{a} \Rightarrow a\end{array} \quad \begin{array}{c}\pi_2\{\theta_2\}\\ \vdots\\ \Gamma_2 \vdash T : \vec{a}\end{array} \quad \eta : \Delta \to \Gamma_1 \otimes \Gamma_2}{\Delta \vdash ST : a}
\sim
\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma_1' \vdash S : \vec{a} \Rightarrow a\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma_2' \vdash T : \vec{a}\end{array} \quad \theta \circ \eta : \Delta \to \Gamma_1' \otimes \Gamma_2'}{\Delta \vdash ST : a}
$$

$$
\dfrac{\left(\begin{array}{c}\pi_i\{\theta_i\}\\ \vdots\\ \Gamma_i \vdash S : a_i\end{array}\right)_{i=1}^{k} \quad \eta : \Delta \to \bigotimes_{i=1}^{k} \Gamma_i}{\Delta \vdash S^! : \langle a_1, \dots, a_k\rangle}
\sim
\dfrac{\left(\begin{array}{c}\pi_i\\ \vdots\\ \Gamma_i' \vdash S : a_i\end{array}\right)_{i=1}^{k} \quad \bigotimes_{i=1}^{k} \theta_i \circ \eta : \Delta \to \bigotimes_{i=1}^{k} \Gamma_i'}{\Delta \vdash S^! : \langle a_1, \dots, a_k\rangle}
$$

Figure 6 Congruence on type derivations, where $\langle\sigma, \vec{f}\rangle \colon \vec{a} \to \vec{b}$ and $\theta = \theta_1 \otimes \theta_2$ with $\theta_i \colon \Gamma_i \to \Gamma_i'$.

Let S be a term and $\mathsf{fv}(S) \subseteq \vec{x} = \{x_1, \dots, x_n\}$ with the x_i's pairwise distinct. With any $\langle\Delta, a\rangle \in \mathrm{ob}((!U^n)^{\mathrm{op}} \times U)$, the distributor $\mathsf{T}_U(S)_{\vec{x}} \colon !U^n \nrightarrow U$ associates the set of (equivalence classes of) type derivations for S with conclusion $\Delta \vdash S : a$. Formally, $\mathsf{T}_U(S)_{\vec{x}}$ is defined by:

1. for $\langle\Delta, a\rangle \in \mathrm{ob}((!U^n)^{\mathrm{op}} \times U)$, $\mathsf{T}_U(S)_{\vec{x}}(\Delta, a) = \left\{ \begin{array}{c}\tilde{\pi}\\ \vdots\\ \Delta \vdash S : a\end{array} \;\middle|\; \pi \text{ is a type derivation for } S \right\}$;
2. for $f \colon a \to a'$ and $\eta \colon \Delta' \to \Delta$, $\mathsf{T}_U(S)_{\vec{x}}(\eta, f) \colon \mathsf{T}_U(S)_{\vec{x}}(\Delta, a) \to \mathsf{T}_U(S)_{\vec{x}}(\Delta', a')$ such that $\mathsf{T}_U(S)_{\vec{x}}(\eta, f)(\tilde{\pi}) = \widetilde{[f]\pi\{\eta\}} \in \mathsf{T}_U(S)_{\vec{x}}(\Delta', a')$ for any $\tilde{\pi} \in \mathsf{T}_U(S)_{\vec{x}}(\Delta, a)$.

▶ **Lemma 9** (Functoriality). *For any $S \in !\Lambda$, $\mathsf{T}_U(S)_{\vec{x}}$ is a functor from $(!U^n)^{\mathrm{op}} \times U$ to* Set.

The following theorem states that the distributor semantics induced by our category of types U can be seen in a completely type-theoretic way. The semantics $[\![S]\!]_{\vec{x}}(\Delta, a)$ of a term S is equal to the set of (equivalence classes of) type derivations whose conclusion is the sequent $\Delta \vdash S : a$. For this reason we have a bicategorical *proof relevant* semantics. This is a major improvement over relational semantics, where the elements of the denotation of a term are only witnesses of typability. Said differently, the relational semantics of S is just the set of conclusions of the type derivations for S, while the distributor semantics of S provides, for any conclusion, the set of type derivations for S with such a conclusion.

▶ **Theorem 10** (Proof-relevance). *Let $S \in !\Lambda$. There is an isomorphism of functors*

$$\psi \colon [\![S]\!]_{\vec{x}} \cong \mathsf{T}_U(S)_{\vec{x}} \quad \textit{which is natural in } \langle\Delta, a\rangle \in \mathrm{ob}((!U^n)^{\mathrm{op}} \times U).$$

Proof. By induction on the structure of S. The core of the proof is the remark that we can write the equivalence relation induced by the coend in the application and box cases with the rules in Figure 6. ◀

For any type derivation π in system $\mathcal{R}_{\to}$ we define its *size* $\mathrm{s}(\pi)$ in Figure 7 (by induction on π). It counts the number of rules for application in π. Note that if $\pi \sim \pi'$ then $\mathrm{s}(\pi) = \mathrm{s}(\pi')$. We also have that size is invariant under morphisms action: $\mathrm{s}([f]\pi) = \mathrm{s}(\pi\{\eta\}) = \mathrm{s}(\pi)$.

Let $\psi \colon [\![S]\!]_{\vec{x}} \cong \mathsf{T}_U(S)_{\vec{x}}$ be the natural isomorphism of Theorem 10. For $\alpha \in [\![S]\!]_{\vec{x}}(\Delta, a)$ we set $\mathrm{s}(\alpha) = \mathrm{s}(\psi_{\Delta,a}(\alpha))$, *i.e.* the size of a point α is the size of its derivation $\psi_{\Delta,a}(\alpha)$.

Substitution and Reduction. We prove both subject reduction and expansion for non-idempotent intersection type distributors. We enrich this result with a quantitative flavor,

$$\mathrm{s}\left(\dfrac{f : a' \to a}{x_1 : \langle\rangle, \ldots, x_i : \langle a' \rangle, \ldots, x_n : \langle\rangle \vdash x_i : a}\right) = 0 \qquad \mathrm{s}\left(\dfrac{\left(\begin{array}{c}\pi_i \\ \vdots \\ \Gamma_i \vdash S : a_i\end{array}\right)_{i=1}^{k} \quad \theta : \Delta \to \bigotimes_{i=1}^{k} \Gamma_i}{\Delta \vdash {!S} : \langle a_1, \ldots, a_k \rangle}\right) = \sum_{i \in [k]} \mathrm{s}\,(\pi_i)$$

$$\mathrm{s}\left(\dfrac{\begin{array}{c}\pi' \\ \vdots \\ \Delta, x : \vec{a} \vdash S : a\end{array}}{\Delta \vdash \lambda x.S : \vec{a} \Rightarrow a}\right) = \mathrm{s}\,(\pi') \qquad \mathrm{s}\left(\dfrac{\begin{array}{c}\pi_1 \\ \vdots \\ \Gamma_1 \vdash S : \vec{a} \Rightarrow a\end{array} \quad \begin{array}{c}\pi_2 \\ \vdots \\ \Gamma_2 \vdash T : \vec{a}\end{array} \quad \theta : \Delta \to \Gamma_1 \otimes \Gamma_2}{\Delta \vdash ST : a}\right) = \mathrm{s}\,(\pi_1) + \mathrm{s}\,(\pi_2) + 1$$

Figure 7 Size of type derivations in system $\mathcal{R}_{\to}$.

accounting for how the size of points is affected by a reduction step. In this way, we can give a combinatorial proof for the characterization of terms that are normalizable at depth 0.

The key ingredient is the substitution lemma below. We set:

$$Sub_{S,x,T}(\Delta, a) = \int^{\vec{a} \in !U} \int^{\Gamma_0, \Gamma_1 \in !U^n} [\![S]\!]_{\vec{x} \oplus \langle x \rangle}(\Gamma_0 \oplus \langle \vec{a} \rangle, a) \times [\![T^!]\!]_{\vec{x}}(\Gamma_1, \vec{a}) \times !U^n(\Delta, \Gamma_0 \otimes \Gamma_1).$$

▶ **Lemma 11** (Substitution). *Let S and T be terms. There is an isomorphism of functors*

$$\varphi \colon Sub_{S,x,T} \cong [\![S\{T/x\}]\!]_{\vec{x}}$$

natural in $\langle \Delta, a \rangle \in \mathrm{ob}((!U^n)^{\mathrm{op}} \times U)$ and such that $s\left(\varphi_{\Delta,a}(\langle \widetilde{\alpha_1, \alpha_2, \eta} \rangle)\right) = s\,(\alpha_1) + s\,(\alpha_2)$.

Proof. By induction on the structure of S, *via* lengthy coend manipulations. The proof of the application and list cases strongly relies on the fact that the tensor product of $!U$ is symmetric. The proof of the preservation of sizes relies on the fact that size is invariant under morphism actions and equivalence. Details are in Appendix A. ◀

▶ **Theorem 12** (Subject reduction and expansion). *Let S, T be two terms.*

1. *If $S \to_{\mathrm{b}} T$ then there is a natural isomorphism $[\![S]\!]_{\vec{x}}(\Delta, a) \cong [\![T]\!]_{\vec{x}}(\Delta, a)$.*
2. *If $S \to_{\mathrm{b_g}} T$ then $[\![S]\!]_{\vec{x}}(\Delta, a) \cong [\![T]\!]_{\vec{x}}(\Delta, a)$ via a natural isomorphism $\varphi_{\Delta,a}$ such that $s\,(\varphi_{\Delta,a}(\alpha)) = s\,(\alpha) - 1$ for any $\alpha \in [\![S]\!]_{\vec{x}}(\Delta, a)$.*
3. *If $S \to_{\sigma} T$ then $[\![S]\!]_{\vec{x}}(\Delta, a) \cong [\![T]\!]_{\vec{x}}(\Delta, a)$ via a natural isomorphism $\varphi_{\Delta,a}$ such that $s\,(\varphi_{\Delta,a}(\alpha)) = s\,(\alpha)$ for any $\alpha \in [\![S]\!]_{\vec{x}}(\Delta, a)$.*

Proof. We prove the base case of Item 2, which follows from the substitution lemma (Lemma 11). Let $S = (\lambda x.S_1)S_2^! \mapsto_{\mathrm{b}} S_1\{S_2/x\} = T$. By definition, we have

$$[\![S]\!]_{\vec{x}}(\Delta, a) = \int^{\vec{a} \in !U} \int^{\Gamma_1, \Gamma_2 \in !U^n} [\![\lambda x.S_1]\!]_{\vec{x}}(\Gamma_1, \iota(\langle \vec{a}, a \rangle)) \times [\![S_2^!]\!]_{\vec{x}}(\Gamma_2, \vec{a}) \times !U^n(\Delta, \Gamma_1 \otimes \Gamma_2).$$

By definition of an abstraction's denotation we have $[\![\lambda x.S_1]\!]_{\vec{x}}(\Gamma_1, \iota(\langle \vec{a}, a \rangle)) = [\![S_1]\!]_{\vec{x} \oplus \langle x \rangle}(\Gamma_1 \oplus \langle \vec{a} \rangle, a)$. Then, $[\![S]\!]_{\vec{x}}(\Delta, a) = Sub_{S_1,x,S_2}(\Delta, a)$. By Lemma 11, $\varphi_{\Delta,a} : [\![(\lambda x.S_1)S_2]\!]_{\vec{x}}\utilde{(\Delta, a)} \cong [\![S_1\{S_2/x\}]\!]_{\vec{x}}(\Delta, a)$. Again by Lemma 11, $\mathrm{s}\,(\varphi_{\Delta,a}(\beta)) = \mathrm{s}\,(\alpha_1) + \mathrm{s}\,(\alpha_2)$ for $\beta = \langle \alpha_1, \alpha_2, \eta \rangle \in [\![S]\!]_{\vec{x}}(\Delta, a)$. By definition, we have that $\mathrm{s}\,(\beta) = \mathrm{s}\,(\alpha_1) + \mathrm{s}\,(\alpha_2) + 1$. So, we can conclude.

For Item 3, a step $\to_{\sigma}$ just requires to rearrange the rule order in a type derivation. ◀

Roughly, Theorem 12.2 states that if $S \to_{\mathrm{b_g}} T$ then for every type derivation for S there is a type derivation for T, with the same conclusion, whose size decreases by 1. In Theorem 12.1 such a quantitative account does not hold. Indeed, consider $((\lambda x.x)y^!)^! \to_{\mathrm{b}} y^!$: each of the two terms can be typed with a derivation of size 0 (take the rule for boxes with 0 premises).

▶ **Example 13.** We provide a simple example of reduction of type derivations to ease the understanding of the congruence's role in establishing the natural isomorphisms. Consider $S = (\lambda x.x)y^!$. We type it with the following type derivations:

$$\pi_1 = \dfrac{\dfrac{\dfrac{h \circ f \colon a \to b}{x \colon \langle a\rangle \vdash x : b}}{\vdash \lambda x.x : \langle a\rangle \Rightarrow b} \qquad \dfrac{\dfrac{g \colon c \to a}{y : \langle c\rangle \vdash y : a}\;1}{y : \langle c\rangle \vdash y^! : \langle a\rangle}\;1}{y : \langle c\rangle \vdash (\lambda x.x)y^! : b} \qquad \pi_2 = \dfrac{\dfrac{\dfrac{h \circ f' \colon a' \to b}{x : \langle a\rangle \vdash x : b}}{\vdash \lambda x.x : \langle d\rangle \Rightarrow b} \qquad \dfrac{\dfrac{g' \colon c \to a'}{y : \langle c\rangle \vdash y : a'}\;1}{y : \langle c\rangle \vdash y^! : \langle a'\rangle}\;1}{y : \langle c\rangle \vdash (\lambda x.x)y^! : b}$$

Suppose that $f \circ g = f' \circ g'$ and $h \colon b \to b$, $f \colon a \to b$, $f' \colon a' \to b$. We have that $\pi_1 \sim \pi_2$. Indeed, by the first rule of Figure 6:

$$\pi_1 \sim \dfrac{\dfrac{\dfrac{h \colon b \to b}{x : \langle b\rangle \vdash x : b}}{\vdash \lambda x.x : \langle b\rangle \Rightarrow b} \qquad \dfrac{\dfrac{f \circ g : c \to b}{y : \langle c\rangle \vdash y : b}\;1}{y : \langle c\rangle \vdash y^! : \langle b\rangle}\;1}{y : \langle c\rangle \vdash (\lambda x.x)y^! : b} \qquad \pi_2 \sim \dfrac{\dfrac{\dfrac{h \colon b \to b}{x : \langle b\rangle \vdash x : b}}{\vdash \lambda x.x : \langle b\rangle \Rightarrow b} \qquad \dfrac{\dfrac{f' \circ g' : c \to b}{y : \langle c\rangle \vdash y : b}\;1}{y : \langle c\rangle \vdash y^! : \langle b\rangle}\;1}{y : \langle c\rangle \vdash (\lambda x.x)y^! : b}$$

and, by the hypothesis $f \circ g = f' \circ g'$, we conclude that $\pi_1 \sim \pi_2$ by transitivity. In particular, this means that the quotient identify all couple of morphisms leading to the same composition.

Now, we have that $S \to_{\mathsf{b}_\mathsf{g}} y$. Consider the following type derivation of y:

$$\pi_3 = \dfrac{h \circ (f \circ g) : c \to b}{y : \langle c\rangle \vdash y : b} \qquad \text{(note that } \mathsf{s}(\pi_1) = \mathsf{s}(\pi_2) = 1 \text{ and } \mathsf{s}(\pi_3) = 0).$$

By an easy inspection of the definitions we have that for $\varphi_{\langle c\rangle,b} : [\![S]\!]_{\langle y\rangle}(\langle c\rangle, b) \cong [\![y]\!]_{\langle y\rangle}(\langle c\rangle, b)$, $\varphi_{\langle c\rangle,b}(\tilde{\pi_1}) = \pi_3$, where we keep implicit the isomorphism given by Theorem 10. There is then a nice correspondence between *substitution* on the term side and *composition* on the morphism side, that validates the basic intuition of categorical semantics[7].

We prove that non-idempotent intersection type distributors characterize normalization at depth 0, when normal forms are clash-free at depth 0. First, we characterize syntactically the normal forms for $\to_{\mathsf{b}\sigma_\mathsf{g}}$ that are clash-free at depth 0. Consider the subsets $!\Lambda_\mathsf{d}$, $!\Lambda_\mathsf{n}$, $!\Lambda_\ell$ (whose elements are denoted by D, N, L, respectively) of $!\Lambda$:

$$(!\Lambda_\mathsf{d}) \quad D ::= x \mid DS^! \mid DD' \qquad (!\Lambda_\mathsf{n}) \quad N ::= S^! \mid D \mid (\lambda x.N)D \qquad (!\Lambda_\ell) \quad L ::= N \mid \lambda x.L$$

All terms in $!\Lambda_\mathsf{d}$ are not closed (they have a free "head variable") and are neither a box nor a β-like redex nor an abstraction. Clearly, $!\Lambda_\mathsf{d} \subsetneq !\Lambda_\mathsf{n}$ and $!\Lambda_! \subsetneq !\Lambda_\mathsf{n} \subsetneq !\Lambda_\ell$ with $!\Lambda_\mathsf{d} \cap !\Lambda_! = \emptyset$.

▶ **Proposition 14** (Syntactic characterization of clash-free at depth 0 normal forms for $\to_{\mathsf{b}\sigma_\mathsf{g}}$).

1. *A term S is normal for $\to_{\mathsf{b}\sigma_\mathsf{g}}$, clash-free at depth 0 and is neither a box nor a β-like redex (i.e. nor of the form $(\lambda x.S)T$) nor an abstraction iff $S \in !\Lambda_\mathsf{d}$.*
2. *A term S is normal for $\to_{\mathsf{b}\sigma_\mathsf{g}}$, clash-free at depth 0 and is not an abstraction iff $S \in !\Lambda_\mathsf{n}$.*
3. *A term S is normal for $\to_{\mathsf{b}\sigma_\mathsf{g}}$ and clash-free at depth 0 iff $S \in !\Lambda_\ell$.*

▶ **Lemma 15** (Semantics vs. clash-free at depth 0). *Let S be a term.*

1. *If $[\![S]\!]_{\vec{x}} \neq \emptyset_{!U^n,U}$ then S is clash-free at depth 0.*
2. *If S is normal for $\to_{\mathsf{b}\sigma_\mathsf{g}}$ and clash-free at depth 0, then $[\![S]\!]_{\vec{x}} \neq \emptyset_{!U^n,U}$.*

Proof. 1. By induction on $S \in !\Lambda$.

[7] The natural isomorphism $\varphi_{\langle c\rangle,b} : [\![S]\!]_{\langle y\rangle}(\langle c\rangle, b) \cong [\![y]\!]_{\langle y\rangle}(\langle c\rangle, b)$ is a particular instance of Yoneda's lemma for coends (see Lemma 20 in Appendix A), also known as the density formula for coends [40].

2. According to Proposition 14, we can proceed by induction on $S \in !\Lambda_\ell$. ◀

▶ **Theorem 16** (Normalization at depth 0). *Let S be a term. The following are equivalent:*

1. *S is typable in system $\mathcal{R}_\rightarrow$;*
2. *$[\![S]\!]_{\vec{x}} \neq \emptyset_{!U^n,U}$;*
3. *S is strongly $\rightarrow_{\mathrm{b}\sigma_\mathrm{g}}$-normalizable with a normal form for $\rightarrow_{\mathrm{b}\sigma_\mathrm{g}}$ that is clash-free at depth 0;*
4. *S is weakly $\rightarrow_{\mathrm{b}\sigma_\mathrm{g}}$-normalizable with a normal form for $\rightarrow_{\mathrm{b}\sigma_\mathrm{g}}$ that is clash-free at depth 0;*
5. *$S \rightarrow^*_{\mathrm{b}\sigma} T$ for some term T that is normal for $\rightarrow_{\mathrm{b}\sigma_\mathrm{g}}$ and clash free at depth 0.*

Proof. The equivalence (1) ⇔ (2) is given by Theorem 10. The implication (5) ⇒ (2) follows from Lemma 15.2 and Theorem 12. The implication (4) ⇒ (5) holds because $\rightarrow_{\mathrm{b}\sigma_\mathrm{g}} \subseteq \rightarrow_{\mathrm{b}\sigma}$. The implication (3) ⇒ (4) is trivial.

For the implication (2) ⇒ (3), as $[\![S]\!]_{\vec{x}} \neq \emptyset_{!U^n,U}$, there is a point $\alpha \in [\![S]\!]_{\vec{x}}(\Delta, a)$ for some $\langle \Delta, a\rangle \in \mathrm{ob}(!U^n \times U)$. Let k_S be the sum of the lengths of all $\rightarrow_{\sigma_\mathrm{g}}$-reduction sequences from S to a normal form for $\rightarrow_{\sigma_\mathrm{g}}$ (such a k_S exists because $\rightarrow_{\sigma_\mathrm{g}}$ is strongly normalizing [24]). We prove (3) by induction on $(\mathrm{s}\,(\alpha)\,, k_S)$ ordered lexicographically. If S is normal for $\rightarrow_{\mathrm{b}\sigma_\mathrm{g}}$, we are done by Lemma 15.1, as $\alpha \in [\![S]\!]_{\vec{x}}(\Delta, a)$ implies $[\![S]\!]_{\vec{x}} \neq \emptyset_{!U^n,U}$. Suppose $S \rightarrow_{\mathrm{b}\sigma_\mathrm{g}} S'$.

1. If $S \rightarrow_{\sigma_\mathrm{g}} S'$, let $\varphi\colon [\![S]\!]_{\vec{x}} \cong [\![S']\!]_{\vec{x}}$ be the natural isomorphism of Theorem 12.3. Thus, $\varphi_{\Delta,a}(\alpha) \in [\![S']\!]_{\vec{x}}(\Delta, a)$ and $\mathrm{s}\,(\alpha) = \mathrm{s}\,(\varphi_{\Delta,a}(\alpha))$ but $k_{S'} = k_S - 1$.
2. If $S \rightarrow_{\mathrm{b}_\mathrm{g}} S'$, let $\varphi\colon [\![S]\!]_{\vec{x}} \cong [\![S']\!]_{\vec{x}}$ be the natural isomorphism of Theorem 12.2. Thus, $\varphi_{\Delta,a}(\alpha) \in [\![S']\!]_{\vec{x}}(\Delta, a)$ and $\mathrm{s}\,(\varphi_{\Delta,a}(\alpha)) = \mathrm{s}\,(\alpha) - 1$.

In both cases, by *i.h.*, (3) holds for S'. Therefore, (3) holds for S. ◀

5 Conclusions

In this paper, we recalled some well-known and linear-logic based categorical semantics with an intersection type presentation. We showed that they can be generalized in the bicategory of distributors. We defined non-idempotent intersection type distributors in the bang calculus and provided a syntactic presentation of them as a non-idempotent intersection type system generalizing De Carvalho's system $\mathcal{R}$ [16, 17]. We proved that non-idempotent intersection type distributors determine a proof-relevant denotational semantics, and characterize normalization at depth 0 in the bang calculus via a combinatorial proof.

Perspectives. Reconciling the different methods used here and in [43, 41, 48] to categorify – non-idempotent or possibly idempotent – intersection types is the first and natural open question. The (non-trivial) answer should rely on a *subtyping-aware polyadic calculus* to be defined. This would allow [41, 48] to have a denotational semantics that supports subtyping.

Another line of research is the study of the extensional collapse [22] in the bicategorical setting of distributors, which should shed new light on the link between non-idempotent and idempotent intersection types. Relating the methods of [29, 43] should be a first step.

A relevant question immediately arises also for what concerns typed call-by-push-value [23, 39]. The extension of our work to that framework is tricky, since the semantics of types adds technical machinery. Moreover, we believe that difficulties similar to the ones found in [13] in order to define the Taylor expansion could arise also in our perspective.

Other interesting perspectives are the investigation of the relationship between our categorified rigid framework and rigid intersection types [50], and an extension of our approach to probabilistic computation. This extension is far from trivial, but the results of Tsukada, Asada and Ong [49] are encouraging, and the study of probabilistic Taylor expansion [38] and probabilistic intersection types [7] might be a starting point.

References

1 Fabio Alessi, Franco Barbanera, and Mariangiola Dezani-Ciancaglini. Intersection types and lambda models. *Theoretical Computer Science*, 355(2):108–126, 2006. Logic, Language, Information and Computation. doi:10.1016/j.tcs.2006.01.004.

2 Hendrik Pieter Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1984.

3 Jean Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, pages 1–77, Berlin, Heidelberg, 1967. Springer Berlin Heidelberg.

4 Alexis Bernadet and Stéphane Jean Lengrand. Non-idempotent intersection types and strong normalisation. *Logical Methods in Computer Science*, Volume 9, Issue 4, 2013. doi:10.2168/LMCS-9(4:3)2013.

5 Robert Blackwell, Gregory Maxwell Kelly, and John Power. Two-dimensional monad theory. *Journal of Pure and Applied Algebra*, 59(1):1–41, 1989. doi:10.1016/0022-4049(89)90160-6.

6 Francis Borceux. *Handbook of Categorical Algebra*, volume 1 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1994. doi:10.1017/CBO9780511525858.

7 Flavien Breuvart and Ugo Dal Lago. On Intersection Types and Probabilistic Lambda Calculi. In *Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming, PPDP 2018, Frankfurt am Main, Germany, September 03-05, 2018*, pages 8:1–8:13, 2018. doi:10.1145/3236950.3236968.

8 Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. In *Functional and Logic Programming - 15th International Symposium, FLOPS 2020*, volume 12073 of *Lecture Notes in Computer Science*, pages 13–32. Springer, 2020. doi:10.1007/978-3-030-59025-3_2.

9 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017. doi:10.1093/jigpal/jzx018.

10 Jean Bénabou. Distributors at work. Lecture notes of a course given at TU Darmstadt, 2000. URL: http://www2.mathematik.tu-darmstadt.de/~streicher/FIBR/DiWo.pdf.

11 Alberto Carraro and Giulio Guerrieri. A Semantical and Operational Account of Call-by-Value Solvability. In *Foundations of Software Science and Computation Structures, FOSSACS 2014*, volume 8412 of *Lecture Notes in Computer Science*, pages 103–118, Berlin, Heidelberg, 2014. Springer. doi:10.1007/978-3-642-54830-7_7.

12 Gian Luca Cattani and Glynn Winskel. Profunctors, open maps and bisimulation. *Mathematical Structures in Computer Science*, 15(3):553–614, 2005. doi:10.1017/S0960129505004718.

13 Jules Chouquet and Christine Tasson. Taylor expansion for Call-by-Push-Value. In *28th EACSL Annual Conference on Computer Science Logic, CSL 2020*, volume 152 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CSL.2020.16.

14 Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type-assignment for lambda terms. *Arch. Math. Log.*, 19(1):139–156, 1978. doi:10.1007/BF02011875.

15 Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980. doi:10.1305/ndjfl/1093883253.

16 Daniel de Carvalho. *Semantique de la logique lineaire et temps de calcul*. PhD thesis, Aix-Marseille Université, 2007.

17 Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Math. Struct. Comput. Sci.*, 28(7):1169–1203, 2018. doi:10.1017/S0960129516000396.

18 Daniel de Carvalho. Taylor expansion in linear logic is invertible. *Log. Methods Comput. Sci.*, 14(4), 2018. doi:10.23638/LMCS-14(4:21)2018.

19 Daniel de Carvalho, Michele Pagani, and Lorenzo Tortora de Falco. A semantic measure of the execution time in linear logic. *Theoretical Computer Science*, 412(20):1884–1902, 2011. doi:10.1016/j.tcs.2010.12.017.

20 Daniel de Carvalho and Lorenzo Tortora de Falco. A semantic account of strong normalization in linear logic. *Inf. Comput.*, 248:104–129, 2016. doi:10.1016/j.ic.2015.12.010.

21 Thomas Ehrhard. Collapsing non-idempotent intersection types. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012*, volume 16 of *LIPIcs*, pages 259–273. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.CSL.2012.259.

22 Thomas Ehrhard. The Scott model of linear logic is the extensional collapse of its relational model. *Theoretical Computer Science*, 424:20–45, 2012. doi:10.1016/j.tcs.2011.11.027.

23 Thomas Ehrhard. Call-by-push-value from a linear logic point of view. In *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016*, volume 9632 of *Lecture Notes in Computer Science*, pages 202–228. Springer, 2016. doi:10.1007/978-3-662-49498-1_9.

24 Thomas Ehrhard and Giulio Guerrieri. The bang calculus: An untyped lambda-calculus generalizing call-by-name and call-by-value. In *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, PPDP 2016*, pages 174–187. Association for Computing Machinery, 2016. doi:10.1145/2967973.2968608.

25 Thomas Ehrhard and Laurent Regnier. Böhm trees, Krivine's machine and the Taylor expansion of lambda-terms. In *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006*, volume 3988 of *Lecture Notes in Computer Science*, pages 186–197. Springer, 2006. doi:10.1007/11780342_20.

26 Thomas Ehrhard and Laurent Regnier. Uniformity and the Taylor expansion of ordinary λ-terms. *Theoretical Computer Science*, 403(2-3), 2008. doi:10.1016/j.tcs.2008.06.001.

27 Marcelo Fiore, Nicola Gambino, Martin Hyland, and Glynn Winskel. The cartesian closed bicategory of generalised species of structures. *J. of the London Mathematical Society*, 77(1):203–220, 2008. doi:10.1112/jlms/jdm096.

28 Marcelo Fiore, Nicola Gambino, Martin Hyland, and Glynn Winskel. Relative pseudomonads, Kleisli bicategories, and substitution monoidal structures. *Selecta Mathematica*, 24(3):2791–2830, November 2017. doi:10.1007/s00029-017-0361-3.

29 Zeinab Galal. A Profunctorial Scott Semantics. In *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, volume 167 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2020.16.

30 Nicola Gambino and André Joyal. On operads, bimodules and analytic functors. *Memoirs of the American Mathematical Society*, 249(1184):0–0, September 2017. doi:10.1090/memo/1184.

31 Philippa Gardner. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software, International Conference TACS '94*, volume 789 of *Lecture Notes in Computer Science*, pages 555–574. Springer, 1994. doi:10.1007/3-540-57887-0_115.

32 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987. doi:10.1016/0304-3975(87)90045-4.

33 Giulio Guerrieri and Giulio Manzonetto. The bang calculus and the two Girard's translations. In *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications, Linearity-TLLA@FLoC 2018*, volume 292 of *EPTCS*, pages 15–30, 2018. doi:10.4204/EPTCS.292.2.

34 Martin Hyland. Classical lambda calculus in modern dress. *Mathematical Structures in Computer Science*, 27(5):762–781, 2017. doi:10.1017/S0960129515000377.

35 André Joyal. Foncteurs analytiques et espèces de structures. In *Combinatoire énumérative*, pages 126–159, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

36 Gregory Maxwell Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on. *Bulletin of the Australian Mathematical Society*, 22(1):1–83, 1980. doi:10.1017/S0004972700006353.

37 Jean-Louis Krivine. Lambda-calculus, types and models. In *Ellis Horwood series in computers and their applications*, 1993.

38 Ugo Dal Lago and Thomas Leventis. On the Taylor expansion of probabilistic lambda-terms. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.13.

39 Paul Blain Levy. Call-by-Push-Value: A Subsuming Paradigm. In *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99*, volume 1581 of *Lecture Notes in Computer Science*, page 228–242. Springer, 1999. doi:10.1007/3-540-48959-2_17.

40 Fosco Loregian. This is the (co)end, my only (co)friend, 2015. arXiv:1501.02503.

41 Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *Proc. ACM Program. Lang.*, 2(POPL):6:1–6:28, 2018. doi:10.1145/3158094.

42 Paul-André Melliès and Noam Zeilberger. Functors are type refinement systems. *SIGPLAN Not.*, 50(1):3–16, January 2015. doi:10.1145/2775051.2676970.

43 Federico Olimpieri. Intersection Type Distributors, 2020. arXiv:2002.01287.

44 Federico Olimpieri. *Intersection Types and Resource Calculi in the Denotational Semantics of Lambda-Calculus.* PhD thesis, Aix-Marseille Université, 2020.

45 Luca Paolini and Simona Ronchi Della Rocca. Call-by-value solvability. *RAIRO Theor. Informatics Appl.*, 33(6):507–534, 1999. doi:10.1051/ita:1999130.

46 José Espírito Santo, Luís Pinto, and Tarmo Uustalu. Modal embeddings and calling paradigms. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019*, volume 131 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.18.

47 Alex K. Simpson. Reduction in a linear lambda-calculus with applications to operational semantics. In *Term Rewriting and Applications, 16th International Conference, RTA 2005*, volume 3467 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2005. doi:10.1007/978-3-540-32033-3_17.

48 Takeshi Tsukada, Kazuyuki Asada, and C.-H. Luke Ong. Generalised Species of Rigid Resource Terms. In *Proceedings of the 32rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS 2017, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005093.

49 Takeshi Tsukada, Kazuyuki Asada, and C.-H. Luke Ong. Species, profunctors and Taylor expansion weighted by SMCC: A unified framework for modelling nondeterministic, probabilistic and quantum programs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 889–898. IEEE Computer Society, 2018. doi:10.1145/3209108.3209157.

50 Pierre Vial. Infinitary intersection types as sequences: A new answer to Klop's problem. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005103.

A Appendix

Bicategories in a Nutshell [3, 6]. Intuitively, a bicategory is a category with "morphisms between morphisms", that is, where each hom-set itself carries the structure of a category, but the composition of morphisms is only associative up to an isomorphism, and similarly for the identities laws. Formally, a *bicategory* $\mathcal{C}$ consists of:

- a set $\mathrm{ob}(\mathcal{C})$ of *objects*, also called 0-*cells* and denoted by $A, B, C, \dots$;
- for all $A, B \in \mathrm{ob}(\mathcal{C})$, a category $\mathcal{C}(A, B)$; objects in $\mathcal{C}(A, B)$ are called 1-*cells* or *morphisms* from A to B; while arrows in $\mathcal{C}(A, B)$ (between 1-cells from A to B) are called 2-*cells* or 2-*morphisms*; composition of 2-cells is generally called *vertical composition*;
- for every $A, B, C \in \mathrm{ob}(\mathcal{C})$, a bifunctor

  $$\circ_{A,B,C} \colon \mathcal{C}(B, C) \times \mathcal{C}(A, B) \to \mathcal{C}(A, C)$$

  called *horizontal composition* (often the indices A, B, C in $\circ_{A,B,C}$ are omitted); hence, for all 1-cells $F \colon A \to B$, $F' \colon A \to B$ and $G \colon B \to C$, $G' \colon B \to C$, and for all 2-cells $\alpha \colon F \Rightarrow F'$ and $\beta \colon G \Rightarrow G'$, we have

  $$\text{a 1-cell } G \circ_{A,B,C} F \colon A \to C \quad \text{a 2-cell } \beta \circ_{A,B,C} \alpha \colon (G \circ_{A,B,C} F) \Rightarrow (G' \circ_{A,B,C} F');$$

- for every $A \in \mathrm{ob}(\mathcal{C})$ a functor $1_A \colon 1 \to \mathcal{C}(A, A)$; with an abuse of notation we identify $1_A(\star)$ with 1_A and we call it the identity of A;
- for all 1-cells $F \colon A \to B$, $G \colon B \to C$ and $H \colon C \to D$, a family of invertible 2-cells

  $$\alpha_{H,G,F} \colon H \circ (G \circ F) \cong (H \circ G) \circ F$$

  expressing the associativity laws;
- for every 1-cell $F \colon A \to B$, two families of invertible 2-cells

  $$\lambda_F \colon 1_B \circ F \cong F \qquad \rho_F \colon F \cong F \circ 1_A$$

  expressing the identity laws.

This data is subject to additional coherence axioms. A 2-*category* is a bicategory where the associativity and identities are strict equalities, not only isomorphisms.

▶ **Definition 17** (Retraction). *Let D, E be* 0*-cells in a bicategory $\mathcal{C}$. A* retraction *of D to E is a couple of* 1*-cells $i \colon E \to D$, $j \colon D \to E$ together with an invertible* 2*-cell β such that the diagram below commute. We write $E \lhd D$ is there is a retraction of D to E.*

$$\begin{tikzcd}
E \arrow[r, "i"] \arrow[d, "1_E"'] & D \arrow[dl, bend left, "j"] \arrow[dl, phantom, "\overset{\beta}{\Swarrow}"] \\
E &
\end{tikzcd}$$

Coends. Given a functor $F : C^{op} \times C \to \mathrm{Set}$ we recall that the coend is the coequalizer of the following diagram

$$\sum_{c,c' \in C} C(c', c) \times F(c, c') \rightrightarrows \sum_{c \in C} F(c, c) \to \int^{c \in C} F(c, c)$$

where the parallel arrows are given by left and right actions of F on morphisms $f \in C(c', c)$. Since we work with coends in the category of set, we have that this coequalizer is actually given by the quotient $\sum_{c \in C} F(c, c)/\sim$ where the equivalence relation is generated by the rule $x \sim y$ iff $F(f, c')(x) = y, F(c, f)(y) = x$, for $f : c' \to c$.

We list the three fundamental lemmas of coend calculus [40].

▶ **Lemma 18.** *Every cocontinuous functor preserves coends.*

▶ **Lemma 19** (Fubini [40])**.** *Let $F : C^{op} \times C \times D^{op} \times D \to Set$ be a functor. We have*

$$\int^{\langle c,d\rangle \in C\times D} F(c,c,d,d) \cong \int^{c\in C}\int^{d} F(c,c,d,d) \cong \int^{d\in D}\int^{c\in C} F(c,c,d,d).$$

▶ **Lemma 20** (Yoneda Ninja [40])**.** *Let $K, H : C \to Set$ be, respectively, a contravariant and a covariant functor. We have the following natural isomorphisms*

$$K(-) \cong \int^{c\in C} K(c) \times C(-,c) \qquad H(-) \cong \int^{c\in C} H(c) \times C(c,-).$$

Denotation under Reduction. In what follows we do not explicitly state, for readability reasons, when we apply Lemmas 18 and 19. For $\vec{\Gamma} = \langle \Gamma_1, \ldots, \Gamma_n\rangle$ we set $\bigotimes \vec{\Gamma} = \bigotimes_{i=1}^{n} \Gamma_i$.

▶ **Lemma 11** (Substitution)**.** *Let S and T be terms. There is an isomorphism of functors*

$$\varphi\colon Sub_{S,x,T} \cong [\![S\{T/x\}]\!]_{\vec{x}}$$

natural in $\langle \Delta, a\rangle \in \mathrm{ob}((!U^n)^{\mathrm{op}} \times U)$ and such that $s\left(\varphi_{\Delta,a}(\widetilde{\langle \alpha_1, \alpha_2, \eta\rangle})\right) = s(\alpha_1) + s(\alpha_2)$.

Proof. By induction on the structure of $S \in !\Lambda$, via lengthy coend manipulations.

If $S = x$ then

$$Sub_{S,x,T}(\Delta, a) = \int^{\Gamma_0,\Gamma_1 \in !U^n}\int^{\vec{a}\in !U} [\![x]\!]_{\vec{x}}(\Gamma_0 \oplus \langle \vec{a}\rangle, a) \times [\![T^!]\!]_{\vec{x}}(\Gamma_1, \vec{a}) \times !U(\Delta, \Gamma_0 \otimes \Gamma_1).$$

By definition we have

$$\cong \int^{\Gamma_0,\Gamma_1 \in !U^n}\int^{\vec{a}\in !U} !U^n(\Gamma_0 \oplus \langle \vec{a}\rangle, \langle\langle\rangle, \ldots, \langle\rangle, \langle a\rangle\rangle) \times [\![T^!]\!]_{\vec{x}}(\Gamma_1, \vec{a}) \times !U(\Delta, \Gamma_0 \otimes \Gamma_1).$$

Then, by the structure of the product category

$$\cong \int^{\Gamma_0,\Gamma_1 \in !U^n}\int^{\vec{a}\in !U} !U^n(\Gamma_0, \langle\langle\rangle, \ldots, \langle\rangle\rangle) \times !U(\vec{a}, \langle a\rangle) \times [\![T^!]\!]_{\vec{x}}(\Gamma_1, \vec{a}) \times !U(\Delta, \Gamma_0 \otimes \Gamma_1).$$

Then, by Yoneda (Lemma 20) we have

$$\cong \int^{\Gamma_1 \in !U^n}\int^{\vec{a}\in !U} !U(\vec{a}, \langle a\rangle) \times [\![T^!]\!]_{\vec{x}}(\Gamma_1, \vec{a}) \times !U(\Delta, \Gamma_1).$$

Again, by Yoneda (Lemma 20),

$$\cong \int^{\Gamma_1 \in !U^n} [\![T^!]\!]_{\vec{x}}(\Gamma_1, \langle a\rangle) \times !U(\Delta, \Gamma_1).$$

Then, by applying Yoneda one more time on the context Γ and by definition of the denotation of a box we can conclude. For what concerns the size, simply notice that $\mathrm{s}(\tilde{\pi}) = \mathrm{s}([f]\tilde{\pi})$.

The abstraction case follows from the *i.h.* immediately.

We do the application, the box case being similar to it. If $S = QR$ then

$$Sub_{S,x,T}(\Delta, a) = \int^{\Gamma_1,\Gamma_2}\int^{\vec{a}} [\![QR]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma_1 \oplus \langle \vec{a}\rangle, a) \times [\![T^!]\!]_{\vec{x}}(\Gamma_2, \vec{a}) \times !U^n(\Delta, \Gamma_1 \otimes \Gamma_2).$$

We develop $[\![QR]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma_1 \oplus \langle\vec{a}\rangle, a)$:

$$[\![QR]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma_1 \oplus \langle\vec{a}\rangle, a) = \int^{\Gamma'_1\oplus\langle\vec{a}_1\rangle,\Gamma'_2\oplus\langle\vec{a}_2\rangle}\int^{\vec{b}} [\![Q]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_1 \oplus \langle\vec{a}_1\rangle, \iota(\vec{b}, a))$$
$$\times [\![R]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_2 \oplus \langle\vec{a}_2\rangle, \vec{b}) \times !U^n(\Gamma_1 \oplus \langle\vec{a}\rangle, \Gamma'_1 \oplus \langle\vec{a}_1\rangle \otimes \Gamma'_2 \oplus \langle\vec{a}_2\rangle).$$

By the structure of the product category, we have

$$[\![QR]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma_1 \oplus \langle\vec{a}\rangle, a) = \int^{\Gamma'_1\Gamma'_2}\int^{\vec{a}_1,\vec{a}_2}\int^{\vec{b}} [\![Q]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_1 \oplus \langle\vec{a}_1\rangle, \iota(\vec{b}, a))$$
$$\times [\![R]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_2 \oplus \langle\vec{a}_2\rangle, \vec{b}) \times !U^n(\Gamma_1, \Gamma'_1 \otimes \Gamma'_2) \times !U(\vec{a}, \vec{a}_1 \oplus \vec{a}_2).$$

We apply Yoneda (Lemma 20) on Γ_1 and on $\vec{a}$ and we get

$$Sub_{S,x,T}(\Delta, a) \cong \int^{\Gamma'_1\Gamma'_2,\Gamma_2}\int^{\vec{b},\vec{a}_1,\vec{a}_2} [\![Q]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_1 \oplus \langle\vec{a}_1\rangle, \iota(\vec{b}, a)) \times [\![R]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_2 \oplus \langle\vec{a}_2\rangle, \vec{b})$$
$$\times [\![T^!]\!]_{\vec{x}}(\Gamma_2, \vec{a}_1 \oplus \vec{a}_2) \times !U^n(\Delta, (\Gamma'_1 \otimes \Gamma'_2) \otimes \Gamma_2).$$

By a simple inspection of the definition of the denotation of a box, we can rewrite it as

$$\cong \int^{\Gamma'_i,\Gamma_{\vec{a}_i},\Gamma_2}\int^{\vec{b},\vec{a}_1,\vec{a}_2} [\![Q]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_1 \oplus \langle\vec{a}_1\rangle, \iota(\vec{b}, a)) \times [\![R]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_2 \oplus \langle\vec{a}_2\rangle, \vec{b}) \times [\![T^!]\!]_{\vec{x}}(\Gamma_{\vec{a}_1}, \vec{a}_1)$$
$$\times [\![T^!]\!]_{\vec{x}}(\Gamma_{\vec{a}_2}, \vec{a}_2) \times !U^n(, \Gamma_2, \bigotimes\Gamma_{\vec{a}_1} \otimes \bigotimes\Gamma_{\vec{a}_2}) \times !U^n(\Delta, (\Gamma'_1 \otimes \Gamma'_2) \otimes \Gamma_2).$$

Where, if we set $\vec{a}_i = \langle a_{i,1}, \dots, a_{i,k_i}\rangle$, $[\![T]\!]_{\vec{x}}(\Gamma_{\vec{a}_i}, \vec{a}_i) = \prod_{j\in k_i}[\![T]\!]_{\vec{x}}(\Gamma_{i,j}, a_{i,j})$ and $\bigotimes\Gamma_{\vec{a}_i} = \bigotimes_{j\in k_i}\Gamma_{i,j}$ with $i \in \{1, 2\}$. We apply Yoneda on Γ_2

$$\cong \int^{\Gamma'_i,\Gamma_{\vec{a}_i}}\int^{\vec{b},\vec{a}_1,\vec{a}_2} [\![Q]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_1 \oplus \langle\vec{a}_1\rangle, \iota(\vec{b}, a)) \times [\![R]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_2 \oplus \langle\vec{a}_2\rangle, \vec{b}) \times [\![T^!]\!]_{\vec{x}}(\Gamma_{\vec{a}_1}, \vec{a}_1)$$
$$\times [\![T^!]\!]_{\vec{x}}(\Gamma_{\vec{a}_2}, \vec{a}_2) \times !U^n(\Delta, (\Gamma'_1 \otimes \Gamma'_2) \otimes (\bigotimes\Gamma_{\vec{a}_1} \otimes \bigotimes\Gamma_{\vec{a}_2})).$$

Now, by the *symmetry* of the tensor product $\otimes$ and by the fact that functors preserves isomorphisms, we get

$$\cong \int^{\Gamma'_i,\Gamma_{\vec{a}_i}}\int^{\vec{b},\vec{a}_1,\vec{a}_2} [\![Q]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_1 \oplus \langle\vec{a}_1\rangle, \iota(\vec{b}, a)) \times [\![R]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_2 \oplus \langle\vec{a}_2\rangle, \vec{b}) \times [\![T^!]\!]_{\vec{x}}(\Gamma_{\vec{a}_1}, \vec{a}_1)$$
$$\times [\![T^!]\!]_{\vec{x}}(\Gamma_{\vec{a}_2}, \vec{a}_2) \times !U^n(, \Gamma_2, \bigotimes\Gamma_{\vec{a}_1} \otimes \bigotimes\Gamma_{\vec{a}_2}) \times !U^n(\Delta, ((\Gamma'_1 \otimes \Gamma_{\vec{a}_1}) \otimes (\Gamma'_2 \otimes \Gamma_{\vec{a}_2})).$$

Now, if we apply Yoneda twice to $\Gamma'_i \otimes \Gamma_{\vec{a}_i}$, we get

$$\cong \int^{\Gamma'_i,\Gamma_{\vec{a}_i},\Delta_i}\int^{\vec{b},\vec{a}_1,\vec{a}_2} [\![Q]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma_1 \oplus \langle\vec{a}_1\rangle, \iota(\vec{b}, a)) \times [\![R]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_2 \oplus \langle\vec{a}_2\rangle, \vec{b}) \times [\![T^!]\!]_{\vec{x}}(\Gamma'_{\vec{a}_1}, \vec{a}_1)$$
$$\times [\![T^!]\!]_{\vec{x}}(\Gamma_{\vec{a}_2}, \vec{a}_2) \times !U^n(\Delta, \Delta_1 \otimes \Delta_2) \otimes !U^n(\Delta_1, \Gamma'_1 \otimes \bigotimes\Gamma_{\vec{a}_1}) \otimes !U^n(\Delta_2, \Gamma'_2 \otimes \bigotimes\Gamma_{\vec{a}_2}).$$

By co-continuity and commutativity, and by applying Yoneda (Lemma 20) twice, we have

$$\cong \int^{\vec{b}}\int^{\Gamma'_1,\Gamma_{\vec{a}_1},\Delta_1,\Phi_1}\int^{\vec{a}_1} [\![Q]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_1 \oplus \langle\vec{a}_1\rangle, \iota(\vec{b}, a)) \times [\![T^!]\!]_{\vec{x}}(\Phi_1, \vec{a}_1)$$
$$\times !U^n(\Phi_1, \bigotimes\Gamma'_{\vec{a}_1}) \times U^n(\Delta_1, \Gamma'_1 \otimes \Phi_1)$$
$$\times \int^{\Gamma'_2,\Gamma_{\vec{a}_2},\Delta_2,\Phi_2}\int^{\vec{a}_2} [\![R]\!]_{\vec{x}\oplus\langle x\rangle}(\Gamma'_2 \oplus \langle\vec{a}_2\rangle, \vec{b}) \times [\![T^!]\!]_{\vec{x}}(\Phi_2, \vec{a}_2)$$
$$\times !U^n(\Phi_2, \bigotimes\Gamma'_{\vec{a}_2}) \times U^n(\Delta_2, \Gamma'_2 \otimes \Phi_2) \times !U^n(\Delta, \Delta_1 \otimes \Delta_2).$$

By definition, the former coend is just

$$\int^{\vec{b}}\int^{\Delta_1,\Delta_2} Sub_{Q,x,T}(\Delta_1,\iota(\vec{b},a)) \times Sub_{R,x,T}(\Delta_2,\vec{b}) \times {!}U^n(\Delta,\Delta_1\otimes\Delta_2).$$

We remark that, forgetting the equivalence relation, the built isomorphism

$$Sub_{S,x,T}(\Delta,a) \cong \int^{\vec{b}} Sub_{Q,x,T}(\Delta_1,\iota(\vec{b},a)) \times Sub_{R,x,T}(\Delta_2,\vec{b}) \times {!}U^n(\Delta,\Delta_1\otimes\Delta_2)$$

consists of the following map

$$\langle\vec{a},\langle\vec{b},\langle\Gamma_1,\Gamma_2,\langle\langle\Gamma_1'\oplus\langle\vec{a}_1\rangle,\Gamma_2'\oplus\langle\vec{a}_2\rangle,\langle\alpha_1,\alpha_2,\eta_1\rangle\rangle,\langle\langle\vec{\Gamma}=\langle\Gamma_{2,1},\ldots,\Gamma_{2,\mathsf{len}(\vec{a})}\rangle,\vec{\beta}=\langle\beta_1,\ldots,\beta_{\mathsf{len}(\vec{a})}\rangle,\eta_2\rangle\rangle\rangle,\theta\rangle\rangle\rangle \mapsto$$

$$\langle\vec{b},\langle\Gamma_1'\otimes\bigotimes\Gamma_{\vec{a}_1},\alpha_1,\langle\vec{\beta}_{\vec{a}_1},1_{\bigotimes\Gamma_{\vec{a}_1}}\rangle,1_{\Gamma_1'\otimes\bigotimes\Gamma_{\vec{a}_1}}\rangle,\langle\Gamma_2'\otimes\bigotimes\Gamma_{\vec{a}_2},\alpha_2,\langle\vec{\beta}_{\vec{a}_2},1_{\bigotimes\Gamma_{\vec{a}_2}}\rangle,1_{\Gamma_2'\otimes\bigotimes\Gamma_{\vec{a}_2}}\rangle,$$
$$((\eta_1\otimes(\sigma^\star\circ\eta_2))\circ\theta)\circ\tau\rangle$$

where

- $\theta:\Delta\to\Gamma_1\otimes\Gamma_2, \alpha_1\in[\![Q]\!]_{\vec{x}}(\Gamma_1'\oplus\langle\vec{a}_1\rangle,\iota(\vec{b},a))$ and $\alpha_2\in[\![R]\!]_{\vec{x}}(\Gamma_2'\oplus\langle\vec{a}_2\rangle,\vec{b})$;
- $\langle\eta_1,f=\langle\sigma,\vec{f}\rangle\rangle:\Gamma_1\oplus\langle\vec{a}\rangle\to\Gamma_1'\oplus\langle\vec{a}_1\rangle\otimes\Gamma_2'\oplus\langle\vec{a}_2\rangle$ and $\eta_2:\Gamma_2\to\bigotimes\vec{\Gamma},\vec{\beta}\in[\![T]\!]_{\vec{x}}(\Gamma_2,\vec{a})$;
- $[f]\vec{\Gamma}=\Gamma_{\vec{a}_1}\otimes\Gamma_{\vec{a}_2}$ and $[f]\vec{\beta}=\vec{\beta}_{\vec{a}_1}\oplus\vec{\beta}_{\vec{a}_2}$;
- $\tau:(\Gamma_1'\otimes\Gamma_2')\otimes(\bigotimes\Gamma_{\vec{a}_1}\otimes\bigotimes\Gamma_{\vec{a}_2})\to(\Gamma_1'\otimes\bigotimes\Gamma_{\vec{a}_1})\otimes(\Gamma_2'\otimes\bigotimes\Gamma_{\vec{a}_2})$ is the obvious symmetry.

By definition, we have (for $S=QR$)

$$[\![S\{T/x\}]\!]_{\vec{x}}(\Delta,a)=\int^{\vec{b}}\int^{\Delta_1,\Delta_2}[\![Q\{T/x\}]\!]_{\vec{x}}(\Delta_1,\iota(\vec{b},a))\times[\![R\{T/x\}]\!]_{\vec{x}}(\Delta_2,\vec{b})\times{!}U^n(\Delta,\Delta_1\otimes\Delta_2).$$

By *i.h.*, we get two isomorphisms $[\![Q\{T/x\}]\!]_{\vec{x}}(\Delta_1,\iota(\vec{b},a))\cong Sub_{Q,x,T}(\Delta_1,\iota(\vec{b},a))$ and $[\![R\{T/x\}]\!]_{\vec{x}}(\Delta_2,\vec{b})\cong Sub_{R,x,T}(\Delta_2,\vec{b})$. We have our isomorphism, since isomorphisms are preserved by products and coends. Then we can conclude, since morphism actions do not change size of points and we have $\mathrm{s}\left(\vec{\beta}\right)=\mathrm{s}\left(\vec{\beta}_{\vec{a}_1}\right)+\mathrm{s}\left(\vec{\beta}_{\vec{a}_2}\right)$. ◀

Failure of Subject Reduction with Subtyping for Polyadic Terms. In Section 1 (see Footnote 2), we mentioned that polyadic terms [41, 48] fail dramatically subject reduction for intersection type distributors. Here we show a counterexample. We recall the definition of linear polyadic calculus [41] in the framework of bang calculus.

$$p,q ::= x \mid \lambda\langle x_1,\ldots,x_k\rangle.p \mid pq \mid \langle p_1,\ldots,p_k\rangle \mid \bot$$

Terms are taken up to α-equivalence and up to linearity with respect to $\bot$ (*i.e.*, $\lambda\vec{x}.\bot = p\langle\bot\rangle=\bot$, etc.[8]). The reduction $\to_{\mathsf{p}}$ is the contextual closure of the following base case:

$$(\lambda\vec{x}.p)\vec{q}\mapsto_{\mathsf{p}}\begin{cases}p\{\vec{q}/\vec{x}\} & \text{if } \mathsf{len}(\vec{q})=\mathsf{len}(\vec{x})\\ \bot & \text{otherwise.}\end{cases}$$

Since we want to link a calculus of approximants to intersection type distributors, the first thing to check is that the calculus satisfies subject reduction and expansion within our system

[8] This is slightly different from the original definition of [41], but being up to linearity simplify calculations.

$\mathcal{R}_{\to}$. Let $\zeta = \langle \vec{x}_1, \dots, \vec{x}_n \rangle$ and $\Delta = \langle \vec{a}_1, \dots, \vec{a}_n \rangle$. We write $\zeta : \Delta$ for $\vec{x}_1 : \vec{a}_1, \dots, \vec{x}_n : \vec{a}_n$. We give the following naive type assignment:

$$\dfrac{f : a' \to a}{\langle\rangle : \langle\rangle, \dots, \langle x \rangle : \langle a' \rangle, \dots, \langle\rangle : \langle\rangle \vdash x : a} \qquad \dfrac{(\zeta_i : \Gamma_i \vdash q_i)_{i=1}^{k} \qquad \eta : \Delta \to \bigotimes_{i=1}^{k} \Gamma_i}{[\eta](\bigotimes_{i=1}^{k} \zeta_i) : \Delta \vdash \langle q_1, \dots, q_k \rangle : \langle a_1, \dots, a_k \rangle}$$

$$\dfrac{\zeta \oplus \langle \vec{x} \rangle : \Delta \oplus \langle \vec{a} \rangle \vdash p : a}{\zeta : \Delta \vdash \lambda \vec{x}.p : \vec{a} \Rightarrow a} \qquad \dfrac{\zeta_0 : \Gamma_0 \vdash p : \vec{a} \Rightarrow a \qquad \zeta_1 : \Gamma_1 \vdash q \qquad \eta : \Delta \to \Gamma_0 \otimes \Gamma_1}{[\eta](\zeta_0 \otimes \zeta_1) : \Delta \vdash pq : a}$$

where in the application case the left action $[\eta]\zeta$ means only that the positions of variables in ζ are rearranged in accordance with the permutation induced by the morphism η. This is reasonable and necessary, since the morphism η can in general rearrange the position of types. This means that if $\zeta = \langle \vec{x}_1, \dots, \vec{x}_n \rangle$ and $\eta = \langle \langle \sigma_1, \vec{f_1} \rangle, \dots, \langle \sigma_n, \vec{f_n} \rangle \rangle$ then $[\eta]\zeta = \langle [\sigma_1]\vec{x}_1, \dots, [\sigma_n]\vec{x}_n \rangle$ where $[\sigma]\langle x_1, \dots, x_k \rangle = \langle x_{\sigma(1)}, \dots, x_{\sigma(k)} \rangle$ is just the left action of the symmetry group. It is easy to see that $\perp$ is not typable in the type system above.

▶ **Example 21.** We present a counter-example for the subject reduction of the former system. Take the polyadic term $p = (\lambda x.x\langle \lambda\langle\rangle.y_1\langle\rangle, \lambda\langle f \rangle.y_2\langle f \rangle \rangle)\langle \lambda \langle z_1, z_2 \rangle.z_1\langle z_2 \rangle \rangle$. This term clearly reduces to $\perp$, but it is typable in the former type system. Let $\pi =$

$$\dfrac{\dfrac{\dfrac{g : b' \to b}{\langle x \rangle : \langle b' \rangle, \langle\rangle \vdash x : b} \quad \langle\rangle : \langle\rangle, \langle y_1 \rangle : \langle \langle\rangle \Rightarrow a \rangle \vdash \lambda\langle\rangle.y_1\langle\rangle : \langle\rangle \Rightarrow a \quad \langle\rangle : \langle\rangle, \langle y_1 \rangle : \langle \langle c \rangle \Rightarrow a \rangle \vdash \lambda\langle f \rangle.y_1\langle f \rangle : \langle c \rangle \Rightarrow a}{\langle x \rangle : \langle b' \rangle, \langle y_1, y_2 \rangle : \langle \langle\rangle \Rightarrow a, \langle c \rangle \Rightarrow a \rangle \vdash x\langle \lambda\langle\rangle.y_1\langle\rangle, \lambda\langle f \rangle.y_2\langle f \rangle \rangle : a}}{\langle y_1, y_2 \rangle : \langle \langle\rangle \Rightarrow a, \langle c \rangle \Rightarrow a \rangle \vdash \lambda\langle x \rangle.x\langle \lambda\langle\rangle.y_1\langle\rangle, \lambda\langle f \rangle.y_2\langle f \rangle \rangle : \langle b' \rangle \Rightarrow a}$$

Where $c = \langle\rangle \Rightarrow a$ and $b' = \langle \langle c \rangle \Rightarrow a, \langle\rangle \Rightarrow a \rangle \Rightarrow a$ and $b = \langle \langle\rangle \Rightarrow a, \langle c \rangle \Rightarrow a \rangle \Rightarrow a$ the morphism g being of the shape $\langle \sigma, 1_{\langle\rangle \Rightarrow a}, 1_{\langle a \rangle \Rightarrow a} \rangle \Rightarrow 1$ with sigma being the obvious permutation. Consider $\rho =$

$$\dfrac{\dfrac{\langle z_1 \rangle : \langle \langle c \rangle \Rightarrow a \rangle \vdash z_1 : \langle c \rangle \Rightarrow a \qquad \langle z_2 \rangle : \langle c \rangle \vdash z_2 : c}{\langle z_1, z_2 \rangle : \langle \langle c \rangle \Rightarrow a, c \rangle \vdash z_1\langle z_2 \rangle : a}}{\vdash \lambda\langle z_1, z_2 \rangle.z_1\langle z_2 \rangle : \langle \langle c \rangle \Rightarrow a, c \rangle \Rightarrow a}$$

Now take $\pi' =$

$$\dfrac{\begin{array}{c}\pi \\ \vdots \\ \langle y_1, y_2 \rangle : \langle \langle\rangle \Rightarrow a, \langle a \rangle \Rightarrow a \rangle \vdash \lambda\langle x \rangle.x\langle \lambda\langle\rangle.y_1\langle\rangle, \lambda\langle f \rangle.y_2\langle f \rangle \rangle : \langle b' \rangle \Rightarrow a\end{array} \qquad \begin{array}{c}\rho \\ \vdots \\ \vdash \lambda\langle z_1, z_2 \rangle.z_1\langle z_2 \rangle : b'\end{array}}{\langle y_1, y_2 \rangle : \langle \langle\rangle \Rightarrow a, \langle a \rangle \Rightarrow a \rangle \vdash p : a}$$

The term p reduces to $\perp$. Indeed,

$$\begin{aligned} p &= (\lambda x.x\langle \lambda\langle\rangle.y_1\langle\rangle, \lambda\langle f \rangle.y_2\langle f \rangle \rangle)\langle \lambda\langle z_1, z_2 \rangle.z_1\langle z_2 \rangle \rangle \\ &\to_{\mathsf{p}} (\lambda\langle z_1, z_2 \rangle.z_1\langle z_2 \rangle)\langle \lambda\langle\rangle.y_1\langle\rangle, \lambda\langle f \rangle.y_2\langle f \rangle \rangle \\ &\to_{\mathsf{p}} (\lambda\langle\rangle.y_1\langle\rangle)\langle \lambda\langle f \rangle.y_2\langle f \rangle \rangle \ \to_{\mathsf{p}} \ \perp \end{aligned}$$

Therefore, $p \to_{\mathsf{p}}^{*} \perp$ and p is typable, while $\perp$ it is not. The problem relies completely in the variable rule: the subtyping feature of the system is not detected by the syntax of the standard polyadic calculus. If we want to find an appropriate term language for our system, whose elements are also approximants of ordinary bang terms, we need to take seriously the qualitative information produced by the subtyping.

The Alternating-Time μ-Calculus with Disjunctive Explicit Strategies

Merlin Göttlinger
Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
merlin.goettlinger@fau.de

Lutz Schröder
Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
lutz.schroeder@fau.de

Dirk Pattinson
The Australian National University, Canberra, Australia
dirk.pattinson@anu.edu.au

Abstract

Alternating-time temporal logic (ATL) and its extensions, including the alternating-time μ-calculus (AMC), serve the specification of the strategic abilities of coalitions of agents in concurrent game structures. The key ingredient of the logic are path quantifiers specifying that some coalition of agents has a joint strategy to enforce a given goal. This basic setup has been extended to let some of the agents (revocably) commit to using certain named strategies, as in *ATL with explicit strategies (ATLES)*. In the present work, we extend ATLES with fixpoint operators and strategy disjunction, arriving at the *alternating-time μ-calculus with disjunctive explicit strategies (AMCDES)*, which allows for a more flexible formulation of temporal properties (e.g. fairness) and, through strategy disjunction, a form of controlled non-determinism in commitments. Our main result is an ExpTime upper bound for satisfiability checking (which is thus ExpTime-complete). We also prove upper bounds QP (quasipolynomial time) and NP $\cap$ coNP for model checking under fixed interpretations of explicit strategies, and NP under open interpretation. Our key technical tool is a treatment of the AMCDES within the generic framework of coalgebraic logic, which in particular reduces the analysis of most reasoning tasks to the treatment of a very simple *one-step logic* featuring only propositional operators and next-step operators without nesting; we give a new model construction principle for this one-step logic that relies on a set-valued variant of first-order resolution.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Computing methodologies → Multi-agent systems

Keywords and phrases Alternating-time logic, multi-agent systems, coalitional strength

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.26

Funding *Merlin Göttlinger*: Work performed under the DFG Project *Reconstructing Arguments from Noisy Text (RANT)*, SCHR 1118/14-1 / SCHR 1118/14-2.
Lutz Schröder: Work performed under the DFG Project *RANT*, SCHR 1118/14-1 / SCHR 1118/14-2.

1 Introduction

Alternating-time temporal logic (ATL) [1] extends computation tree logic (CTL) with path quantifiers $\langle\langle A\rangle\rangle$ read "coalition A of agents has a (long-term) joint strategy to enforce". It is embedded into the *alternating-time μ-calculus (AMC)*, which instead of path quantifiers, features nested least and greatest fixpoints alongside the next-step coalition modalities $\langle\langle A\rangle\rangle \bigcirc$ ("A can enforce in the next step"). The AMC is strictly more expressive than ATL, e.g. supports fairness constraints.

Coalitional power in ATL and the AMC is measured without any restrictions on the moves chosen by the opponents. There has been interest in extensions of ATL where the power of the opponents can be constrained, e.g. by committing some of them to a particular strategy, allowing for statements such as "no matter what the other network actors do, Alice and Bob can collaborate to exchange keys via Server S provided that S adheres to the protocol". One such extension is provided in *ATL with explicit strategies (ATLES)* [29], which has path quantifiers $\langle\langle A\rangle\rangle_\rho$ additionally parametrized over a commitment ρ of some agents to given named strategies, read "provided that the commitments ρ are kept, A can enforce ... ". This extension has substantial impact on expressiveness; e.g. unlike in basic ATL, the semantics of ATLES over history-free strategies differs from the one over history-dependent strategies.

Restricting opponents to fixed moves is, of course, quite drastic; as noted already in the conclusion of Walther [28, Chapter 4], it is desirable to allow for more permissive restrictions where the opponents can still pick among several designated moves, as in "Alice has a strategy to get her print job executed if Bob either cancels his large print job or splits it into several smaller ones". In the present paper, we introduce such an extension with disjunctive commitments. Additionally, we include full support for least and greatest fixpoint operators, with associated gains in expressivity analogous to the extension from ATL to the AMC. We thus arrive at the *alternating-time μ-calculus with disjunctive explicit strategies (AMCDES)*.

Our main result on this logic is that satisfiability checking remains only ExpTime-complete (i.e. no harder than the AMC, or in fact than basic ATL or even CTL). We note also that (following a distinction made also in work on ATLES [29]) model checking is in quasipolynomial time QP and in NP ∩ coNP under fixed interpretation of explicit strategies (matching the best known bounds for the AMC and in fact even the plain relational μ-calculus), and in NP under open interpretation; these results are obtained by fairly straightforward adaptation of results on the AMC [11], and therefore discussed in full only in the appendix. We obtain our results by casting the AMCDES as an instance of *coalgebraic logic* [5], a unifying framework for modal and temporal logics. The driving principle of coalgebraic logic is to reduce reasoning tasks to the analysis of a simple *one-step logic*, whose formulae employ only Boolean connectives and a single layer of next-step modalities [22, 4, 11]. In particular, the automata- and game-theoretic machinery needed for the treatment of fixpoint logics is entirely encapsulated in results on the coalgebraic μ-calculus [4, 11]. The actual technical work then lies in providing algorithms, axiomatizations, and model constructions for the one-step logic of AMCDES, still posing substantial challenges due to nested quantification over strategies. The model construction principle for the one-step logic that we employ is based on a set-valued variant of first-order resolution that we introduce here, along with an associated notion of equationally complete model that we use to move from (generally infinite) Herbrand universes to finite models; this principle is the key to supporting strategy disjunction.

Related Work. Many ATL extensions are concerned with commitments of agents to strategies. Besides ATL with explicit strategies (ATLES), this includes, e.g., counterfactual ATL [26], which differs from ATLES by making commitments irrevocable. ATL with actions (ATL-A) [30] has per-agent disjunctive commitments (while the AMCDES allows disjunctions over joint commitments). ATL-A admits polynomial-time model checking; satisfiability checking is not considered (it would be somewhat simpler than in the present setting, as in ATL-A all actions are named, and hence known in advance). ATL with explicit actions (ATLEA) [12] features commitments of agents to a given action at only the current world, and has a fairly straightforward satisfiability-preserving embedding into the AMCDES.

Various forms of *strategy logic* [3, 15, 16] possibly contain ATL* with disjunctive explicit strategies (but presumably not the AMCDES or even the AMC, as they lack fixpoint operators); they tend to be computationally much harder than the AMCDES. Goranko and Ju [7] discuss various forms of conditional strategic modalities, one of which (O_{dd}) is similar in spirit to our strategy disjunction in that it restricts the moves of the opposition, however not to given named moves but rather to moves enforcing a given goal; their main technical result is a Hennessy-Milner style expressiveness theorem. De Nicola and Vandraager [18] consider disjunction of named actions in labelled transition systems, which in that setting can be encoded into next-modalities for single actions using logical disjunction.

Organization. We introduce the syntax and the semantics of the alternating-time μ-calculus with disjunctive explicit strategies (AMCDES) in Section 2. After recalling the requisite principles of coalgebraic logic in Section 3 we introduce the method of set-valued first-order resolution in Section 4. We illustrate these methods on the basic AMC in Section 5, and establish our main results on satisfiability checking for the AMCDES in Section 6.

2 AMC With Disjunctive Explicit Strategies

We proceed to introduce the syntax and semantics of the alternating-time μ-calculus with disjunctive explicit strategies (AMCDES). As indicated in Section 1, the logic is inspired by ATL with explicit strategies (ATLES) [29]. We deviate from the ATLES syntax in that we express (disjunctive) commitments of agents by means of names for strategies in the syntax. Also, we shorten the ATL syntax for next-step operators from $\langle\langle C\rangle\rangle \bigcirc$ (“C can enforce in the next step that …”) to $[C]$ as in coalition logic [20]. We thus arrive at modalities $[C,O]$ where O is a set of named joint strategies for agents in a further coalition D of agents restricted in their choice of strategies, disjoint from C, read “if the agents in D use one of the joint strategies in O, then C can enforce that …”. The dual modality $\langle C,O\rangle$ is read “even if the agents in D are limited to the joint strategies in O, C cannot prevent that …”. Formally, our syntax is defined as follows.

▶ **Definition 2.1.** The syntax of the AMCDES is parametrized over a set At of (propositional) *atoms*, V of *variables*, a finite set Σ of agents (for technical simplicity, assumed to be linearly ordered), and sets M_j of *explicit strategies* (i.e. names for strategies) per agent j; *we fix these data from now on.* A *coalition* is a subset of Σ. We also (and mainly) refer to explicit strategies as *explicit moves.* We write $M_D = \prod_{j\in D} M_j$ for the set of *joint explicit moves* of a coalition D. Formulae ϕ,ψ are then given by the grammar

$$\phi,\psi ::= p \mid \neg p \mid x \mid \top \mid \bot \mid \phi\wedge\psi \mid \phi\vee\psi \mid [C,O]\,\phi \mid \langle C,O\rangle\,\phi \mid \mu x.\,\phi \mid \nu x.\,\phi$$

where $x\in V$, $p\in\mathsf{At}$, and $C\subseteq\Sigma$, i.e. a *coalition.* We generally write $\overline{C}=\Sigma\setminus C$. Moreover, $O\subseteq M_D$ is a set of joint explicit moves, called a *disjunctive explicit strategy* (or *move*), for some coalition D, disjoint from C, that we denote by $Ag(O)$. We call a modality $[C,O]$ or $\langle C,O\rangle$ a *grand coalition modality* if $C\cup Ag(O)=\Sigma$, and *non-disjunctive* if $|O|=1$, in which case we often omit set brackets and just write O as its single element. We restrict grand coalition modalities to be non-disjunctive (cf. Remark 2.8). As usual, μ and ν take least and greatest fixpoints, respectively. Negation $\neg$ is not included but can be defined in the standard way, taking negation normal forms. The AMC with explicit strategies (AMCES) is the fragment of the AMCDES allowing only non-disjunctive modalities.

The AMCDES thus subsumes both the standard AMC [1] (with $[C]$ corresponding to $[C, O]$ with $Ag(O) = \emptyset$) and the history-free variant of ATLES [29] (which as we will detail in Remark 2.7 is the variant to which previous technical results refer).

▶ **Example 2.2.** The formula indicated in the introduction,

$$[\mathsf{Alice}, (\mathsf{Bob}\colon \{\mathsf{cancelPrint}, \mathsf{splitPrint}\})]\, \mathsf{printed}$$

says (using hopefully self-explanatory human-readable syntax for disjunctive explicit moves) that "Alice has a strategy to have her print job executed, provided that Bob opts to either cancel his print job or to split it into smaller jobs". The fixpoint formula

$$\nu x.\, \neg\mathsf{corrupted} \wedge [\mathsf{ECC}, (\mathsf{Env}\colon \{\mathsf{0\text{-}flips}, \mathsf{1\text{-}flip}\})]\, x$$

expresses that ECC memory can ensure that the stored data is not corrupted provided that in each cycle the environment flips either one or zero bits. The formula

$$\nu x.\, \neg\mathsf{intrusion} \wedge \langle \mathsf{Attacker}, (\mathsf{IPS}\colon \{\mathsf{dropPackage}, \mathsf{blockIP}\})\rangle\, x$$

expresses that "No matter what an attacker tries, the intrusion prevention system can always drop suspicious packets or block his IP address to prevent illegitimate access to company resources".

▶ Remark 2.3. One can encode an extension ATLDES of ATL with disjunctive explicit strategies into the AMCDES, e.g. defining $\langle\langle C, O\rangle\rangle(G\,\phi)$ "C can enforce that ϕ always holds, provided that $Ag(O)$ are committed to play strategies in O" as

$$\langle\langle C, O\rangle\rangle(G\,\phi) := \nu x.\, \phi \wedge [C, O]\, x.$$

The AMCDES is more expressive than ATLDES in this sense; e.g. for $C = \{\mathsf{client}\}$ and $O = (\mathsf{server}\colon \{\mathsf{protocol}, \mathsf{recover}\})$ the formula $\nu x.\, \mu y.\, (\mathsf{granted} \wedge [C, O]\, x) \vee [C, O]\, y$ says that "client can enforce that his requests are granted infinitely often, provided that server always either keeps to the protocol or immediately recovers when failures occur" (a specification that may, of course, hold or fail in a given system).

Note that the definition of $\langle\langle C, O\rangle\rangle$ allows $Ag(O)$ to choose their joint move from O anew in each step, like in the fixpoint formulae of Example 2.2, which in fact belong to the ATLDES fragment of the AMCDES. To illustrate that this is really the reasonable choice of a semantics for ATLDES (as opposed to letting O choose only in the beginning of a play), consider a situation where players K (*Kangaroo*) and M (*Marc-Uwe*) [14] play rock-paper-scissors (R, P, S) for an indefinite number of rounds, say to determine daily who does the dishwashing, until someone quits. Let the model include memory for the moves in the previous round, and atoms p "at least two rounds have been played" and k "K won the previous round". Consider the ATL with disjunctive explicit strategies (ATLDES) formula

$$\mathsf{rigged} = \langle\langle K, (M\colon \{R, P, S\})\rangle\rangle\, G(p \to k)$$

"K wins all rounds after the first if M keeps playing". In ATLDES, rigged does not hold in the model, as one would expect. If M could make his choice of R, P, S only once (in reality, sadly, he does just that [13]), then rigged would in fact hold.

We proceed to define the semantics, which is based on concurrent game structures [1] extended with interpretations of explicit moves.

▶ **Notation 2.4.** For $k \in \mathbb{N}$, we write $[k] = \{1, \dots, k\}$. For $C \subseteq \Sigma$ and a tuple $(k_j)_{j \in C} \in \mathbb{N}^C$, we put $[k_C] = \prod_{j \in C}[k_j]$. Given $m \in [k_C]$ and $D \subseteq C$, we write $m|_D$ for the restriction of m to an element of $[k_D]$. We write $n \sqsubseteq m$ if $n = m|_{Ag(n)}$, and $n =_\sqcap m$ if $n|_{Ag(n) \cap Ag(m)} = m|_{Ag(n) \cap Ag(m)}$. We write $\mathcal{P}X$ for the powerset of a set X.

▶ **Definition 2.5.** A *concurrent game structure with explicit strategies (CGSES)* is a tuple (W, k, v, f, ι) consisting of

- a finite set W of *states*,
- for each agent j and each state w, a natural number $k_j^w \geq 1$ determining the set of *moves* available to agent j at state w to be $[k_j^w]$,
- for each state $w \in W$,
  - a set $v(w) \subseteq \mathsf{At}$ of propositional atoms true at w,
  - an *outcome function* $f^w : [k_\Sigma^w] \to W$, and
  - for each agent j, a *move interpretation* $\iota_j^w : M_j \to [k_j^w]$.

For a joint explicit move $m \in M_D$, we just write $\iota^w(m)$ for the joint move with components $\iota_j^w(m_j)$ for $j \in D$. We use function image notation $\iota^w[O]$ to denote the result of applying ι^w to each joint move in the set O. The semantics of the AMCDES is then defined by assigning to each formula ϕ an extension $[\![\phi]\!]_S^\sigma \subseteq Q$, which depends on a CGSES $S = (W, k, v, f, \iota)$ and a valuation $\sigma : V \to \mathcal{P}W$. The propositional cases are standard (e.g. $[\![p]\!]_S^\sigma = \{w \in W \mid p \in v(w)\}$, $[\![x]\!]_S^\sigma = \sigma(x)$, $[\![\top]\!]_S^\sigma = W$, and $[\![\phi \wedge \psi]\!]_S^\sigma = [\![\phi]\!]_S^\sigma \cap [\![\psi]\!]_S^\sigma$). The remaining clauses are

$$\begin{aligned}
[\![[C,O]\,\phi]\!]_S^\sigma &= \{w \in W \mid \exists m_C \in [k_C^w].\, \forall m_\Sigma \in [k_\Sigma^w]. \\
&\qquad\qquad (m_C \sqsubseteq m_\Sigma \wedge m_\Sigma|_{Ag(O)} \in \iota^w[O]) \Rightarrow f^w(m_\Sigma) \in [\![\phi]\!]_S^\sigma\} \\
[\![\langle C,O\rangle\,\phi]\!]_S^\sigma &= \{w \in W \mid \forall m_C \in [k_C^w].\, \exists m_\Sigma \in [k_\Sigma^w]. \\
&\qquad\qquad m_C \sqsubseteq m_\Sigma \wedge m_\Sigma|_{Ag(O)} \in \iota^w[O] \wedge f^w(m_\Sigma) \in [\![\phi]\!]_S^\sigma\} \\
[\![\mu x.\, \phi(x)]\!]_S^\sigma &= \bigcap\{B \subseteq W \mid [\![\phi(x)]\!]_S^{\sigma[x \mapsto B]} \subseteq B\} \\
[\![\nu x.\, \phi(x)]\!]_S^\sigma &= \bigcup\{B \subseteq W \mid B \subseteq [\![\phi(x)]\!]_S^{\sigma[x \mapsto B]}\}
\end{aligned}$$

where $\sigma[x \mapsto B]$ denotes σ updated to return B on input x; and $[\![\langle C,O\rangle\,\phi]\!]_S^\sigma = [\![\neg\,[C,O]\,\neg\phi]\!]_S^\sigma$. That is, μ and ν take least and greatest fixpoints according to the Knaster-Tarski fixpoint theorem. At a state w, $[C,O]\,\phi$ holds if the agents in C have a joint move such that a state satisfying ϕ is reached no matter what the other agents do, as long as the agents in $Ag(O)$ play one of the joint moves in O. Dually, $\langle C,O\rangle\,\phi$ holds at w if whatever the agents in C do, the other agents have a joint move that leads to an outcome in ϕ and in which the joint move of $Ag(O)$ is in O.

▶ Remark 2.6. In the modal operators $[C,O]$, $Ag(O)$ is in opposition to C. One may envision an alternative setup where $Ag(O)$ is instead made a part of C. However, then $[C,O]\,\phi$ would become equivalent to $\bigvee_{m \in O}[C,\{m\}]\,\phi$, hence expressible already in ATLES. We thus opt for our present more expressive version where $Ag(O)$ and C are disjoint. Note that $[C,O]\,\phi$ then is *not* equivalent to $\bigwedge_{m \in O}[C,\{m\}]\,\phi$: The latter formula allows C to use different moves against each $m \in O$, while in $[C,O]\,\phi$, the *same* joint move of C must work against every $m \in O$.

▶ Remark 2.7. The above semantics uses history-free strategies (i.e. ones that look only at the present state, not the history of previously visited states). While basic ATL is insensitive to whether it is interpreted over history-free or history-dependent strategies [1], ATLES does distinguish these semantics [29]. Although this may not be always apparent from the phrasing, all technical results on ATLES in Walther et al. [29] are meant to apply to the semantics over history-free strategies only[1] (in particular the fixpoint unfolding axioms [29, Figure 1] clearly hold only over the history-free semantics). Note that the basic AMC, which the AMCDES extends, similarly is interpreted over history-free strategies (and nevertheless includes ATL*, which is history-dependent [1]).

▶ Remark 2.8. The interdiction of proper strategy disjunction in grand coalition modalities is needed (only) for the upper bound on satisfiability checking (Section 6); our results on model checking (Section 2) would actually not need this restriction. The fragment we term AMCES in Definition 2.1 does include grand coalition modalities with (non-disjunctive) explicit strategies. It is hence more permissive on these modalities than the original version of ATLES [29], where the set of agents is made variable, which for purposes of satisfiability is equivalent to excluding grand coalition modalities.

We note that the axiomatization we present later and its completeness proof become much simpler if one excludes the grand coalition completely (like, effectively, in ATLES): E.g. in the rule (C) for basic coalition logic / ATL (Section 5), the literals $\langle\Sigma\rangle\, c_j$ disappear; and in the proof of one-step tableau completeness (Theorem 5.1), one can, in this simplified setting, just use a single move $\bot$ as witness for all $\langle C_j\rangle\, c_j$ in Ξ, using non-determinism to ensure satisfaction of the $\langle C_j\rangle\, c_j$. This is discussed in detail in Appendix B.

Model Checking

Walther et al. [29] consider two variants of the model checking problem that differ on whether the interpretation of explicit strategies is considered part of the model (*fixed*) or to be found by the model checking algorithm (*open*). They show for ATLES that if strategies are restricted to be history-free, then the problem is P-complete under fixed interpretation, and NP-complete under open interpretation, with the upper bound being by straightforward guessing of history-free strategies. The complexity for the history-dependent variant remains open.

We obtain upper bounds for model checking in the AMCDES using generic results on the coalgebraic μ-calculus [11]:

▶ **Theorem 2.9.** *Model checking for the full AMCDES is in NP $\cap$ coNP as well as in QP under fixed interpretation of explicit strategies, and in NP under open interpretation.*

We defer a summary of the requisite results in coalgebraic logic and the proof of Theorem 2.9 to Appendix A, as the details are mostly by simple adaptation from the AMC [11].

3 Preliminaries: Coalgebraic Logic

We will employ the machinery of coalgebraic logic to obtain our main complexity results; we recall basic definitions and tools, using the standard AMC as our running example.

Coalgebraic logic [5] is a uniform framework for modal and temporal logics interpreted over state-based systems. It parametrizes the **semantics** of logics over the type of such systems, encapsulated in a *functor* F on the category of sets. Such a functor assigns to each set X a set FX and to each map $f : X \to Y$ a map $Ff : FX \to FY$, preserving identities and composition. We think of the elements of FX as structured collections over X. Systems are then F-*coalgebras*, i.e. pairs (W, γ) consisting of a set W of *states* and a *transition map*

[1] Personal communication with the authors

$\gamma : W \to FW$, which thus assigns to each state a structured collection of successors. Our leading example is the functor G that maps a set X to the set

$$\mathsf{G}X = \{((k_j)_{j\in\Sigma}, f) \mid (k_j) \in \mathbb{N}^{\Sigma}_{\geq 1}, f : (\prod_{j\in\Sigma}[k_j]) \to X\}$$

of *one-step games* over X. G-Coalgebras are essentially concurrent game structures (CGSs) [1] without the interpretation of propositional atoms, as they assign to each state numbers k_j of available moves for the agents and an outcome function f. Propositional atoms are covered by extending G to $\mathsf{G}_{\mathsf{p}}X = \mathcal{P}\mathsf{At} \times \mathsf{G}X$; although the logic becomes trivial without propositional atoms, we mostly elide their explicit treatment, which is straightforward and can be dealt with using fusion results in coalgebraic logic [23]. To obtain CGSESs, we extend G to the functor G_{ES} with $\mathsf{G}_{\mathsf{ES}}X$ consisting of *one-step games with explicit strategies* $((k_j), f, \iota)$ over X, where $((k_j), f)$ is a one-step game over X and $\iota_j \colon M_j \to [k_j]$ (for $j \in \Sigma$) interprets explicit strategies; we use the same notation for ι as introduced for ι^w in Section 2.

The **syntax** of coalgebraic logics is then parametrized over the choice of a set Λ of (next-step) *modal operators* with assigned finite arities; nullary modalities are just propositional atoms. For readability, we assume in the technical treatment that all modalities are unary. We require that for every $\heartsuit \in \Lambda$ there is a *dual* operator $\overline{\heartsuit} \in \Lambda$. The *coalgebraic μ-calculus* [4] over Λ then has formulae ϕ, ψ given by the grammar

$$\phi, \psi ::= \top \mid \bot \mid x \mid \phi \wedge \psi \mid \phi \vee \psi \mid \heartsuit\, \phi \mid \mu x.\, \phi \mid \nu x.\, \phi$$

where x ranges over a reservoir V of *fixpoint variables*, and $\heartsuit$ over Λ. The operators μ and ν take least and greatest fixpoints, respectively. Again, negation is definable. We assume a representation of the modalities in Λ as strings over some alphabet, with an ensuing notion of *representation size* for formulae and modalities.

Over F-coalgebras, a modal operator $\heartsuit \in \Lambda$ is interpreted by assigning to it a *predicate lifting* $[\![\heartsuit]\!]$, which is a family of maps $[\![\heartsuit]\!]_X$, indexed over all sets X, that assign to each subset $Y \subseteq X$ a subset $[\![\heartsuit]\!]_X(Y) \subseteq FX$, subject to a naturality condition. To enable fixpoint formation, we require $[\![\heartsuit]\!]_X$ to be monotone w.r.t. subset inclusion. Moreover, we require predicate liftings to respect duals, i.e. $[\![\overline{\heartsuit}]\!]_X(Y) = FX \setminus [\![\heartsuit]\!]_X(X \setminus Y)$. Given an F-coalgebra $C = (W, \gamma)$ and a valuation $\sigma : V \to \mathcal{P}W$, the semantic clauses defining the extension $[\![\phi]\!]^{\sigma}_C \subseteq W$ of a formula ϕ are then the standard ones for the Boolean connectives; μ and ν take least and greatest fixpoints in the same way as made explicit for the AMCDES in Section 2; and

$$[\![\heartsuit\, \phi]\!]^{\sigma}_C = \gamma^{-1}[[\![\heartsuit]\!]_W([\![\phi]\!]^{\sigma}_C)].$$

We fix the data F, Λ, $[\![\heartsuit]\!]$ for the remainder of this section.

▶ **Example 3.1.** The AMC is cast as a coalgebraic μ-calculus by interpreting the modality $[C]$ over the functor G by the predicate lifting

$$[\![[C]]\!]_X(Y) = \{((k_j), f) \in \mathsf{G}X \mid \exists m_C \in [k_C].\, \forall m \in [k_\Sigma].\, m_C \sqsubseteq m \Rightarrow f(m) \in Y\}$$

(using notation introduced in Section 2). The more general modalities $[C, O]$ of AMCDES are interpreted by a predicate lifting that correspondingly lifts a predicate Y on X to the set of all one-step games with explicit strategies $((k_j), f, \iota) \in \mathsf{G}_{\mathsf{ES}}X$ such that there exists a joint move $m_C \in [k_C]$ such that $f(m) \in Y$ for all $m \in [k_\Sigma]$ such that $m_C \sqsubseteq m$ and $\iota(n) \sqsubseteq m$ for some $n \in O$.

Satisfiability checking in coalgebraic logics can be based on the provision of a complete set of tableau rules for the next-step modal operators [22, 4]. The basic example of such a rule is the tableau rule $\Box a_1, \dots, \Box a_n, \Diamond b / a_1, \dots, a_n, b$ for standard modal logic, which says essentially that in order to satisfy $\Box a_1 \wedge \dots \wedge \Box a_n \wedge \Diamond b$, we need to generate a successor state satisfying $a_1 \wedge \dots \wedge a_n \wedge b$. Formal definitions are as follows.

▶ **Definition 3.2** (One-step tableau rules)**.** Fix a supply V of *(propositional) variables*, serving as placeholders for formulae in rules. A *(monotone) one-step (tableau) rule* has the form

$$\frac{\Phi}{\Theta_1 \mid \dots \mid \Theta_n} \qquad (n \geq 0)$$

where the *conclusions* $\Theta_1, \dots, \Theta_n$ are finite subsets of V, read as finite conjunctions, and the *premiss* Φ is a finite subset of the set $\Lambda(\mathsf{V}) = \{\heartsuit a \mid \heartsuit \in \Lambda, a \in \mathsf{V}\}$ of *modal atoms*, also read conjunctively; additionally, we require that Φ mentions each variable at most once, and the Θ_i mention only variables occurring in Φ. Given a set X and a $\mathcal{P}X$-valuation $\tau : \mathsf{V} \to \mathcal{P}X$, we interpret such a Θ_i as $[\![\Theta_i]\!]\tau = \bigcap_{a \in \Theta_i} \tau(a)$, and Φ as $[\![\Phi]\!]\tau = \bigcap_{\heartsuit a \in \Phi} [\![\heartsuit]\!]_X(\tau(a)) \subseteq FX$.

The rule $\Phi/\Theta_1 \mid \dots \mid \Theta_n$ is *one-step tableau sound* if $[\![\Theta_i]\!]\tau \neq \emptyset$ for some i whenever $[\![\Phi]\!]\tau \neq \emptyset$. Let $\mathcal{R}$ be a set of one-step tableau rules, closed under injective renaming of variables. Then $\mathcal{R}$ is *one-step tableau complete* if the following condition holds: For all X, $\tau : \mathsf{V} \to \mathcal{P}X$, and $\Xi \subseteq \Lambda(\mathsf{V})$, whenever for each rule $\Phi/\Theta_1 \mid \dots \mid \Theta_n \in \mathcal{R}$ such that $\Phi \subseteq \Xi$, we have $[\![\Theta_i]\!]\tau \neq \emptyset$ for some i, then $[\![\Xi]\!]\tau \neq \emptyset$.

We will give one-step tableau sound and complete sets of rules for the AMCDES in Section 6. To obtain complexity results, rule sets formally need to be *ExpTime-tractable*, meaning that rule matches are encodable as strings over some alphabet such that all rule matches to a given set of formulae can be represented by polynomially sized codes and moreover basic operations on codes (well-formedness check, check for rule matching, access to conclusions) can be performed in exponential time [22, 4]; we refrain from elaborating details, as all rule sets we consider here will be clearly computationally harmless. The main benefit that we draw from these rule sets is the following generic upper complexity bound.

▶ **Theorem 3.3** (Satisfiability checking [4])**.** *If a coalgebraic μ-calculus admits an ExpTime-tractable one-step tableau complete set of one-step tableau sound rules, then its satisfiability problem is in ExpTime.*

In the algorithm underlying the above theorem, one-step rules combine with standard tableau rules for propositional and fixpoint operators. The arising tableaux need to be checked for bad branches (where least fixpoints are unfolded indefinitely) using dedicated parity automata, which combine with the tableau to form the *tableau game*, a parity game that is won by Eloise iff the target formula is satisfiable.

4 Set-Valued First-Order Resolution

For use in completeness proofs of modal rules, we next introduce *set-valued first-order resolution*, an adaptation of the standard first-order resolution method [6] to a logic of *outcome models* $\mathcal{G} = ((S_j)_{j \in \Sigma}, f, W, [\![-]\!])$ where the S_j are sets and W is a finite set, $[\![-]\!]$ interprets sorted algebraic operations over the S_j, and $f : \left(\prod_{j \in \Sigma} S_j\right) \to W$ is an outcome function. One-step games in $\mathsf{G}W$ are (operation-free reducts of) outcome models where the S_j are finite; for the time being, we allow infinite S_j for readability, explaining in the proof sketches in Sections 5 and 6 how finiteness can be regained. Formulae of *set-valued first-order* *logic* are clause sets formed over literals of the form $A(\bar{t})$ where $A \subseteq W$ and $\bar{t}$ is an Σ-tuple of terms (i.e. a clause is a finite set of literals, read disjunctively, and a clause set is a finite set of clauses, read conjunctively). Terms live in a sorted setting with one sort j (interpreted as S_j) for each agent j, and the j-th term in $\bar{t}$ has sort j. Terms are built from sorted variables and function symbols with given sort profiles (e.g. $g : 1 \times 0 \to 2$ takes moves of agents 1 and 0, and produces a move of agent 2) in the standard way, ensuring well-sortedness. Function symbols are interpreted as sorted functions on the S_j, respecting the sort profile; this induces an interpretation of (tuples of) terms depending on sort-respecting valuations of the variables as usual. We write $[\![\bar{t}]\!]\eta$ for the interpretation of a tuple $\bar{t}$ of terms under a valuation η. An outcome model $\mathcal{G}$ as above *satisfies* a literal $A(\bar{t})$ under a valuation η (notation: $\mathcal{G}, \eta \models A(\bar{t})$) if $f([\![\bar{t}]\!]\eta) \in A$, and $\mathcal{G}$ satisfies a clause Γ under η (notation: $\mathcal{G}, \eta \models \Gamma$) if $\mathcal{G}, \eta \models A(\bar{t})$ for some literal $A(\bar{t})$ in Γ. Finally, $\mathcal{G}$ *satisfies* a clause Γ (notation: $\mathcal{G} \models \Gamma$) if $\mathcal{G}, \eta \models \Gamma$ for every valuation η. A clause set is satisfiable if there exists an outcome model that satisfies all its clauses. We will generate clauses from modal atoms in $\Lambda(\mathsf{V})$ (Definition 3.2); e.g. given a $\mathcal{P}W$-valuation $\tau : \mathsf{V} \to \mathcal{P}W$, modalized atoms $[C]\, a$ and $\langle C \rangle\, a$ induce singleton clauses of the form

$$\{\tau(a)(e_C, x_{\overline{C}})\} \qquad \text{(for } [C]\, a\text{)} \qquad (1)$$

$$\{\tau(a)(x_C, g_{\overline{C}}(x_C))\} \qquad \text{(for } \langle C \rangle\, a\text{)} \qquad (2)$$

respectively, where x_C, $x_{\overline{C}}$ are tuples of variables (implicitly universally quantified, and representing moves for the agents in C and $\overline{C}$, respectively); e_C is a family of Skolem constants witnessing the ability of C to force a; and $g_{\overline{C}}$ is a family of Skolem functions producing countermoves $g_{\overline{C}}(x_C)$ for the agents in $\overline{C}$ that keep C from enforcing $\neg a$ using x_C. Of course these symbols are fresh so that clauses induced by different modalized atoms have disjoint sets of function symbols and variables, which we will later distinguish via superscripts in proofs.

We implicitly normalize clauses to mention each tuple of terms at most once (rewriting $A(\bar{t}), B(\bar{t})$ into $(A \cup B)(\bar{t})$), and operate on clauses using the *(set-valued) resolution rule*

$$(SR)\ \frac{\Gamma, A(\bar{t}) \qquad B(\bar{u}), \Delta}{\Gamma\sigma, (A \cap B)(\bar{t}\sigma), \Delta\sigma}$$

where σ is the most general unifier (mgu) of $\bar{t}$ and $\bar{u}$, with variables in the premises made disjoint by suitable renaming; as usual, we write "," for union of clauses and omit set brackets around singleton clauses (so $\Gamma, A(\bar{t})$ is shorthand for $\Gamma \cup \{A(\bar{t})\}$). A clause is *blatantly inconsistent* if all its literals are of the form $\emptyset(\bar{t})$. A clause set ϕ is *blatantly inconsistent* if it contains a blatantly inconsistent clause, and *inconsistent* if a blatantly inconsistent clause can be derived from it using the resolution rule; otherwise, ϕ is *consistent*.

Recall that unification can fail either due to a *clash*, i.e. when terms with distinct head symbols need to be unified, or at the *occurs check*, which happens when a variable needs to be unified with a term that contains it. In particular, this happens in clauses (2) associated to diamonds: E.g. the modal atoms $\langle\{0\}\rangle\, a$ and $\langle\{1\}\rangle\, b$ generate clauses $\{\tau(a)(x_0, g^1_1(x_0))\}$ and $\{\tau(b)(g^2_0(x_1), x_1)\}$, whose (tuples of) argument terms fail to unify since no substitution solves $x_0 = g^2_0(g^1_1(x_0))$.

Set-valued propositional resolution in *set-valued propositional logic* simplifies the above setup by replacing tuples $\bar{t}$ of terms in literals $A(\bar{t})$ with elements y of some index set Y; models are then just functions $f\colon Y \to W$, and f *satisfies* a literal $A(y)$ if $f(y) \in A$. The resolution rule is just like the above but of course does not involve unification and substitution, i.e. just derives $\Gamma, (A \cap B)(y), \Delta$ from $\Gamma, A(y)$ and $B(y), \Delta$.

▶ **Theorem 4.1** (Soundness and completeness of set-valued resolution). *A clause set in set-valued propositional (first-order) logic is satisfiable iff it is consistent under set-valued propositional (first-order) resolution.*

Proof sketch. Soundness ("only if") is clear (see Appendix B). Completeness ("if") of the propositional variant depends on W being finite. It proceeds via maximally consistent clause sets (MCS) and a Hintikka lemma stating in particular that an MCS containing $(A \cup B)(y)$ must also contain one of $A(y), B(y)$. Completeness of the first-order variant is by adaptation of the completeness proof for standard first-order resolution, going via Herbrand models (i.e. models having the set of ground terms as the carrier set) and reduction to completeness of set-valued propositional resolution. ◀

Of course, the Herbrand models constructed in the proof of Theorem 4.1 are in general infinite. For purposes of constructing finite models, we identify a property of "sufficient completeness" of a model for a set of terms.

▶ **Definition 4.2.** A set $\mathcal{T}$ of (tuples of) terms is *closed under unification* if whenever $t, s \in \mathcal{T}$ are unifiable and σ is an mgu of t, s, then $u\sigma \in \mathcal{T}$ for every $u \in \mathcal{T}$.

▶ Remark 4.3. If $\mathcal{T}$ is closed under unification, then $\mathcal{T}$ is in particular closed under injective renaming of variables: For $u \in \mathcal{T}$, every injective renaming σ is an mgu of u, u, so that $u\sigma \in \mathcal{T}$.

We will treat tuples of terms like terms in the following, in particular mentioning equations between tuples of terms and unifiers of such equations; this is to be understood via componentwise equality in the evident sense.

▶ **Definition 4.4.** A *solution* of an equation $t = s$ in an outcome model $\mathcal{G}$ is a valuation η such that $[\![t]\!]\eta = [\![s]\!]\eta$ in $\mathcal{G}$. Let $\mathcal{T}$ be a set of tuples of terms. We say that $\mathcal{G}$ is *$\mathcal{T}$-equationally complete* if whenever an equation $\bar{t} = \bar{s}$ with $\bar{t}, \bar{s} \in \mathcal{T}$ has a solution in $\mathcal{G}$, then $\bar{t}, \bar{s}$ are unifiable, and the mgu σ of $\bar{t}, \bar{s}$ is a *most general solution* of $\bar{t} = \bar{s}$ in $\mathcal{G}$, i.e. every solution η of $\bar{t} = \bar{s}$ in $\mathcal{G}$ has the form $\eta(x) = [\![\sigma(x)]\!]\eta'$ for some valuation η'; we then say briefly that η *factorizes through* σ.

▶ **Theorem 4.5.** *Let $\mathcal{T}$ be a set of tuples of terms that is closed under unification, and let $\mathcal{G}$ be $\mathcal{T}$-equationally complete. Let ϕ be a clause set such that $\bar{t} \in \mathcal{T}$ for every literal $B(\bar{t})$ occurring in ϕ. If ϕ is consistent under set-valued first-order resolution, then ϕ is satisfiable over $\mathcal{G}$.*

Proof. By completeness of set-valued propositional resolution (Theorem 4.1), it suffices to show that the clause set $\phi^{\mathcal{G}}$ consisting of all instances over $\mathcal{G}$ of clauses in ϕ is consistent under set-valued propositional resolution. Formally, an instance $[\![\Gamma]\!]\eta$ over $\mathcal{G}$ of a clause Γ is induced by an A-valuation η, and given as

$$[\![\Gamma]\!]\eta = \{B([\![\bar{t}]\!]\eta) \mid B(\bar{t}) \in \Gamma\}.$$

Since $\mathcal{T}$ is closed under unification, we can assume w.l.o.g. that ϕ is closed under set-valued first-order resolution (since all terms that appear when closing ϕ under resolution remain in $\mathcal{T}$); then it suffices to show that $\phi^{\mathcal{G}}$ is closed under set-valued propositional resolution, since ϕ and, hence, $\phi^{\mathcal{G}}$ do not contain blatantly inconsistent clauses.

So let $\Gamma, A(\bar{t})$ and $B(\bar{s}), \Delta$ be clauses in ϕ, with variables made disjoint. By the latter restriction, resolvable instances of these clauses in $\mathcal{G}$ can be assumed to use the same valuation; so let η be a valuation such that $[\![\bar{t}]\!]\eta = [\![\bar{s}]\!]\eta$. Then in particular $\bar{t} = \bar{s}$ is solvable in $\mathcal{G}$. Since

$\bar{t}, \bar{s} \in \mathcal{T}$, it follows by $\mathcal{T}$-equational completeness of $\mathcal{G}$ that $\bar{t}, \bar{s}$ are unifiable, hence have an mgu σ, and that σ is a most general solution of $\bar{t} = \bar{s}$ in $\mathcal{G}$. This implies that η has the form $\eta(x) = [\![\sigma(x)]\!]\eta'$ for some A-valuation η'. Thus, the resolvent $[\![\Gamma, (A \cap B)(\bar{t}), \Delta]\!]\eta$ of the two instances has the form $[\![\Gamma\sigma, (A \cap B)(\bar{t}\sigma), \Delta\sigma]\!]\eta'$, and hence is in $\phi^{\mathcal{G}}$ as required since $\Gamma\sigma, (A \cap B)(\bar{t}\sigma), \Delta\sigma$ is in ϕ by closure of ϕ under resolution. ◀

5 The AMC, Coalgebraically

To illustrate the use of one-step tableau rules, we briefly indicate how to obtain the ExpTime upper bound for the AMC by Theorem 3.3. The requisite functor G and the associated predicate liftings have been recalled in Section 3. We recall the known rule set [22, 4]:

$$(CD)\ \frac{[D_1]\, a_1, \dots, [D_\alpha]\, a_\alpha}{a_1, \dots, a_\alpha}$$

$$(C)\ \frac{[D_1]\, a_1, \dots, [D_\alpha]\, a_\alpha, \langle E\rangle\, b, \langle\Sigma\rangle\, c_1, \dots, \langle\Sigma\rangle\, c_\beta}{a_1, \dots, a_\alpha, b, c_1, \dots, c_\beta}$$

where for each j, k, $D_j \cap D_k = \emptyset$ and $D_j \subseteq E$. Soundness of these rules is straightforward (they say in particular that disjoint coalitions can combine their abilities and that coalitions inherit the abilities of subcoalitions); for illustration, we show one-step tableau completeness using set-valued resolution (Section 4), alternative to proofs in the literature [27, 8, 21].

▶ **Theorem 5.1** (One-step tableau completeness). *The rules (C), (CD) are one-step tableau complete w.r.t. AMC.*

By Theorem 3.3, this implies the known (tight) ExpTime upper bound for satisfiability checking in the AMC [21].

Proof. As indicated above, we present a proof producing infinite sets of moves in one-step games, and then discuss how finiteness of move sets is regained using the notion of $\mathcal{T}$-equationally complete (finite) model (Theorem 4.5).

Let τ be a $\mathcal{P}W$-valuation, and let $\Xi = \{[D_1]\, a_1, \dots, [D_\alpha]\, a_\alpha, \langle C_1\rangle\, c_1, \dots, \langle C_\beta\rangle\, c_\beta\}$ such that for every instance of (C) or (CD) that applies to (some subset of) Ξ, the conclusion Θ satisfies $[\![\Theta]\!]\tau \neq \emptyset$. We have to show that $[\![\Xi]\!]\tau \neq \emptyset$. To this end, we translate Ξ into a clause set ϕ in set-valued first-order logic (Section 4), generating one (singleton) clause for each modalized atom $[D_j]\, a_j$ and $\langle C_j\rangle\, c_j$ according to (1) and (2) (Section 4), with distinct Skolem constants $e^j_{D_j}$ and Skolem functions $g^j_{\overline{C_j}}$, respectively. By Theorem 4.1, it suffices to show that ϕ is consistent under set-valued resolution. We observe the following.

1. Two clauses b_j and b_k of shape (1), for $j \neq k$, resolve only if $D_j \cap D_k = \emptyset$ – otherwise, unification fails due to a clash between e^j_i and e^k_i for each agent $i \in D_j \cap D_k$.
2. Similarly, a clause b_j of shape (1) resolves with a clause d_k of shape (2) only if $D_j \cap \overline{C}_k = \emptyset$, i.e. $D_j \subseteq C_k$.
3. Similarly, two clauses d_j and d_k of shape (2), for $k \neq j$, resolve only if $\overline{C_j} \cap \overline{C_k} = \emptyset$, i.e. $C_j \cup C_k = \Sigma$
4. Crucially, two clauses d_j and d_k of shape (2) , for $k \neq j$, resolve only if at least one of C_j and C_k is Σ: Assume that $p \in \overline{C_j}$ and $q \in \overline{C_k}$. By the previous item, $p \in C_k$ and $q \in C_j$, so x_p is an argument in g^k_q and x'_q (renamed for purposes of the resolution step) is an argument in g^j_p, implying that unification of d_j and d_k fails at the occurs check (cf. p. 4). This explains why only one $\langle E\rangle$ with $E \neq N$ is needed in rule (C).

These observations imply that a resolution proof of a blatantly inconsistent (necessarily singleton) clause from ϕ will witness a rule match of either (C) or (CD) (depending on whether clauses of shape (2) are involved), and blatant inconsistency means that $[\![\Theta]\!]\tau = \emptyset$ for the corresponding rule conclusion Θ, contradicting the assumption on Ξ.

Finitely many moves. As indicated in Section 4, the model of Ξ thus produced will have infinitely many moves per agent, namely the ground terms generated by the Skolem constants and functions. We can replace these with finitely many moves where agents play *Skolem symbols* paired with *colours* – simulating the effect of the occurs check from the unification procedure – taken from a finite abelian group U (with neutral element 0 and group operation $+$) that contains distinct elements $u_1, \dots, u_\beta$ (e.g. $U = \mathbb{Z}/\beta\mathbb{Z}$). Specifically, all agents receive (for simplicity) the same moves, namely

- moves $(e^j, 0)$ for $j = 1, \dots, \alpha$, intended as witnesses for $[D_j]\, a_j$, and
- moves (g^j, u) for $j = 1, \dots, \beta$ and $u \in U$, intended as witnesses for $\langle C_j \rangle\, c_j$.

We refer to the first component of a move as its *move symbol*, and to the second as its *colour*. By $\mathsf{col}(m_C)$ we denote the sum of all colours of the moves in a joint move m_C for C.

Let $\mathcal{T}$ be the unification closure of the set of all tuples of argument terms occuring in clauses from ϕ. By the above analysis, all tuples in $\mathcal{T}$ essentially have the shape $(x_A, e_B, g_{\overline{A \cup B}}(x_A, e_B))$ where x_A are variables, e_B are Skolem constants possibly from different box modalities, and $g_{\overline{A \cup B}}$ are Skolem functions from a single diamond (as Skolem functions for different diamonds do not initially occur in the same tuple of terms and such occurrences are not introduced during unification due to the occurs check); any one of x_A, e_B, g may be absent. The (finite) model $\mathcal{G}$ is then defined over coloured moves. Skolem constants e^j are interpreted as $(e^j, 0)$, and Skolem functions g^j_i for $i \in \overline{C_j}$ are interpreted as mapping a joint move m_{C_j} of C_j to $(g^j, u_j - \mathsf{col}(m_{C_j}))$ if i is the least element of $\overline{C_j}$, and to $(g^j, 0)$ otherwise, thus ensuring that $\mathsf{col}(m_{C_j}, g^j(m_{C_j})) = u_j$. We proceed to show that $\mathcal{G}$ is $\mathcal{T}$-equationally complete, obtaining by Theorem 4.5 and consistency of ϕ under set-valued first-order resolution that ϕ is satisfiable over $\mathcal{G}$.

So let $t, u \in \mathcal{T}$ such that $t = u$ has a solution η in $\mathcal{G}$. We proceed by case distinction on the shape of $t = u$:

$(x_A, e_B) = (x'_{A'}, e'_{B'})$: In the simplest case the terms just consist of variables $(x_A,\, x'_{A'})$ and Skolem constants $(e_B,\, e'_{B'})$. Given the interpretation of the Skolem constants in $\mathcal{G}$, it is clear that e_B and e'_B must agree on $B \cap B'$ so t, u are unifiable. The solution η necessarily replaces variables in $A \cap B'$ and $A' \cap B$ with the respective interpretations of Skolem constants on the other side of the equality. Hence, the solution η factorizes through the mgu of t and u.

$(x_A, e_B, g^j_{\overline{A \cup B}}(x'_A, e'_B)) = (x_{A'}, e_{B'})$: This case is similar to the previous one, using the observation that given the interpretation of g^j in $\mathcal{G}$, the equation can only have a solution if $(\overline{A \cup B}) \cap B' = \emptyset$, i.e. $(\overline{A \cup B}) \subseteq A'$.

$(x_A, e_B, g^j_{\overline{A \cup B}}(x_A, e_B)) = (x'_{A'}, e'_{B'}, g^k_{\overline{A' \cup B'}}(x'_{A'}, e'_{B'}))$: The interpretations of the terms $g^j_{\overline{A \cup B}}(x_A, e_B)$ and $g^k_{\overline{A' \cup B'}}(x_{A'}, e'_{B'}))$ in $\mathcal{G}$ (under η) have the form (g^j, c) and (g^k, d) for some c and d, respectively. The case where $j = k$ is essentially like the previous cases. The interesting case is where $j \neq k$, in which case necessarily $\overline{A \cup B} \subseteq A'$ and $\overline{A' \cup B'} \subseteq A$; this is the case where unification of t, u fails at the occurs check as explained above. However, the construction of $\mathcal{G}$ ensures that now $t = u$ also has no solution in $\mathcal{G}$, as the respective interpretations of g^j and g^k ensure that the colour of the whole joint move is u_j on the left and u_k on the right. ◀

The proof for the AMCDES proceeds in a quite similar fashion, and will be presented in less detail.

6 AMCDES Satisfiability

We now extend this treatment to obtain ExpTime satisfiability checking for AMCDES, cast coalgebraically using the functor and predicate liftings presented in Section 3. We have one-step rules (DES_0), (DES_1), where (DES_1) is

$$(DES_1)\ \frac{[D_1, P_{G_1}]\, a_1, \dots, [D_\alpha, P_{G_\alpha}]\, a_\alpha, \langle E, Q_K\rangle\, b, \langle C_1, r_{H_1}\rangle\, c_1, \dots, \langle C_\beta, r_{H_\beta}\rangle\, c_\beta}{(a_j)_{j\in I_q}, b, (c_j)_{j\in J_q} \mid \cdots \text{ for } q \in Q_K}$$

(i.e. the rule has one conclusion for each q) where $Ag(Q_K) = K$; the r_{H_j} are (non-disjunctive) explicit joint moves for coalitions H_j; $I_q \subseteq \{1, \dots, \alpha\}$, $J_q \subseteq \{1, \dots, \beta\}$ for each $q \in Q_K$; and the following side conditions hold, with $L := \bigcup_{j=1}^{\alpha} G_j \cup \bigcup_{j=1}^{\beta} H_j$:

1. For each j, k, $D_j \cap D_k = \emptyset$.
2. For each j, $C_j \cup H_j = \Sigma$.
3. $\bigcup_{j=1}^{\alpha} D_j \cap L = \emptyset$.
4. $\bigcup_{j=1}^{\alpha} D_j \subseteq E$.
5. $E \cup K \supseteq L$.
6. $r_{H_j} =_\sqcap q$ for all $q \in Q_K$, $j \in J_q$.
7. There is a joint explicit move l for $E \cap L$ such that $r_{H_j} =_\sqcap l$ for each $q \in Q_K$, $j \in J_q$, and moreover for each $j \in I_q$ there exists $p \in P_{G_j}$ such that $p =_\sqcap q$ and $p =_\sqcap l$.

Rule (DES_0) is a variant of (DES_1) obtained by instantiating to $\langle E, Q_K\rangle\, b = \langle \Sigma, \{()\}\rangle\, \top$, $I_{()} = \{1, \dots, \alpha\}$, and $J_{()} = \{1, \dots, \beta\}$, and then omitting the (valid) literal $\langle \Sigma, \{()\}\rangle\, \top$ from the rule premiss; side conditions 4.–6. then become trivial and can be omitted.

Rule (DES_1) extends the rules for the basic AMC as recalled in Section 5. The new features are intuitively understood as follows. Imagine that $D_1, \dots, D_n$ play moves witnessing their ability to (conditionally) enforce $a_1, \dots, a_n$. According to $\langle E, Q_K\rangle$, K can then play some move $q \in Q_K$ additionally ensuring b; the q-th conclusion of (DES_1) captures the constraints on the next state reached in this situation. These additionally depend on the moves chosen by the remaining agents (those in $E \setminus \bigcup D_i$): If the arising joint move restricts to one of the moves in P_{G_j}, then D_j successfully enforces a_j, and if it restricts to r_{H_j}, then the next state must satisfy c_j (note that since $C_j \cup H_j = \Sigma$, $\langle C_i, r_{H_k}\rangle\, c_j$ says that c_j is enforced as soon as H_j play r_{H_j}). The index sets I_q and J_q indicate for which j this applies, and side conditions 6 and 7 ensure that a corresponding joint move actually exists. For definiteness, we note

▶ **Lemma 6.1** (One-step soundness). *The rules (DES_0), (DES_1) are one-step tableau sound w.r.t. AMCDES.*

Proof. By the above, it suffices to show soundness of (DES_1), formalizing the above intuitive explanation. Write ϕ for the premiss of the rule, and ψ_q for the conclusion associated to $q \in Q_K$. Let τ be a $\mathcal{P}W$-valuation such that $[\![\phi]\!]\tau \neq \emptyset$, and fix $\mathcal{G} = ((k_j), f, \iota) \in [\![\phi]\!]\tau$; we have to show that $[\![\psi_q]\!]\tau \neq \emptyset$ for some $q \in Q_K$. We refer to side conditions by their numbers:

- For each $j \in \{1, \dots, \alpha\}$, we have a joint move e_j for D_j witnessing $[D_j, P_{G_j}]\, a_j$. By 1., the e_j can be combined into a joint move e for $\bigcup_{j=1}^{\alpha} D_j$.

- By 3., e can be combined with (the interpretation of) the explicit move l postulated in 7. into a move x_0 for $(E \cap L) \cup \bigcup_{j=1}^{\alpha} D_j \subseteq E$, where the inclusion is by 4. Extend x_0 arbitrarily to a move x for the whole coalition E.
- Since $\mathcal{G} \in [\![\langle E, Q_K \rangle\, b]\!]$ and $Ag(x) = E$, there is some $q \in Q_K$ and a joint move m_q for Σ such that $x, q \sqsubseteq m_q$ and $f(m_q) \in \tau(b)$.
- To obtain that $f(m_q) \in [\![\psi_q]\!]\tau$ for this q, it remains to show that $f(m_q)$ satisfies the remaining literals a_j, c_j of ψ_q:
  - For $j \in I_q$, we have $e_j \sqsubseteq m_q$ and, by 5. and 7., $\iota[p] \sqsubseteq m_q$ for some $p \in P_{G_j}$, so that $\mathcal{G} \in [\![[D_j, P_{G_j}]\, a_j]\!]\tau$ implies $f(m_q) \in \tau(a_j)$.
  - For $j \in J_q$, we have $\iota[r_{H_j}] \sqsubseteq m_q$ by 5., 6., and 7. Since $C_j \cup H_j = \Sigma$, we thus have that $\mathcal{G} \in [\![\langle C_k, r_{H_k} \rangle\, c_k]\!]\tau$ implies $f(m_j) \in \tau(c_k)$. ◀

It remains to prove ompleteness:

▶ **Lemma 6.2** (One-step tableau completeness). *The rules (DES_0), (DES_1) are one-step tableau complete w.r.t. AMCDES.*

Proof. Let τ be a $\mathcal{PW}$-valuation, and let $\Xi = \{[D_1, P_{G_1}]\, a_1, \dots, [D_\alpha, P_{G_\alpha}]\, a_\alpha, \langle C_1, R_{H_1} \rangle\, c_1, \dots, \langle C_\beta, R_{H_\beta} \rangle\, c_\beta\}$ such that every instance of (DES_0) or (DES_1) whose premise is contained in Ξ has a conclusion that is non-empty under τ. We have to show that $[\![\Xi]\!]\tau \neq \emptyset$. We translate Ξ into a clause set ϕ in set-valued first-order logic by including for each $[D_j, P_{G_j}]\, a_j$ and each $p \in P_{G_j}$ a singleton clause

$$\{\tau(a_j)(e^j_{D_j}, x_{\overline{D_j \cup G_j}}, p)\}, \tag{3}$$

(so $e^j_{D_j}$ witnesses $[D_j, P_{G_j}]\, a_j$), and for each $\langle C_j, R_{H_j} \rangle\, c_j$ a clause

$$\{\tau(c_j)(x_{C_j}, g^j_{\overline{C_j \cup H_j}}(x_{C_j}), r) \mid r \in R_{H_j}\} \tag{4}$$

(so the $g^j_{\overline{C_j \cup H_j}}$ are Skolem functions witnessing $\langle C_j, R_{H_j} \rangle\, c_j$). We now proceed as in the proof of Theorem 5.1: We first show that ϕ is consistent under set-valued resolution, obtaining by Theorem 4.1 that ϕ is satisfiable in a model that may have infinitely many moves, and then present a finite $\mathcal{T}$-equationally complete model for the unification closure $\mathcal{T}$ of the involved terms. Write b^p_j for clauses of type (3) for given $j = 1, \dots, \alpha$ and $p \in G_j$, and d_j for the j-th clause of type (4).

Unlike in the proof of Theorem 5.1, we thus may have non-singleton clauses, of shape (4). However, we shall see that these non-singleton clauses do not resolve among each other. We note the following observations.

1. b^p_j and b^q_j, for $p \neq q$, do not resolve (and resolving b^p_j with itself is pointless).
2. b^p_j and b^q_k, for $k \neq j$, resolve only if $D_j \cap D_k = D_j \cap G_k = D_k \cap G_j = \emptyset$, and moreover $p =_\sqcap q$.
3. b^p_j and d_k resolve, at the d_k-literal for $r \in R_{H_k}$, only if $D_j \subseteq C_k$, and hence in particular also $D_j \cap H_k = \emptyset$, $D_j \cup G_j \subseteq C_k \cup H_k$ (equivalently $\overline{C_k \cup H_k} \subseteq \overline{D_j \cup G_j}$), and $r =_\sqcap p$.
4. d_j and d_k, for $k \neq j$, resolve, at the d_j-literal for $r \in H_j$ and the d_k-literal for $r' \in H_k$, only if $C_j \cup C_k \cup H_k = \Sigma$ (equivalently $\overline{C_k \cup H_k} \subseteq C_j$), $C_k \cup C_j \cup H_j = \Sigma$, and $r =_\sqcap r'$.
5. Like in the proof of Theorem 5.1, it follows that d_j and d_k resolve only if at least one of $\langle C_j, R_{H_j} \rangle$ and $\langle C_k, R_{H_k} \rangle$ is a grand coalition modality (since otherwise unification fails at the occurs check), in which case the corresponding clause is a singleton.
6. Clauses obtained from clauses of shape (4) by resolving with singleton clauses retain essentially shape (4), only with some of the variables x_i replaced with constants. Resolution of such clauses is thus subject to the same restrictions; in particular, non-singleton clause of this kind they will not resolve among each other.

Thus, a proof of a blatantly inconsistent clause from ϕ by set-valued resolution will involve either zero or one clauses d_j where $C_j \cup H_j \neq \Sigma$. We will refer to resolution proofs of the first kind as type-0 and to proofs of the second kind as type-1.

Type-0 proofs. We show that in this case, the impossibility of deriving a blatantly inconsistent clause is obtained via rule (DES_0). To apply (DES_0) to the set of modal atoms involved in the proof, we need to show the side conditions of the rule (1.–3. and 7). Indeed, condition 2. holds by the definition of type-0 proofs. As no disjunctive diamond is involved in a type-0 proof, all involved clauses are singletons. Hence, 1., 3., and 7. directly follow from the observations above. The type-0 proof at hand thus induces a match of rule (DES_0) to a subset of Ξ; the conclusion of this rule match having non-empty extension under τ means precisely that the resolution proof does not produce a blatantly inconsistent clause.

Type-1 proofs. Those consist in successively resolving all literals of a single clause of the form d_{j_0} where $C_{j_0} \cup H_{j_0} \neq \Sigma$ with suitable singleton clauses, of the form either b_j^p or d_k where $C_k \cup H_k = \Sigma$. We will refer to these resolution steps as "resolving into d_{j_0}", although of course d_{j_0} will have been modified by previous resolution steps as described above. To match the notation of rule (DES_1), we rename $\langle C_{j_0}, R_{H_{j_0}} \rangle$ into $\langle E, Q_K \rangle\, b$ (so that all the $\langle C_j, R_{H_j} \rangle\, c_j$ that remain have $C_j \cup H_j = \Sigma$ and hence $|R_{H_j}| = 1$). The literals in d_{j_0} are then indexed over $q \in Q_K$. Let I_q be the set of all j such that for some $p \in P_j$, b_j^p is resolved into d_{j_0} at the literal for q, and put $G = \bigcup_{q \in Q_K, j \in I_q} G_j$; similarly, let J_q be the set of all j such that d_j (a singleton clause) is resolved into d_{j_0} at the literal for q, and put $H = \bigcup_{q \in Q_K, j \in J_q} H_j$. Notice that two clauses resolve only if whenever they both assign a constant (either a Skolem constant or an explicit move) to a certain agent, then the constant is the same in both clauses; this implies condition 7. Conditions 1. and 3. are established as in the type-0 case, condition 2. is ensured by the above renaming, and the remaining conditions follow directly from the above observations. The type-1 proof at hand thus induces a match of rule (DES_1) to a subset of Ξ; a conclusion of this rule match having non-empty extension under τ means precisely that the resolution proof does not produce a blatantly inconsistent clause.

Finitely many moves. As indicated above, we obtain a model with finitely many moves by constructing a finite $\mathcal{T}$-equationally complete model $\mathcal{G}$, where $\mathcal{T}$ is the unification closure of the tuples of terms occurring in ϕ. This construction is essentially the same as for the AMC, up to the presence of additional constant symbols, viz. the explicit strategies occurring in ϕ. These constants can be treated exactly like the Skolem constants already present in the proof of Theorem 5.1. The full proof is available in Appendix B. ◀

Since the rules (DES_1), (DES_0) are algorithmically sufficiently harmless, our main result follows from Lemmas 6.1 and 6.2 by Theorem 3.3:

▶ **Theorem 6.3.** *Satisfiability checking for the AMCDES is* ExpTime*-complete.*

7 Conclusions

We have introduced the alternating-time μ-calculus with disjunctive explicit strategies (AMCDES), which extends ATL with explicit strategies (ATLES) [29] with fixpoint operators and disjunction over explicit strategies of opposing agents in non-grand modalities. We have employed methods from coalgebraic logic to show that model checking with fixed interpretation of explicit strategies is in QP as well as in NP ∩ coNP, and in NP with open interpretation of strategies, and moreover that satisfiability checking is in ExpTime.

The coalgebraic treatment in fact implies a whole range of additional results, e.g. reasoning in the next-step fragment of the logic extended with nominals (ExpTime with global axioms, and PSpace without) [24, 17, 9]; cut-free sequent systems for the next-step fragment [19]; and completeness of a Kozen-Park axiomatization for flat (i.e. single-variable) fragments of the AMCDES, e.g. ATL with disjunctive explicit strategies [25]. A special case of the latter result is completeness of ATLES as proved already in Walther et al. [29].

In ongoing work we are extending our axiomatization and complexity results to allow strategy disjunction also in grand coalition modalities. A natural but more challenging further extension would be to add negative strategies prohibiting moves for some agents as suggested by Herzig et al. [12].

References

1 Rajeev Alur, Thomas Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49:672–713, 2002. doi:10.1145/585265.585270.

2 Cristian Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Theory of Computing, STOC 2017*, pages 252–263. ACM, 2017.

3 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010. doi:10.1016/j.ic.2009.07.004.

4 Corina Cîrstea, Clemens Kupke, and Dirk Pattinson. EXPTIME tableaux for the coalgebraic μ-calculus. *Log. Methods Comput. Sci.*, 7(3), 2011. doi:10.2168/LMCS-7(3:3)2011.

5 Corina Cîrstea, Alexander Kurz, Dirk Pattinson, Lutz Schröder, and Yde Venema. Modal logics are coalgebraic. *Comput. J.*, 54(1):31–41, 2011. doi:10.1093/comjnl/bxp004.

6 Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 2nd edition, 1996. doi:10.1007/978-1-4612-2360-3.

7 Valentin Goranko and Fengkui Ju. Towards a logic for conditional local strategic reasoning. In Patrick Blackburn, Emiliano Lorini, and Meiyun Guo, editors, *Logic, Rationality, and Interaction, LORI 2019*, volume 11813 of *LNCS*, pages 112–125. Springer, October 2019. doi:10.1007/978-3-662-60292-8_9.

8 Valentin Goranko and Govert van Drimmelen. Complete axiomatization and decidability of alternating-time temporal logic. *Theor. Comput. Sci.*, 353(1-3):93—117, March 2006. doi:10.1016/j.tcs.2005.07.043.

9 Rajeev Goré, Clemens Kupke, Dirk Pattinson, and Lutz Schröder. Global caching for coalgebraic description logics. In *Automated Reasoning, IJCAR 2010*, volume 6173 of *LNCS*, pages 46–60. Springer, 2010. doi:10.1007/978-3-642-14203-1.

10 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002. doi:10.1007/3-540-36387-4.

11 Daniel Hausmann and Lutz Schröder. Game-based local model checking for the coalgebraic mu-calculus. In *Concurrency Theory, CONCUR 2019*, volume 140 of *LIPIcs*, pages 35:1–35:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, August 2019. doi:10.4230/LIPIcs.CONCUR.2019.35.

12 Andreas Herzig, Emiliano Lorini, and Dirk Walther. Reasoning about actions meets strategic logics. In *Logic, Rationality, and Interaction, LORI 2013*, volume 8196 of *LNCS*, pages 162–175. Springer, 2013. doi:10.1007/978-3-642-40948-6_13.

13 Marc-Uwe Kling. *Das Känguru-Manifest*. Ullstein, Berlin, 2011.

14 Marc-Uwe Kling. *The Kangaroo Chronicles*. Voland & Quist, 2016. Translated by Sarah Cossaboon and Paul-Henri Campbell.

15 Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Vardi. Reasoning about strategies: On the model-checking problem. *ACM Trans. Comput. Log.*, 15(4):34:1–34:47, 2014. doi:10.1145/2631917.

16 Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Vardi. Reasoning about strategies: on the satisfiability problem. *LMCS*, 13(1), 2017. doi:10.23638/LMCS-13(1:9)2017.

17 Robert Myers, Dirk Pattinson, and Lutz Schröder. Coalgebraic hybrid logic. In *Foundations of Software Science and Computational Structures, FOSSACS 2009*, volume 5504 of *LNCS*, pages 137–151. Springer, 2009. doi:10.1007/978-3-642-00596-1.

18 Rocco De Nicola and Frits W. Vaandrager. Action versus state based logics for transition systems. In Irène Guessarian, editor, *Semantics of Systems of Concurrent Processes, LITP Spring School on Theoretical Computer Science 1990*, volume 469 of *LNCS*, pages 407–419. Springer, April 1990. doi:10.1007/3-540-53479-2_17.

19 Dirk Pattinson and Lutz Schröder. Cut elimination in coalgebraic logics. *Inf. Comput.*, 208(12):1447–1468, 2010. doi:10.1016/j.ic.2009.11.008.

20 Marc Pauly. A modal logic for coalitional power in games. *J. Log. Comput.*, 12(1):149–166, 2002. doi:10.1093/logcom/12.1.149.

21 Sven Schewe. *Synthesis of Distributed Systems*. PhD thesis, Universität des Saarlandes, 2008.

22 Lutz Schröder and Dirk Pattinson. PSPACE bounds for rank-1 modal logics. *ACM Trans. Comput. Log.*, 10(2):13:1–13:33, 2009. doi:10.1145/1462179.1462185.

23 Lutz Schröder and Dirk Pattinson. Modular algorithms for heterogeneous modal logics via multi-sorted coalgebra. *Math. Struct. Comput. Sci.*, 21(2):235–266, 2011. doi:10.1017/S0960129510000563.

24 Lutz Schröder, Dirk Pattinson, and Clemens Kupke. Nominals for everyone. In *International Joint Conference on Artificial Intelligence, IJCAI 2009*, pages 917–922, July 2009. URL: http://ijcai.org/proceedings/2009.

25 Lutz Schröder and Yde Venema. Completeness of flat coalgebraic fixpoint logics. *ACM Trans. Comput. Log.*, 19(1):4:1–4:34, 2018. doi:10.1145/3157055.

26 Wiebe van der Hoek, Wojciech Jamroga, and Michael Wooldridge. A logic for strategic reasoning. In *Autonomous Agents and Multiagent Systems, AAMAS 2005*, pages 157–164. ACM, 2005. doi:10.1145/1082473.1082497.

27 Govert van Drimmelen. Satisfiability in alternating-time temporal logic. In *Logic in Computer Science, LICS 2003*, pages 208–217. IEEE Comp. Soc., June 2003. doi:10.1109/LICS.2003.1210060.

28 Dirk Walther. *Strategic Logics: Complexity, Completeness and Expressivity*. PhD thesis, University of Liverpool, 2007.

29 Dirk Walther, Wiebe van der Hoek, and Michael Wooldridge. Alternating-time temporal logic with explicit strategies. In *Theoretical Aspects of Rationality and Knowledge, TARK 2007*, pages 269–278. ACM Press, 2007. doi:10.1145/1324249.1324285.

30 Thomas Ågotnes. Action and knowledge in alternating-time temporal logic. *Synthese*, 149(2):375–407, March 2006. doi:10.1007/s11229-005-3875-8.

A Appendix: AMCDES Model Checking Details

Summary of Results on Coalgebraic Model Checking

Given a functor F, we assume a representation of the elements of FX, for finite X, as strings over some alphabet. Specifically, we represent elements of $((k_j)_{j\in\Sigma}, f) \in \mathsf{G}X$ as tabulations of f.

Model checking results [11] for the full coalgebraic μ-calculus require only very simple properties of the predicate liftings:

▶ **Definition A.1.** The *one-step satisfaction problem* is to determine, given a finite set X, $Y \subseteq X$, $\heartsuit \in \Lambda$, and $t \in FX$, whether $t \in [\![\heartsuit]\!]_X(Y)$.

▶ **Theorem A.2** (Model checking via one-step satisfaction [11, Theorem 11])**.** *If the one-step satisfaction problem is in P, then the model checking problem for the coalgebraic μ-calculus over this logic is in NP $\cap$ coNP.*

The proof of this upper bound is via parity games, specifically by noting that Cîrstea et al.'s *evaluation games* [4] are exponentially large but have only polynomially many Eloise-nodes, so that winning strategies for Eloise can be guessed and verified in (nondeterministic) polynomial time.

On the other hand, to obtain a model checking algorithm in QP (deterministic quasipolynomial time $2^{\mathcal{O}((\log n)^k)}$ for some k; a complexity class not currently known to be comparable with NP) we need to show that we can design suitable one-step satisfaction arenas for use in model checking games (we use standard terminology for games, e.g. [10]):

▶ **Definition A.3.** A *one-step satisfaction arena* A for a set X, a modality $\heartsuit \in \Lambda$, and $t \in FX$ is an acyclic arena for games with two players Eloise and Abelard (recall that an arena is like a game in that it specifies nodes, each assigned to one of the players, and allowed moves between nodes but does not include a winning condition; acyclicity refers to the move relation), with a single initial node, with X as the set of terminal nodes, and with additional inner nodes. A *one-step game* on A additionally specifies a winning condition in the shape of a subset Y of the terminal nodes; then, Eloise wins plays that either get stuck at an inner Abelard node without successors or terminate in a node in Y. We say that A is *sound and complete* if for every $Y \subseteq X$, Eloise wins (the initial node of) the one-step game on A with winning condition Y iff $t \in [\![\heartsuit]\!]_X(Y)$.

▶ **Theorem A.4** (Model checking via one-step games [11, Corollary 18])**.** *If for every set X, $\heartsuit \in \Lambda$, and $t \in FX$, there is a sound and complete one-step satisfaction arena with polynomially many inner nodes in the representation size of $\heartsuit$ and t, then the model checking problem for the μ-calculus over this logic is in QP.*

The model checking procedure underlying this theorem is to construct a polynomial-size model checking parity game using one-step games as building blocks; by well-known recent advances in parity game solving [2], these games can be solved in quasipolynomial time.

Proof of Theorem 2.9

Proof. The one-step satisfaction problem for the AMCDES is to check whether $((k_j), f, \iota) \in [\![[C, O]]\!]_X(Y)$ can be decided in P for given C, O, $Y \subseteq X$, and a one-step game with explicit strategies $((k_j), f, \iota) \in \mathsf{G}_{\mathsf{ES}}X$. This can be done by iterating over joint moves of C in an outer loop and over joint moves of $\overline{C}$ in an inner loop. Since f needs to tabulate the outcomes of all joint moves of Σ, both loops have at most linearly many (in the size of f) iterations per invocation, making for a quadratic overall number of iterations of the inner loop, and hence polynomial run time.

Algorithm 1 One-step Satisfaction Algorithm.

```
for $m_C \leftarrow [k_C]$ do
    $x := \top$
    for $o \leftarrow O, m_{\bar{C}} \leftarrow [k_{\Sigma\setminus C\setminus Ag(O)}]$ do
        if $f(m_c, m_{\bar{C}}, \iota[o]) \notin Y$ then $x := \bot$
    if $x$ then return $\top$
return $\bot$
```

By Theorem A.2, we thus obtain the NP $\cap$ coNP bound for the fixed case. The NP bound for the open case follows by guessing history-free strategies.

For the QP bound, we use Theorem A.4 and adapt the one-step satisfaction arenas for the AMC [11, Example 15.5] to obtain small one-step satisfaction arenas for the AMCDES: The one-step satisfaction arena $A_{[C,O],w} = (V_{[C,O],w}, E_{[C,O],w})$ for X, $[C,O]$, and a one-step game $((k_j), f, \iota) \in \mathsf{G_{ES}}X$ for disjoint $C, D \subseteq \Sigma$, $O \subseteq \prod_{a\in D} M_a$ is constructed as follows. The node set $V_{[C,O],w}$ consists of an initial node $([C,O],w)$ belonging to Eloise, and additionally a set of inner nodes $I_{[C,O],w} := [k_C]$ belonging to Abelard i.e. one node for each joint move of C. The set $E_{[C,O],w}(x)$ of moves available at a node x is

$$E_{[C,O],w}(x) = \begin{cases} I_{[C,O],w} \text{ if } x = ([C,O],w) \\ \{f(x, m_{\bar{C}}, o) \mid m_{\bar{C}} \in [k_{\overline{C\cup D}}], o \in \iota[O]\} \end{cases}$$

It is easy to see that the size of the arena is thus linear in the tabulation size of f. The soundness and completeness of the resulting one-step satisfaction game stems from the fact that the moves of Eloise and Abelard essentially construct the witnessing moves from the original game. ◀

B Appendix: Omitted Proofs and Further Details

Proof of Theorem 4.1

Soundness. It suffices to show that the rule (SR) is sound. Let $\Gamma, A(\bar{t})$ and $B(\bar{u}), \Delta$ be two clauses such that $\bar{t}$ and $\bar{u}$ are unifiable, and let $\sigma = mgu(\bar{t}, \bar{u})$. Let $\mathcal{G} = ((S_j)_{j\in N}, f, W, [\![-]\!])$ be an outcome model satisfying both $\Gamma, A(\bar{t})$ and $B(\bar{u}), \Delta$. Let η be a valuation such that $\mathcal{G}, \eta \not\models \Gamma\sigma, \Delta\sigma$; we have to show $\mathcal{G}, \eta \models (A \cap B)(\bar{t}\sigma)$. By the evident substitution lemma, $\mathcal{G}, \eta_\sigma \not\models \Gamma, \Delta$ where $\eta_\sigma(x) = [\![\sigma(x)]\!]\eta$ for all x; hence necessarily $\mathcal{G}, \eta_\sigma \models A(\bar{t})$ and $\mathcal{G}, \eta_\sigma \models B(\bar{u})$. Again by the substitution lemma, $\mathcal{G}, \eta \models A(\bar{t}\sigma)$ and $\mathcal{G}, \eta \models B(\bar{u}\sigma)$. Since $\bar{t}\sigma = \bar{u}\sigma$, our goal $\mathcal{G}, \eta \models (A \cap B)(\bar{t}\sigma)$ follows by the semantics of literals.

Completeness. The completeness proof for the propositional variant proceeds via *maximally consistent clause sets*, defined in the expected way. By Zorn's lemma, we have

▶ **Lemma B.1** (Lindenbaum lemma for set-valued propositional resolution). *Every consistent clause set in set-valued propositional logic is contained in a maximally consistent set.*

Moreover, we have the following set of Hintikka properties:

▶ **Lemma B.2** (Hintikka lemma for set-valued propositional resolution). *Let ϕ be a maximally consistent clause set in set-valued propositional logic. Then*

1. *A clause Γ, Δ is in ϕ iff $\Gamma \in \phi$ or $\Delta \in \phi$.*
2. *A clause $\Gamma, (A \cup B)(y)$ is in ϕ iff one of $\Gamma, A(y)$ and $\Gamma, B(y)$ is in ϕ.*
3. *For every $y \in Y$, $W(y) \in \phi$.*

Proof. *1, "if":* Assume w.l.o.g. that $\Gamma \in \phi$. By maximality, it suffices to show that $\phi \cup \{\Gamma, \Delta\}$ remains consistent. So assume that a blatantly inconsistent clause can be derived from $\phi \cup \{\Gamma, \Delta\}$. Then by removing literals from the clauses in this derivation, we obtain a derivation of a blatantly inconsistent clause from $\phi \cup \{\Gamma\}$, contradiction.

1, "only if": By maximality, it suffices to show that one of $\phi \cup \{\Gamma\}$ and $\phi \cup \{\Delta\}$ is consistent. Assume the contrary. Then one can derive a blatantly inconsistent clause Γ' from $\phi \cup \{\Gamma\}$. Adding Δ to all clauses in the derivation (that is, to the original Γ and then to all clauses newly produced by the resolution rule), we obtain a derivation of Γ', Δ from $\phi \cup \{\Gamma, \Delta\}$. Similarly, we have a derivation of a blatantly inconsistent clause Δ' from $\phi \cup \{\Delta\}$, from which we obtain a derivation of Γ', Δ' from $\phi \cup \{\Gamma', \Delta\}$. Chaining the two derivations, we obtain a derivation of the blatantly inconsistent clause Γ', Δ' from $\phi \cup \{\Gamma, \Delta\}$, contradiction.

2, "if": Assume w.l.o.g. that $\Gamma, A(y)$ is in ϕ. By maximality, it suffices to show that $\phi \cup \{\Gamma, (A \cup B)(y)\}$ is consistent. Assume the contrary, i.e. we can derive a blatantly inconsistent clause from $\Gamma, (A \cup B)(y)$. Tracing $(A \cup B)(y)$ through the derivation in the obvious sense (with $A \cup B$ possibly transformed into strictly smaller subsets by the resolution rule) and intersecting with A at each occurrence, we obtain a derivation of a blatantly inconsistent clause from $\phi \cup \{\Gamma, A(y)\} = \phi$, contradiction.

2, "only if": By contraposition, again using maximality: assume that both $\phi \cup \{\Gamma, A(y)\}$ and $\phi \cup \{\Gamma, B(y)\}$ are inconsistent; we have to show that $\phi \cup \{\Gamma, (A \cup B)(y)\}$ is inconsistent. By assumption, we can derive from $\phi \cup \{\Gamma, A(y)\}$ a blatantly inconsistent clause, necessarily of the form $\Gamma', \emptyset(y)$ (since no $y \in Y$ can be made to disappear by the resolution rule). Tracing $A(y)$ through the derivation and taking unions with B at each occurrence, we obtain a derivation of $\Gamma', B(y)$ from $\phi \cup \{\Gamma, (A \cup B)(y)\}$. Similarly, we can derive a blatantly inconsistent clause from $\phi \cup \{\Gamma, B(y)\}$. Replacing literals $C(z)$ with $\emptyset(z)$ and adding new literals of the form $\emptyset(z)$, we obtain a derivation of a blatantly inconsistent clause Θ from $\phi \cup \{\Gamma', B(y)\}$. Chaining derivations, we obtain a derivation of Θ from $\phi \cup \{(A \cup B)(y)\}$, showing the required inconsistency.

3: Clear. ◀

Now fix a maximally consistent clause set ϕ, and assume that W is finite; we construct a model, i.e. a function $f_\phi : Y \to W$, from ϕ as follows. For $y \in Y$, we have $W(y) \in \phi$ by the Hintikka lemma, and then, again by the Hintikka lemma and by finiteness of W, $\{w_y\}(y) \in \phi$ for some $w_y \in W$, which by consistency of ϕ is moreover unique; we put $f_\phi(y) = w_y$.

▶ **Lemma B.3** (Truth lemma for set-valued propositional resolution). *Given a maximally consistent clause set ϕ in set-valued propositional logic over a finite set W, the function f_ϕ constructed above satisfies ϕ.*

Proof. Induction over the size of clauses Γ, measured as the sum of the cardinalities of the subsets of W occurring in Γ. The inductive step makes a case distinction over whether there is more than one or exactly one literal in Γ (the case of zero literals does not occur, as a clause without literals is blatantly inconsistent), and then proceeds according to the relevant clause of the Hintikka lemma. We are left with the induction base, where Γ has the form $\{w\}(y)$; in this case, the claim holds by construction of f_ϕ. ◀

In combination with Lemma B.1, this proves completeness of the propositional variant. Completeness for the first-order variant is then shown via a form of Herbrand theory. We build a *Herbrand universe* where the moves of each agent i are ground terms of sort i. We denote these sets of moves by S_i. A *ground substitution* replaces variables by ground terms, respecting sorts. *Ground instances* of literals $A(\bar{t})$, clauses, and clause sets are obtained by applying a ground substitution.

Now let ϕ be a clause set in set-valued first-order logic that is closed under set-valued first-order resolution and not blatantly inconsistent; it suffices to show that such ϕ are satisfiable. We denote by $I(\phi)$ the set of ground instances of clauses in ϕ. To show that ϕ is satisfiable over the Herbrand universe, it suffices to establish that $I(\phi)$ is satisfiable. Clearly, $I(\phi)$ is not blatantly inconsistent. We show that it is moreover closed under set-valued propositional resolution (implying that $I(\phi)$ is satisfiable, and hence that ϕ is satisfiable). A pair of resolvable clauses in $I(\phi)$ has the form $\Gamma\theta, A(\bar{t}\theta)$ and $B(\bar{u}\theta), \Delta\theta$ where $\Gamma, A(\bar{t})$ and $B(\bar{u}), \Delta$ are in ϕ, w.l.o.g. with disjoint sets of variables, and θ is a ground substitution such that $\bar{t}\theta = \bar{u}\theta$. In particular, $\bar{t}$ and $\bar{u}$ are unifiable, and thus have a most general unifier σ; by definition of the latter, there exists θ' such that $\theta = \sigma\theta'$. Since ϕ is closed under resolution, it follows that the resolvent $\Gamma\sigma, (A \cap B)(\bar{t}\sigma), \Delta\sigma$ is in ϕ. Applying the ground substitution θ' to this clause, we obtain that the propositional resolvent $\Gamma\theta, (A \cap B)(\bar{t}\theta), \Delta\theta$ is in $I(\phi)$, as required. ◀

Remarks on One-step Tableau Completeness for the AMC (Theorem 5.1)

In the proof of Theorem 5.1, one could equally well have used previous one-step model constructions implicit in van Drimmelen, Goranko, and Schewe [27, 8, 21]; we provide our construction for illustration, in preparation for the treatment of disjunctive explicit strategies, to which, as far as we can see, the previous constructions do not adapt (they do extend to explicit strategies without strategy disjunction). We note that the model construction becomes much simpler if one excludes the grand coalition (as, effectively, in ATLES): In the rule (C), the literals $\langle\Sigma\rangle\, c_j$ disappear; in the proof of one-step tableau completeness of the arising rule, one can just use a single move $\bot$ as witness for all $\langle C_j\rangle\, c_j$ in Ξ (in the notation of the original proof of Theorem 5.1), using non-determinism to ensure satisfaction of the $\langle C_j\rangle\, c_j$. In detail, this is seen as follows.

As indicated above, in the absence of grand coalition modalities, rule (C) specializes to

$$(C^-)\ \frac{[D_1]\, a_1, \dots, [D_\alpha]\, a_\alpha, \langle E\rangle\, b}{a_1, \dots, a_\alpha, b}$$

with the same side conditions as (C). The shorter proof of one-step tableau completeness then runs as follows. Let τ be a $\mathcal{P}W$-valuation, and let $\Xi = \{[D_1]\, a_1, \dots, [D_\alpha]\, a_\alpha, \langle C_1\rangle\, c_1, \dots, \langle C_\beta\rangle\, c_\beta\}$ (where $D_j \neq \Sigma$, $C_j \neq \Sigma$ for all j) be such that every rule match of (C^-) to Ξ has non-empty conclusion under τ. We have to construct an element of $[\![\Xi]\!]\tau$. Give every agent moves e_j for $j = 1, \dots, n$ intended as witnesses for $[D_j]\, a_j$, and a single refusal move $\bot$; write (slightly abusively) e_{D_j} for the joint move of D_j that is e_j in all components. Define a non-deterministic outcome function f by $f(m_\Sigma) = \bigcap_{e_{D_j} \sqsubseteq m_\Sigma} \tau(a_j)$, noting that this set is non-empty thanks to rule (CD) since for $j \neq k$, having both $e_{D_j} \sqsubseteq m_\Sigma$ and $e_{D_k} \sqsubseteq m_\Sigma$ implies $D_j \cap D_k = \emptyset$. Then f clearly satisfies $[D_j]\, a_j$ under τ. To see that f also satisfies $\langle C_j\rangle\, c_j$, let m_{C_j} be a joint move of C_j. Let m_Σ be the joint move of Σ extending m_{C_j} by letting all other agents pick $\bot$. We have to show that $f(m_\Sigma) \cap \tau(c_j) \neq \emptyset$. But this is immediate by rule (C^-), since $e_{D_k} \sqsubseteq m_\Sigma$ implies $D_k \subseteq C_j$.

We note further that excluding grand coalition modalities is equivalent to making the outcome function non-deterministic: It is clear that excluding grand coalition modalities is equivalent to always taking the set of agents to consist of the agents Σ_ϕ mentioned in the target formula ϕ and one extra agent $*$ (convert models with larger set C of additional agents into one with only $*$ by taking the previous joint moves of C to be the moves of $*$). Then, note that ϕ is satisfiable in a CGS with set $\Sigma_\phi \cup \{*\}$ of agents iff ϕ is satisfiable in a *non-deterministic CGS* with set $\Sigma = \Sigma_\phi$ of agents, where a non-deterministic CGS is defined like a CGS except that the outcome function f_q at a state q returns a non-empty set of possible post-states rather than just a single post-state. Over such a non-deterministic CGS, a formula $[C]\,\psi$ is satisfied at a state q if C has a joint move m_C such that for all joint moves $m_{\overline{C}}$ of $\overline{C}$, *all* possible post-states of q under the induced joint move of Σ satisfy ψ. A non-deterministic CGS with set Σ of agents is converted into a CGS with set $\Sigma \cup \{*\}$ of agents by giving $*$ all states as moves, allowing $*$ to pick one of the possible post-states determined by the other agents (with some possible post-state chosen arbitrarily if $*$ plays a state that is not a possible post-state). Conversely, a CGS S with set $\Sigma \cup \{*\}$ of agents is converted into a non-deterministic CGS with set Σ of agents by taking the possible post-states under a joint move m_Σ of the agents in Σ to be the set of all post-states of joint moves in S extending m_Σ. Both conversions clearly preserve satisfaction of formulae ϕ mentioning only agents in Σ.

Proof of One-step Tableau Completeness for the AMCDES (Lemma 6.2) with Finite Sets of Moves

Proof. Similarly to how the finite moves were achieved in the proof of Theorem 5.1, we will colour the moves to simulate the effect of the occurs check in unification. We use the same terminology and notation for colours as in the proof of Theorem 5.1, and take the colours from the same Abelian group U. Let ϕ be the clause set constructed in the ongoing proof as shown in the main part of the paper. Now, all agents receive (for simplicity) the same moves, namely

- moves $(e^j, 0)$ for $j = 1, \dots, \alpha$, intended as witnesses for the moves of the agents in D_j in $[D_j, P_{G_j}]\, a_j$,
- moves $(p, 0)$ for $j = 1, \dots, \alpha$, $p \in P_{G_j}$ witnessing explicit moves from $[D_j, P_{G_j}]\, a_j$,
- moves $(r, 0)$ for $j = 1, \dots, \beta$, $r \in R_{H_j}$ witnessing explicit moves from $\langle C_j, R_{H_j}\rangle\, c_j$, and
- moves (g^j, u) for $j = 1, \dots, \beta$ and $u \in U$, intended as witnesses for $\langle C_j, R_{H_j}\rangle\, c_j$.

Let $\mathcal{T}$ be the unification closure of all argument terms occuring in clauses in ϕ. All tuples in $\mathcal{T}$ have the shape $(x_A, e_B, p_C, r_D, g_{\overline{A \cup B \cup C \cup D}}(x_A, e_B, p_C, r_D))$ where the x_A are variables; the e_B are Skolem constants and p_C, r_D are constants for named moves, from possibly different boxes and diamonds; and the $g_{\overline{A \cup B \cup C \cup D}}$ are Skolem functions from a single diamond, as Skolem functions from multiple diamonds do not occur together in the starting terms and such occurrences are not introduced during unification due to the occurs check.

The (finite) model $\mathcal{G}$ is then defined over coloured moves. Skolem constants e^j are interpreted as $(e^j, 0)$, explicit strategies r and p are interpreted as $(r, 0)$ and $(p, 0)$, and Skolem functions g_i^j for $i \in \overline{C_j}$ are interpreted as mapping a joint move m_{C_j} of C_j to $(g^j, u_j - \mathsf{col}(m_{C_j}))$ if i is the least element of $\overline{C_j}$, and to $(g^j, 0)$ otherwise, thus ensuring that $\mathsf{col}(m_{C_j}, g^j(m_{C_j})) = u_j$. It remains to show that $\mathcal{G}$ is $\mathcal{T}$-equationally complete, obtaining by Theorem 4.5 and consistency of ϕ under set-valued first-order resolution that ϕ is satisfiable over $\mathcal{G}$. Indeed, observing that the symbols for explicit strategies represent constants in the unification process and are translated exactly like the Skolem constants, we can treat them as part of e_B and proceed in the same way as in Theorem 5.1. ◀

On the Complexity of Horn and Krom Fragments of Second-Order Boolean Logic

Miika Hannula
Department of Mathematics and Statistics, University of Helsinki, Finland
miika.hannula@helsinki.fi

Juha Kontinen
Department of Mathematics and Statistics, University of Helsinki, Finland
juha.kontinen@helsinki.fi

Martin Lück
Institut für Theoretische Informatik, Leibniz Universität Hannover, Germany
lueck@thi.uni-hannover.de

Jonni Virtema
Faculty of Humanities and Human Sciences, Hokkaido University, Sapporo, Japan
jonni.virtema@let.hokudai.ac.jp

Abstract

Second-order Boolean logic is a generalization of QBF, whose constant alternation fragments are known to be complete for the levels of the exponential time hierarchy. We consider two types of restriction of this logic: 1) restrictions to term constructions, 2) restrictions to the form of the Boolean matrix. Of the first sort, we consider two kinds of restrictions: firstly, disallowing nested use of proper function variables, and secondly stipulating that each function variable must appear with a fixed sequence of arguments. Of the second sort, we consider Horn, Krom, and core fragments of the Boolean matrix. We classify the complexity of logics obtained by combining these two types of restrictions. We show that, in most cases, logics with k alternating blocks of function quantifiers are complete for the kth or $(k-1)$th level of the exponential time hierarchy. Furthermore, we establish **NL**-completeness for the Krom and core fragments, when $k = 1$ and both restrictions of the first sort are in effect.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases quantified Boolean formulae, computational complexity, second-order logic, Horn and Krom fragment

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.27

Funding The first and second authors were supported by grant 308712 of the Academy of Finland. The fourth author was an international research fellow of Japan Society for the Promotion of Science (Postdoctoral Fellowships for Research in Japan (Standard)).

Acknowledgements We wish to thank the anonymous referees for their very careful and thorough reviews.

1 Introduction

The canonical complete problem for **PSPACE** is the *quantified Boolean formula problem* (QBF) [17]. This generalization of the *Boolean satisfiability problem* (SAT) asks whether a Boolean sentence of the form $Q_1p_1 \dots Q_np_n\psi$, where $Q_i \in \{\exists, \forall\}$, is true. Today QBF attracts widespread interest in diverse research communities. In particular, QBF solving techniques are important in application domains such as planning, program synthesis and verification, adversary games, and non-monotonic reasoning, to name a few [15]. A further generalization of QBF is the *dependency quantified Boolean formula problem* (DQBF) [13, 12].

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 27; pp. 27:1–27:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This problem, complete for *nondeterministic exponential time* (**NEXP**), asks whether a Boolean sentence of the form

$$\forall p_1 \ldots \forall p_n \exists q_1 \ldots \exists q_m \psi$$

with constraints $C_i \subseteq \{p_1, \ldots, p_n\}$ is true; here, the selection of truth values for q_i may only depend on that of those variables that are in C_i. In other words, DQBF enriches QBF by allowing nonlinear dependency patterns between variables. DQBF-specifications can be exponentially more succinct compared to that of QBF and have found applications in areas such as non-cooperative games, SMT, and bit-vector logics. Furthermore, the development of DQBF-solvers is also well under way [14].

Put in different terms, DQBF instances can be seen as Boolean sentences of the form

$$\exists f_1 \ldots \exists f_m \forall p_1 \ldots \forall p_n \psi,$$

where each f_i is a Boolean function variable whose occurrences in ψ are of the form $f_i(p_{i_1}, \ldots, p_{i_k})$, for some fixed sequence of proposition variables $p_{i_1}, \ldots, p_{i_k}$. In previous studies, extensions of DQBF with alternating function quantification have also been considered. The so-called *alternating dependency quantified Boolean formula problem* (ADQBF) was shown to be complete for alternating exponential time with polynomially many alternations (**AEXP**(poly)) in [6]. This work was preceded by the works of Lück [9] and Lohrey [8] studying second-order Boolean logic with explicit quantification of Boolean functions (denoted SO_2 in this work). Their results showed, e.g., that restricting the alternations of function quantification to $k-1$ yields complete problems for the kth levels of the exponential hierarchy.

In this article we embark on a systematic study of the complexity of fragments of SO_2, defined by combining restrictions on the structure of function terms and the Boolean matrix. A remarkable fact is that, when restricting attention to Horn formulae, all the complexity distinctions between SAT, QBF, and DQBF disappear. Bubeck and Büning [4] showed that those DQBF instances whose quantifier-free part is a conjunction of Horn clauses are solvable in polynomial time. Consequently, all the aforementioned problems over Horn formulae are **P**-complete. This implies that the high complexity of (D)QBF is not a straightforward consequence of its quantification structure; rather, structural complexity from the quantifier-free part is also needed. An immediate question is: How complex quantification is required to neutralize structural limitations, such as the Horn form, on the quantifier-free part? It is exactly this interplay between quantification and quantifier-free formula structure that will be the focus of this paper.

A formula of SO_2 is in Σ_k or in Π_k if it is in prenex normal form with $k-1$ alternations for function quantification, with the first quantifier block being respectively existential or universal. If the quantifier-free part of a formula is in conjunctive normal form, then it is called (a) *Horn* if each clause has at most one positive literal, (b) *Krom* if each clause contains at most two literals, and (c) *core* if it is both Horn and Krom. A formula is called (i) *simple* if it contains no nested function terms, and (ii) *unique* if in it each function variable is associated with a unique argument tuple. These last two criteria, in particular, are meaningful for formulae involving second-order quantification. Uniqueness and simpleness are also the characteristics of function terms introduced in the process of Skolemization, and more importantly, tacitly assumed in the DQBF problem. One of the goals of this paper is to determine the impact of such restrictions. This way we generalize the aforementioned results on DQBF, which can be understood in our terms as unique simple Σ_1.

Table 1 Complexity of fragments of second-order Boolean logic restricted to Horn, Krom, or core clauses. All entries are completeness results with respect to logspace-reductions. The $\star$ means "any". "H"and "$\in$" are used for references for the hardness and membership results respectively. All trivial upper bounds, i.e., of the form $\Sigma_k^{\mathrm{E}}/\Pi_k^{\mathrm{E}}$, are by Theorem 7.
†: Likely identical with first row. ‡: The result follows from some other result in the table.

Simpleness	Uniqueness	k	Clauses	Σ_k	Π_k	Reference	
Simple	Unique	$k=1$	Horn	**P**	?	[4]	
			Krom/core	**NL**	**NL**	H/$\in$: 18	
		$k=2$	Horn	Σ_2^{E}	?	H/$\in$: ‡	
			Krom/core	Σ_2^{E}	**NL**	H: 25, $\in$: ‡	H/$\in$: 18
		$k \geq 3$ odd	$\star$	$\Sigma_{k-1}^{\mathrm{E}}$	Π_k^{E}	H: 26 $\in$: ‡	H: 25 $\in$: ‡
		$k \geq 4$ even	$\star$	Σ_k^{E}	Π_{k-1}^{E}	H: 25 $\in$: ‡	H: 26 $\in$: ‡
	Non-unique	$k=1$	Horn	**EXP**	Π_1^{E}	H/$\in$: 30	H/$\in$: ‡
			Krom/core	**PSPACE**	Π_1^{E}	H/$\in$: 29	H: 28, $\in$: ‡
		$k \geq 3$ odd	$\star$	$\Sigma_{k-1}^{\mathrm{E}}$	Π_k^{E}	H: ‡ , $\in$: 19	H/$\in$: ‡
		$k \geq 2$ even	$\star$	Σ_k^{E}	Π_{k-1}^{E}	H/$\in$: ‡	H: ‡, $\in$: 19
Non-simple	Unique	$k=1$	Horn	Σ_1^{E}	?†	H/$\in$: ‡	
			Krom/core	Σ_1^{E}	**NL**	H: 23, $\in$: ‡	H/$\in$: 18
		$k \geq 2$	$\star$	Σ_k^{E}	Π_k^{E}	H: 23 H: 24, $\in$: ‡	
	Non-unique	$k \geq 1$	$\star$	Σ_k^{E}	Π_k^{E}	H: 23, 24, 28, $\in$: [8]	
$\star$	$\star$	$k=\omega$	$\star$	**AEXP**(poly)	**AEXP**(poly)	H: 25, $\in$: [6]	

Our contributions are the following. We show, on the one hand, that the complexity of DQBF over Krom or core formulae collapses to **NL**, and that this result extends to simple and unique Π_1 and Π_2. On the other hand, we show that almost all other cases are complete for the corresponding, or their neighboring, levels of the exponential hierarchy. Some cases are left open; most intriguing such case is the inverse of the DQBF-Horn problem (i.e., simple and unique Π_1 Horn), which is only known to be between **NL** and Π_1^{E}. A summary of our results can be found in Table 1.

2 Second-order quantified Boolean formulae

Second-order propositional logic is obtained from usual quantified Boolean formulae by shifting from quantification over proposition variables to quantification over Boolean functions. We call this logic SO_2, as it essentially corresponds to second-order predicate logic restricted to the domain $\{0,1\}$.

2.1 Syntax and semantics

Let $\Phi = \{f_1, f_2, \ldots\}$ denote a countable set of function *variables*, each with an *arity* $\mathrm{ar}(f_i) \in \mathbb{N}$. We assume that there are infinitely many variables of any arity. Variables with arity 0 are called *propositional*. Variables with higher arity are called *proper function variables*. A *Φ-term* is either a propositional variable from Φ, or an expression of the form $f(t_1, \ldots, t_n)$, where $f \in \Phi$ is a variable of arity n and $t_1, \ldots, t_n$ are Φ-terms. The outermost variable in a term is called its *head*. The set of *subterms* $\mathsf{st}(t)$ of a term $t = f(t_1, \ldots, t_n)$ is recursively defined as $\{t\} \cup \bigcup_{i=1}^{n} \mathsf{st}(t_i)$. A term t appears *nested* in a term t' if $t \in \mathsf{st}(t') \setminus \{t'\}$. By identifying a term with its head, we also say that a *variable* appears nested in another term or variable. A *Φ-formula* is either a Φ-term, or an expression of the form $\varphi \wedge \varphi'$, $\neg\varphi$, or $\exists f \varphi$, where $f \in \Phi$ is a variable and φ, φ' are Φ-formulae. We write $\mathsf{SO}_2(\Phi)$ for the set of all Φ-formulae. We often omit Φ if it is clear from the context. The abbreviations $\forall f \varphi := \neg\exists f \neg\varphi$, $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \to \psi := \neg\varphi \vee \psi$ and $\varphi \leftrightarrow \psi := (\varphi \to \psi) \wedge (\psi \to \varphi)$ are defined in the usual fashion. We sometimes make use of the logical constants 0 and 1, which can be expressed with quantified propositions that are forced to take the appropriate truth values. If $\vec{f} = (f_1, \dots, f_n)$ is a tuple of variables, we sometimes write $\forall \vec{f}$ for $\forall f_1 \dots \forall f_n$ and $\exists \vec{f}$ for $\exists f_1 \dots \exists f_n$. Also, the formula $\vec{f} \leftrightarrow \vec{g}$, assuming $|\vec{f}| = |\vec{g}|$, is short for $\bigwedge_{i=1}^{|\vec{f}|} (f_i \leftrightarrow g_i)$.

We write $\mathrm{Var}(\varphi)$ ($\mathrm{Fr}(\varphi)$, resp.) to denote the set of variables that occur (occur freely, resp.) in φ. A formula with no free variables is *closed*. A term t is *free in* φ if $\mathrm{Var}(t) \subseteq \mathrm{Fr}(\varphi)$.

A *Φ-interpretation* I is a function that maps every variable $f \in \Phi$ to its interpretation $I(f)\colon \{0,1\}^{\mathrm{ar}(f)} \to \{0,1\}$. If I is a Φ-interpretation, $f \in \Phi$ has arity n, and $F\colon \{0,1\}^n \to \{0,1\}$, then I^f_F is the Φ-interpretation defined by $I^f_F(f) := F$ and $I^f_F(g) := I(g)$ for all $g \neq f$. The *valuation* $[\![\varphi]\!]_I \in \{0,1\}$ of a formula φ in I is defined as follows:

$$\begin{aligned}
[\![\varphi \wedge \psi]\!]_I &:= [\![\varphi]\!]_I \cdot [\![\psi]\!]_I,\\
[\![\neg\varphi]\!]_I &:= 1 - [\![\varphi]\!]_I,\\
[\![f(\varphi_1, \dots, \varphi_n)]\!]_I &:= I(f)([\![\varphi_1]\!]_I, \dots, [\![\varphi_n]\!]_I),\\
[\![\exists f \varphi]\!]_I &:= \max\left\{ [\![\varphi]\!]_{I^f_F} \;\middle|\; F : \{0,1\}^n \to \{0,1\} \right\}.
\end{aligned}$$

We often write $I \vDash \varphi$ instead of $[\![\varphi]\!]_I = 1$. We write $\varphi \vDash \psi$, if $I \vDash \varphi$ implies $I \vDash \psi$ for all suitable interpretations I. We say that φ and ψ are equivalent and write $\varphi \equiv \psi$, if $\varphi \vDash \psi$ and $\psi \vDash \varphi$. A Φ-formula φ is *valid* if $[\![\varphi]\!]_I = 1$ for all Φ-interpretations I. It is *satisfiable* if there is at least one I such that $[\![\varphi]\!]_I = 1$. Finally, a valid closed formula is called *true*.

2.2 Syntactic restrictions and normal forms

Next we consider basic normal forms of SO_2 such as *prenex form* and *conjunctive normal form*. These are defined as in classical QBF, except that a second-order literal may contain multiple variables in a nested way. Analogously to the classical case, we show that virtually all lower bounds already hold for those fragments. Here $[n]$ is used to denote the set of natural numbers $\{1, 2, \dots, n\}$.

▶ **Definition 1.** *A* literal *is a term or the negation of a term. A* clause *is a disjunction of literals. A formula in* conjunctive normal form (CNF) *is a conjunction of clauses. A formula is a* Horn formula *if it is a CNF such that every clause contains at most one non-negated literal. A formula is a* Krom formula *if it is a CNF such that every clause contains at most two literals. A formula is a* core formula *if it is Horn and Krom.*

▶ **Definition 2** (Σ_k and Π_k)**.** *Let $k \geq 1$. The set Σ_k consists of all formulae of the form $Q_1\vec{f_1} \cdots Q_k\vec{f_k}\, Q_{k+1}\,\vec{x}\,\theta$, where $Q_i = \exists$ ($Q_i = \forall$) if i is odd (even), θ is quantifier-free, and $\vec{x}$ is a tuple of propositional variables. Moreover, we insist that all quantified variables are distinct. The analogous definition of Π_k is achieved by swapping $\exists$ and $\forall$.*

For unbounded quantifier prefixes, we write $\Sigma_\omega := \bigcup_{k\geq 1} \Sigma_k$ and $\Pi_\omega := \bigcup_{k\geq 1} \Pi_k$. A formula φ is in *prenex form* if it is in $\Sigma_\omega \cup \Pi_\omega$. A formula in prenex form is called Horn, Krom, or core, if its quantifier-free part is a CNF of the corresponding form.

Compared to classical QBF, the structure of second-order literals is much richer due to the ability to use nested Boolean functions, and because we can have function variables appear with different arguments. In this paper, we explore the complexity landscape that results from allowing second-order literals to occur only in a controlled fashion. In extension to the fragments introduced above, we define two classes of formulae that play major roles in the subsequent results: uniqueness and simpleness.

▶ **Definition 3** (Uniqueness). *A formula φ has* uniqueness *if for all pairs of terms of the form $f(t_1, \dots, t_n)$ and $f(t'_1, \dots, t'_n)$ that occur in φ, it holds that $t_i = t'_i$ for all $i \in [n]$.*

In other words, a function variable must always appear with the same arguments. For example, the formulae $f(0) \leftrightarrow f(1)$ and $\exists x \forall y (x \leftrightarrow f(y))$ both state that f is a constant function, but only the second one has uniqueness.

▶ **Definition 4** (Simpleness). *A formula is* simple *if functions occurring in it have only propositions as arguments.*

If a formula is not simple, it is not hard to restore simpleness by introducing additional existential variables. For example, $f(g(x))$ is equivalent to $\exists y\, (g(x) \leftrightarrow y \wedge f(y))$.

▶ **Proposition 5.** *For every SO_2-formula φ in prenex form there is a logspace-computable and simple formula ψ equivalent to φ.*

Proof. Suppose $\varphi = Q_1 f_1 \cdots Q_n f_n\, \theta$ with θ quantifier-free. Let $t_1, \dots, t_k$ be an enumeration of all terms in θ. Then φ is equivalent to the formula

$$Q_1 f_1 \cdots Q_n f_n \exists y_1 \cdots \exists y_k \left(\theta^* \wedge \bigwedge_{i=1}^{k} (y_i \leftrightarrow t_i^*) \right),$$

where θ^* is obtained from θ by recursively replacing all terms t_j that occur nested inside other terms by y_j. ◀

▶ **Corollary 6.** *Let k be odd and let $\Psi \in \{\Pi_k, \Sigma_{k+1}\}$. Then for every formula $\varphi \in \Psi$ there is a logspace-computable formula $\psi \in \Psi$ that is simple and equivalent to φ. Furthermore, this translation preserves uniqueness, and the Horn, Krom and core property.*

If Ψ is a set of formulae, then Ψ^{s} is its restriction to simple formulae and Ψ^{u} is its restriction to formulae with uniqueness, and similarly Ψ^{h}, Ψ^{k} and Ψ^{c} for Horn, Krom, and core. E.g., Σ_2^{ush} is the set of all simple Σ_2-formulae with uniqueness which are in Horn CNF.

2.3 Known complexity results

We assume the reader to be familiar with basic complexity classes such as **PSPACE** and the exponential hierarchy, as well as logspace-reductions and basics of Turing machines. For a detailed exposition for these topics we refer the reader to [1] and to the complexity toolbox in Appendix A.

The quantifier alternation hierarchy of second-order Boolean logic is complete for the respective levels of the exponential time hierarchy, completely analogous to fragments of ordinary QBF being complete for the levels of the polynomial hierarchy.

▶ **Theorem 7** ([8, 9]). *Let $k \geq 1$. Truth of Σ_k-formulae is complete for Σ_k^{E}, and truth of Π_k-formulae is complete for Π_k^{E}.*

The result generalizes to unbounded number of quantifier alternations. The full logic is complete for the class **AEXP**(poly), that is, exponential runtime (corresponding to the size of second-order interpretations) but only polynomially many alternations (corresponding to the quantifier alternations in a formula with respect to its length).

▶ **Theorem 8** ([6, 9]). *Truth of $\Sigma^{\mathsf{us}}_{\omega}$-formulae, of $\Pi^{\mathsf{us}}_{\omega}$-formulae and of arbitrary SO_2-formulae is each complete for* $\mathbf{AEXP}(\mathrm{poly})$.

However, as Bubeck and Büning [4] showed, the complexity even of second-order logic can drop down to tractable classes when the matrix (i.e., the quantifier free part) of the formula is restricted to Horn clauses:

▶ **Theorem 9** ([4]). *Truth of $\Sigma^{\mathsf{ush}}_{1}$, that is, Σ_1-Horn formulae with simpleness and uniqueness, is* $\mathbf{P}$*-complete.*

2.4 Simplification based on variable dependencies

We conclude this section with a rather technical auxiliary result called *argument elision* that will be required in the subsequent sections. It allows to simplify formulae as follows. For example, the formula $\forall x\,\exists f\,\big(f(z,x) \leftrightarrow g(z)\big)$ can be simplified to an equivalent formula $\forall x\,\exists f_z\,\big(f_z(x) \leftrightarrow g(z)\big)$, for as the value of z is fixed to some $b \in \{0,1\}$ before f is quantified, the interpretations of f and f_z can be always copied from another such that $f_z(x)$ and $f(b,x)$ are the same functions. Hence the free variable z can be *elided* from the quantified function variable. Perhaps more relevant is the case where z is not free, but simply quantified before f. Indeed, the formulae $\forall z\forall x\exists f\,\big(f(z,x) \leftrightarrow g(z)\big)$ and $\forall z\forall x\exists f_z\,\big(f_z(x) \leftrightarrow g(z)\big)$ are equivalent.

Eliding the i-th position of a function variable f in a formula φ means to replace every quantifier Qf by Qg, where g is a fresh function variable of arity $\mathrm{ar}(f) - 1$ and $Q \in \{\exists, \forall\}$, and every term $f(t_1, \dots, t_n)$ with $g(t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_n)$. If a formula has uniqueness (i.e., functions always appear with the same arguments $t_1, \dots, t_n$) then *eliding a term t* from a function variable f means the consecutive elision of all positions i such that $t_i = t$.

The following proposition follows via a simple inductive argument (see Appendix B).

▶ **Proposition 10** (Free term elision). *Let $\varphi \in \mathsf{SO}^{\mathsf{u}}_2$ be a prenex formula, f a function variable not free in φ, and t a term free in φ. Then eliding t from f yields a formula equivalent to φ.*

In particular, it follows that if $\varphi \in \Sigma^{\mathsf{u}}_{\omega}$ is a formula, f a function variable quantified in φ, and t a term such that all variables in $\mathrm{Var}(t)$ are quantified before f, then the elision of t from f produces an equivalent formula.

3 An NL-complete second-order fragment

In this section, we consider the Krom fragment and obtain tractability results for the first levels of the propositional second-order quantifier hierarchy. We show completeness for $\mathbf{NL}$, and hence obtain fragments that are as hard as the ordinary propositional Krom fragment. In our proofs, we follow the classical approach by Aspvall et al. [2], who showed that classical QBF with the quantifier-free part consisting of Krom clauses are solvable in $\mathbf{NL}$. The approach is to interpret the formula as an *implication graph* $G = (V, E)$. The crucial idea of the approach is that connectedness in the graph corresponds to logical implication. Here, V is the set of all literals in φ, closed under negation and $\neg\neg\ell$ identified with ℓ. An edge $(\ell_1, \ell_2) \in E$ exists when φ contains a clause equivalent to $\ell_1 \to \ell_2$, that is, of the form $\neg\ell_1 \vee \ell_2$. A unit clause ℓ is identified with $(\neg\ell \to \ell)$. A *strongly connected component* (or simply a *component*) S of G is a maximal subset of vertices such that for all distinct $v, v' \in S$ there is a path from v to v'. sec:lower-bounds

In classical propositional logic, a set of Krom clauses is satisfiable precisely if no cycle of the implication graph contains some literal ℓ and its negation $\neg\ell$ [2]. With quantifiers, the matter complicates and we need to account for the notion of *dependency* between variables. A literal t is called *universal* (*existential*) in φ if its head is quantified universally (existentially) in φ. A component is *universal* (*existential*) if it contains some (no) universal vertex.

A bit sloppily, we say that a literal ℓ is an *argument* of a literal ℓ' if there are $r \geq 1$, $i \in [r]$ and a term $f(t_1, \ldots, t_r)$ such that ℓ or $\neg\ell$ equals t_i, and ℓ' or $\neg\ell'$ equals $f(t_1, \ldots, t_r)$. In what follows, we restrict ourselves to simple fragments, that is, all arguments are propositions.

▶ **Definition 11.** *A vertex v* depends on *a vertex v', in symbols $v \rightsquigarrow v'$, if*

a) *v' is an argument of v, or*

b) *v' is quantified before v, and every argument of v' is either an argument of v or*
- *is quantified before v, if the argument is universal, and*
- *is quantified before or at the same quantifier block as v, if the argument is existential.*

If S and S' are components, we write $S \rightsquigarrow S'$ if some universal *vertex $u \in S$ depends on some vertex $v \in S'$ (with possibly $S = S'$).*

For classical Krom formulae, a QBF can be shown to be true if and only if the following conditions all hold [2]:

(1) There is no path from a universal vertex u to another universal vertex u' (with $u \neq u'$, but possibly $u = \neg u'$).

(2) No vertices v and $\neg v$ are in the same component.

(3) Every existential vertex v in the same component as a universal vertex u must depend on u.

Note: For classical QBF, (3) simply means that v must be quantified after u, but in the general case, we need the more complicated Definition 11. Moreover, we require another condition in addition to the above (1)–(3):

(4) There is no $\rightsquigarrow$-cycle among the components (including loops).

▶ **Example 12.** One formula that violates (4) is $\forall y_1 \forall y_2 \exists x_1 \exists x_2 (y_1(x_2) \leftrightarrow x_1) \wedge (y_2(x_1) \leftrightarrow x_2)$. The reason is that $y_1(x_2) \rightsquigarrow x_2$ and $y_2(x_1) \rightsquigarrow x_1$, and therefore $\{y_1(x_2), x_1\} \rightsquigarrow \{x_2, y_2(x_1)\} \rightsquigarrow \{x_1, y_1(x_2)\}$ on the level of components. Indeed, choosing the universal quantifiers as $y_1(x_2) = \neg x_2, y_2(x_1) = x_1$ refutes the formula.

▶ **Example 13.** Another example is the false formula $\forall u \exists x (u(x) \leftrightarrow x)$. Informally, it states that every Boolean function has a fixed point. Indeed, u depends on x because x is an argument of u, and so the only component $\{u(x), x\}$ in this formula already forms a $\rightsquigarrow$-loop. (Also, x depends on u as it is quantified after u, but this fact is not required here. The formula $\exists x \forall u (u(x) \leftrightarrow x)$ is false as well.)

We carry the classical approach to the second-order setting, in particular to the fragment of formulae introduced next.

▶ **Definition 14** (Braided formulae)**.** *Let φ be a closed prenex formula, i.e., it is of the form $Q_1 \vec{f_1} \cdots Q_m \vec{f_m}\, \theta$, for θ quantifier-free. Then φ is* braided *if, for every quantifier Q_i, the arguments of each $g \in \vec{f_i}$ are quantified after g*

a) *in the quantifier blocks Q_i and Q_{i+1}, if Q_i is existential, and*

b) *in the quantifier blocks Q_i, Q_{i+1}, and Q_{i+2}, if Q_i is universal.*

Here, we restrict ourselves to braided $\Sigma_\omega^{\mathsf{usk}}$-formulae. That is, we consider only formulae of the form $Q_1 f_1 \cdots Q_m f_m \bigwedge_{i=1}^{k} C_k$, where $C_k = (\ell_k^1 \vee \ell_k^2)$ for literals ℓ_k^1, ℓ_k^2, and where terms do not contain nested proper functions.

Next, we prove that the conditions (1)–(4) are necessary for φ being true in the braided case. Afterwards, we show that they are also sufficient.

▶ **Lemma 15.** *Assume $\varphi \in \Sigma_\omega^{\mathsf{usk}}$ and braided. If any of* (1) *to* (4) *is violated, then φ is false.*

Proof. Let $G = (V, E)$ be the implication graph of φ.

(1) Let u and u' be distinct universal vertices such that (u, u') belongs to the transitive closure of E. Using an interpretation that maps u and u' to the constant functions 1 and 0, respectively, we can conclude that φ cannot be true.

(2) If v and $\neg v$ are vertices from the same component, it follows that φ can be true only if $v \leftrightarrow \neg v$ holds for some interpretation, which is clearly impossible.

(3) Let v and u be an existential and universal vertex from the same component, respectively, such that $v \not\rightsquigarrow u$. Hence u is not an argument of v. We proceed to a case distinction:

- **i)** The function v is quantified before u in φ: By the braided property, all the arguments of v (if there are any) are in the same quantifier block as v, or in the next one. Since changing the ordering of quantifiers in a universally quantified block does not have semantical consequences, we may stipulate that u is the final quantifier of its block. Hence all arguments of v are quantified before u as well. As a consequence, there is a fixed interpretation of terms such that v fully evaluates to either zero or one, but still must equal the universal u which is quantified later, which is impossible.
- **ii)** The function u is quantified before v: Since $v \not\rightsquigarrow u$, there must exist an argument z of u that is not an argument of v and that is quantified in some block strictly after the block where v is quantified (since v is existential). By the braided property, if u is quantified in a block Q_i it follows that v and z are quantified in the blocks Q_{i+1} and Q_{i+2} respectively. Hence z is universal. Similarly to i), the braided property also implies that all arguments of v are quantified in the quantifier blocks Q_{i+1} and Q_{i+2}. Hence using the same argument as in i), we may assume that z is the final quantifier in its block. Now by selecting u to be the projection function for the universally quantified z, we obtain an analogous contradiction as in i).

(4) Suppose there are components $S_1, \ldots, S_n$ such that $S_i \rightsquigarrow S_{i+1}$ for $i \in [n-1]$ and $S_n \rightsquigarrow S_1$. Let each S_i contain a universal vertex u_i and a vertex v_i such that $u_i \rightsquigarrow v_{i+1}$ for $i \in [n-1]$, and $u_n \rightsquigarrow v_1$. We describe choices of the universal quantifiers such that the formula becomes false. For $1 \leq i < n$, we can pick u_i such that it equals v_{i+1}; either as a projection function if v_{i+1} occurs among its arguments, or as a restriction of v_{i+1} to the set of common arguments of u_i and v_{i+1}. In the second case, every argument of v_{i+1} is also one of u_i or is quantified before u_i. Now the components $S_1, \ldots, S_n$ all have to receive the same truth value, regardless of the existential choices. Finally, u_n is picked as the *negation* of v_1, which renders the formula false. ◀

Next we proceed with the converse direction. We assume that the four above conditions are true, and from this construct a satisfying interpretation.

▶ **Lemma 16.** *Assume $\varphi \in \Sigma_\omega^{\mathsf{usk}}$. If* (1)–(4) *are satisfied, then φ is true.*

Proof. For this direction, we can roughly follow Aspvall et al. [2], but have to take into account that the vertices can also be proper functions.

Let $G = (V, E)$ be the implication graph of φ. The idea is to label the graph with truth values. Each component S in the graph is either unmarked, or marked with `true`, `false`, or `contingent`. Marking a component `true` or `false` means that it can in fact receive the corresponding truth value as a constant function, and `contingent` means that its truth depends on other vertices. Universal components are always contingent.

For every component S, the set $\neg S := \{\neg v \mid v \in S\}$ is again a component. Due to (2), S and $\neg S$ are always distinct. Moreover, the implication graph is *skew-symmetric* in the sense that there is an automorphism (modulo flipping all edges) mapping any literal to its negation. The reason is that the implication $\ell \to \ell'$ is clearly equivalent to $\neg\ell' \to \neg\ell$.

We are now in the position to construct an assignment. This assignment will be consistent in the sense that S is marked `true` iff $\neg S$ is marked `false`, and such that it satisfies all clauses due to the property that no path leads from a `true` component marked to a `false` one. First, we mark all universal components as `contingent`. We then consider the existential components in a reverse topological ordering with respect to E (there exists one, for the strongly connected components always induce an acyclic graph). The algorithm marks each component S in this order as follows.

- **i)** If S is already marked, proceed with the next component.
- **ii)** Otherwise S is existential and unmarked, but everything reachable by S is already marked. If S reaches any `contingent` or `false` component, mark it `false`; otherwise mark it `true`.
- **iii)** Mark $\neg S$ the opposite of S.

Now, whenever a component S is `false`, then either (in ii) it reaches some component marked `contingent` or `false`, or (in iii), by skew-symmetry, all components reaching it are `false`. Likewise, if S is `true`, then either (in ii) it reaches only components marked `true`, or (in iii), by skew-symmetry, it can be reached by a `contingent` or `true` component. Also, by condition (1), there is no path from one `contingent` component to another. It can be shown by induction on the steps of the algorithm, that there is no path from a `true` to a `contingent` or `false` component, and also none from a `contingent` to a `false` component.

All components marked `true` or `false` consist of existential vertices, so these can be assigned the corresponding truth assignment. Let us stress that here it suffices to assign constant functions regardless of the actual dependencies of the variables.

Next, fix some interpretation of the universally quantified variables. We continue the algorithm and refine the labeling of the universal components. By (4), it holds that there is no $\rightsquigarrow$-cycle between the components. This implies that there is again a reverse topological ordering $S_1, S_2, \ldots$ of all components, but now in the sense that $S_j \rightsquigarrow S_i$ implies $i < j$. We process all components in this order as follows.

- **i')** If S is not universal, or if it is already marked, proceed with the next component.
- **ii')** Otherwise, let u be the universal vertex in S (which is unique by (1)).
- **iii')** All dependencies of u are already marked `true` or `false`; in particular, all arguments of u have a marked truth value. Change S to `true` if u evaluates to 1 under the corresponding assignment, and otherwise to `false`.
- **iv')** Mark $\neg S$ the opposite of S.

It remains to establish that the interpretations of the existential variables in universal components can be always selected to mimic the truth value of the universal variable of its component. Recall that any existential vertex v in the component S must depend on u due to (3). This means that either (a) v is a function with u as an argument, or (b) v is quantified after u and has as arguments all arguments of u that are quantified in quantifier blocks after v. If (a) is the case, the we interpret v as the projection function for u. If (b) is the case, then there may be some arguments of u which are not arguments of v, but somewhere in the same quantifier block as v. But note that we may stipulate any fixed order of quantification inside a given quantifier block. Here, we assume that, inside a block, variables are quantified such that, for $i < j$, functions in S_i are quantified before functions in S_j. Then any variable that is quantified in the same block as v and is an argument of u but not of v is quantified before v, and hence has a fixed truth value when we give v its interpretation. Let A be the set of common arguments of v and u, and let $\vec{x}$ and $\vec{b}$ be the sequence of the arguments of u that are not in A and the truth values fixed for those vertices before v is interpreted, respectively. Now interpret v as the restriction of u to A with the determined arguments fixed $\vec{x} \mapsto \vec{b}$. In either case, we assigned v such that it equals u.

Since the above cannot introduce any new paths from a `true` component to a `false` component, all clauses of φ are satisfied. ◀

▶ **Theorem 17.** *The truth problem of braided $\Sigma^{\mathsf{usk}}_{\omega}$-formulae is in* **NL**.

Proof. By the above two lemmas, it suffices to check conditions (1)–(4). But these are simple reachability tests, which are easily solved in non-deterministic logspace. ◀

Next we apply the result to the lowest levels of the second-order quantifier hierarchy, namely Π^{s}_2-formulae and lower. Here, formulae are of the form

$$\forall f_1 \cdots \forall f_n \exists g_1 \cdots \exists g_m \forall x_1 \cdots \forall x_k \, \theta,$$

so the only terms violating the braided property could be of the form $v_1(\ldots, v_2, \ldots)$, where v_2 is quantified before v_1. But then the argument v_2 can be elided from v_1 by Proposition 10. Only for fragments Σ^{s}_2 or higher we can have formulae like $\exists f \forall g \exists x\, f(x)$ which are genuinely not braided, and which cannot be transformed by term elision. Finally, if the propositional quantifier block is existential (in the Π^{s}_1 fragment), we can omit the simpleness constraint due to Corollary 6. This yields the following collection of results, since **NL**-hardness holds already for the satisfiability of classical propositional core formulae (see, e.g., [11, Thm 16.3]).

▶ **Corollary 18.** *Truth of formulae in Σ^{usk}_1, Π^{uk}_1, Π^{usk}_1 or Π^{usk}_2, respectively, is* **NL***-complete. Also, the lower bound still holds for the respective restrictions to core formulae.*

Note that the above proof hinges on the fact that Lemma 15 works only for braided formulae. If we drop this assumption, then the complexity of the truth problem becomes as hard as for arbitrary formulae, as shown in Section 5.

4 Further Upper Bounds

In the previous section, we showed that the first level of the $\mathsf{SO}^{\mathsf{us}}_2$ hierarchy becomes tractable when restricted to Krom formulae. The same holds when restricted to Horn formulae [4]. Next, we consider the question whether these results can be generalized to higher levels of the SO_2 hierarchy. Indeed, we find several cases where the complexity collapses to a lower class. It is worthy to note that such a collapse occurs only if the final propositional quantifier block of a formula is universal, which also is the case, e.g., for the DQBF fragment (cf. Theorem 9). If the final quantifier block is existential, we show later in the next section that no such collapse occurs.

▶ **Theorem 19.** *Let $k > 0$ be even. Then the truth problem of $\Pi^{\mathsf{sk}}_k \cup \Pi^{\mathsf{sh}}_k$ is in Π^{E}_{k-1} and the truth problem of $\Sigma^{\mathsf{sk}}_{k+1} \cup \Sigma^{\mathsf{sh}}_{k+1}$ is in Σ^{E}_k.*

Proof. The following algorithm decides whether a given formula φ is true, if φ is simple and additionally Krom or Horn. Suppose $\varphi \in \Pi_k$ (resp. $\varphi \in \Sigma_{k+1}$).

First we non-deterministically guess in exponential time a truth table for each quantified function, except for the final block of existentially quantified functions, performing $k-2$ (resp. $k-1$) alternations in this process. All so evaluated quantifiers are deleted, and in either case we arrive at a formula φ' of the form $\exists f_1 \cdots \exists f_n \forall x_1 \cdots \forall x_m \theta$ for quantifier-free θ, and some interpretation I for the free variables in φ'. It remains to give a procedure that decides whether $I \vDash \varphi'$. If this part of the algorithm runs in deterministic exponential time w. r. t. $|\varphi|$, then this proves an overall Π^{E}_{k-1} or Σ^{E}_{k} bound, respectively.

To do so, we first perform some simplifications. W.l.o.g. $f_{o+1}, \ldots, f_n$ are propositions and $f_1, \ldots, f_o$ are proper functions, for some $o \in [n]$. We deterministically loop over all possible values for $f_{o+1}, \ldots, f_n$, substitute these in the formula, and remove the quantifiers. This leads only to an exponential factor in the runtime and ensures that all existentially quantified variables are proper functions. By this, we arrive at a Horn or Krom formula

$$\varphi'' = \exists f_1 \cdots \exists f_o \forall x_1 \cdots \forall x_m \theta'$$

for quantifier-free θ'. Note that φ'' may still contain free proper functions. But due to the simpleness condition, and since the f_i are functions as well, no existential variable is nested inside another function. This is crucial for the next step.

We use the *universal expansion* technique, which has been applied to DQBF as well [4]. The idea is to translate the universal quantifiers into an equivalent large conjunction. Let $r_i := \mathrm{ar}(f_i)$. We replace each existential variable f_i by exponentially many propositions $y_{i,\vec{a}}$, one for each possible input tuple $\vec{a} \in \{0,1\}^{r_i}$. For all possible assignments $\vec{b} \in \{0,1\}^m$ to the x_i, we create a modified copy $\theta'[\vec{b}]$ of the matrix θ' defined as follows. If $\vec{b} = (b_1, \ldots, b_m)$, then each x_i is replaced by b_i. Next, all terms t in $\theta'[\vec{b}]$ not containing any f_i are replaced by their valuation $[\![t]\!]_I \in \{0,1\}$. Now all terms are either constant, or have the head f_i and only constant arguments. Finally, the latter terms $f_i(b_1, \ldots, b_{r_i})$ are replaced by the proposition $y_{i,(b_1,\ldots,b_{r_i})}$. The resulting formula is the following:

$$\psi := \mathop{\exists}_{\substack{i\in[o]\\ \vec{a}\in\{0,1\}^{r_i}}} y_{i,\vec{a}} \bigwedge_{\vec{b}\in\{0,1\}^m} \theta'[\vec{b}]$$

This formula contains no free variables and is true if and only if $I \vDash \varphi''$. In other words, it is a simple propositional formula with existential proposition quantifiers, and its matrix $\bigwedge_{\vec{b}\in\{0,1\}^m} \theta'[\vec{b}]$ is Krom or Horn. Hence the truth of ψ can be computed in deterministic polynomial time w. r. t. $|\psi|$, and consequently in deterministic exponential time w. r. t. $|\varphi|$. ◀

If the non-deterministic part of the algorithm, the guessing of all quantified functions but the last block, is removed, then we obtain a deterministic exponential time algorithm for Σ^{sh}_1-formulae. the

▶ **Theorem 20.** *Truth of* Σ^{sh}_1 *is in* **EXP**.

In fact, we can combine this approach with the **NL** algorithm from Section 3 as well:

▶ **Theorem 21.** *Truth of* Σ^{sk}_1 *is in* **PSPACE**.

Proof. Given a formula $\varphi \in \Sigma^{\mathsf{sk}}_1$, we run the reachability algorithm from Section 3 on the formula ψ that would result from the translation in Theorem 19. However, instead of expanding φ to ψ first, which would require exponential space, we perform the reachability tests in polynomial space, constructing only the needed parts of ψ on-the-fly. ◀

Observe why the technique relies on the final quantifier block being universal: otherwise the resulting formula $\bigvee_{\vec{b}\in\{0,1\}^m} \theta'[\vec{b}]$ would not be in CNF, and hence neither Horn nor Krom.

5 Lower bounds

In the previous sections, we showed that the complexity of a fragment sometimes decreases when restricted to Horn or Krom matrix, when compared to the general fragment with the same quantifier prefix. However, in many cases the complexity stays the same. Often the logics are powerful enough to simulate specific Boolean connectives, such as disjunction and negation, in terms of quantified Boolean functions. In these cases, the whole Boolean part of the formula can essentially be reduced to unit clauses, which of course renders the Horn and Krom restriction meaningless.

5.1 Cases with an existential function quantifier

The first result of this section is also the most general; it concerns all non-simple formulae for quantifier prefixes that include $\mathit{\Sigma}_1$ – that is, everything but $\mathit{\Pi}_1$. (Recall that simple and non-simple $\mathit{\Pi}_1$ are equivalent.) By the introduction of additional existential functions that simulate disjunction and negation, we bring an arbitrary CNF into core form. This is stated in the following lemma, of which the proof can be found in Appendix C.

▶ **Lemma 22.** *Let $\mathcal{Q}\,\theta$ be a formula in CNF, with θ quantifier-free in CNF and $\mathcal{Q}$ being a sequence of quantifiers. Then $\mathcal{Q}\,\theta$ is equivalent to a logspace-computable formula $\exists \vec{f}\, \mathcal{Q}\, \forall \vec{y}\, \exists \vec{z} \theta'$ in CNF such that θ' is quantifier-free, $\vec{f}$ are function symbols, and $\vec{y}$, $\vec{z}$ are propositions. Moreover, if θ has uniqueness, then so has θ'.*

▶ **Theorem 23.** *For $k \geq 1$, truth of $\mathit{\Sigma}_k^{\mathsf{uc}}$ is Σ_k^{E}-complete.*

Proof. The upper bound is due to Theorem 7. For the lower bound, we use Lemma 22 and reduce from $\mathit{\Sigma}_k$, for which the truth is Σ_k^{E}-complete by Theorem 7. Let

$$\varphi = \exists \vec{f}_1 \forall \vec{f}_2 \cdots Q_k \vec{f}_k\, Q_{k+1} \vec{x}\, \theta$$

be given, where θ is quantifier-free, each $\vec{f}_i$ is a sequence of functions, and $\vec{x}$ is a sequence of propositions.

The first step is to transform φ to an equivalent formula with uniqueness. For any function h that violates uniqueness, we introduce fresh distinct copies $h_1, \ldots, h_n$ of h, for each distinct tuple of arguments $\vec{a}_1 \ldots \vec{a}_n$ of h occurring in φ, together with distinct fresh propositional variables $\vec{z}, \vec{z}_1, \ldots, \vec{z}_n$. We then append subformulae to φ whose purpose is to state that the interpretations of h_i and h coincide. If $Q_k = \exists$ and $Q_{k+1} = \forall$, we modify φ such that $\forall \vec{x}\, \theta$ is replaced with

$$\exists h_1 \ldots h_n \forall \vec{x}\, \vec{z}\, \vec{z}_1 \ldots \vec{z}_n \Big(\bigwedge_{i\in[n]} \big((\vec{z} \leftrightarrow \vec{z}_i) \to (h(\vec{z}) \leftrightarrow h_i(\vec{z}_i))\big)\Big) \wedge \Big(\big(\bigwedge_{i\in[n]} (\vec{z}_i \leftrightarrow \vec{a}_i)\big) \to \theta^*\Big),$$

where θ^* is obtained from θ by replacing the occurrences of $h(\vec{a}_i)$ by $h_i(\vec{z}_i)$, for each $i \in [n]$. On the other hand, if $Q_k = \forall$ and $Q_{k+1} = \exists$, we modify φ such that $\exists \vec{x}\, \theta$ is replaced with

$$\forall h_1 \ldots h_n \exists \vec{x}\, \vec{z}\, \vec{z}_1 \ldots \vec{z}_n \Big(\bigvee_{i\in[n]} \big((\vec{z} \leftrightarrow \vec{z}_i) \wedge (h(\vec{z}) \leftrightarrow \neg h_i(\vec{z}_i))\big)\Big) \vee \Big(\big(\bigwedge_{i\in[n]} (\vec{z}_i \leftrightarrow \vec{a}_i)\big) \wedge \theta^*\Big),$$

where θ^* is as above.

The second step is to establish CNF. It is folklore that arbitrary formulae can be translated into an equivalent CNF with the introduction of additional existentially quantified propositions after the final quantifier block $\vec{x}$. If k is odd, these existential propositions can be pulled in front of $\vec{x}$ (by increasing their arity and adding $\vec{x}$ as their parameter) and added to the (existential) block $\vec{f_k}$. If k is even this step can be skipped since $\vec{x}$ is existential as well. Hence we can assume that θ is in CNF and has uniqueness.

It remains to conduct the final translation into core clauses. For any Σ_k^{u}-formula $\varphi' = \exists\vec{f_1}\forall\vec{f_2}\cdots Q_k\vec{f_k}Q_{k+1}\vec{x}\,\theta'$, we can apply Lemma 22 to the subformula after $\exists\vec{f_1}$ and obtain an equivalent formula $\exists\vec{f_1}\exists\vec{g}\forall\vec{f_2}\cdots Q_k\vec{f_k}Q_{k+1}\vec{x}\forall\vec{y}\exists\vec{z}\,\theta''$ where θ'' is a quantifier-free CNF with uniqueness. Using the same argument as above, if $Q_{k+1} = \forall$, then we can eliminate the first-order alternation by transforming the $\vec{z}$ into existentially quantified functions depending on both $\vec{x}$ and $\vec{y}$ and adding them to the block $\vec{f_k}$. Otherwise, if $Q_{k+1} = \exists$, then we instead transform the $\vec{y}$ into functions depending on $\vec{x}$ and move them into the universal block $\vec{f_k}$. In each case, we arrive at an Σ_k^{uc}-formula. Note we do not consider the simpleness property at this point, since this step may produce new function symbols that appear nested in other functions. ◀

▶ **Theorem 24.** *For $k \geq 2$, truth of Π_k^{uc} is Π_k^{E}-complete.*

Proof. The proof is the same as for Theorem 23, except that in the last step, the formula is of the form

$$\forall\vec{f_1}\exists\vec{f_2}\forall\vec{f_3}\cdots Q_k\vec{f_k}Q_{k+1}\vec{x}\,\theta'$$

and we apply the lemma to the subformula after $\exists\vec{f_2}$. ◀

Lemma 22, used in the above reductions, introduces existential quantifiers that are not braided. Compared to the previous section, this small difference leads from **NL**-membership to Σ_k^{E}-completeness. If the final proposition block is existential *and* there is at least one existential function block, the result carries over even with simpleness due to Corollary 6:

▶ **Theorem 25.** **1.** *Let $k > 0$ be even. The truth problem of Σ_k^{usc} is Σ_k^{E}-complete and the truth problem of Π_{k+1}^{usc} is Π_{k+1}^{E}-complete.*

2. *The truth problem of $\Sigma_\omega^{\mathsf{usc}}$ is* **AEXP**(poly)*-complete.*

What if the proposition block is universal, i.e., k is odd for Σ_k and even for Π_k? Then, as shown in Theorem 19, we fall down one level in the hierarchy. Hardness results follow from the observation that Σ_k (Π_k, resp.) is a syntactic fragment of Σ_{k+1} (Π_{k+1}, resp.).

▶ **Theorem 26.** *Let $k > 2$ be odd. The truth problem of Σ_k^{usc} is $\Sigma_{k-1}^{\mathrm{E}}$-complete and the truth problem of Π_{k+1}^{usc} is Π_k^{E}-complete.*

5.2 The fragment Π_1 without uniqueness

We established the **NL** upper bound of Π_1 if we have uniqueness and Krom (Corollary 18); the case with uniqueness and Horn is open. Here, we proceed with Π_1 without uniqueness. As we have no existential function quantifiers, the reduction from before does not apply. Nonetheless, it turns out that this fragment is still as hard as the full logic Π_1.

▶ **Theorem 27.** *Truth of Π_1^{sc}-formulae is Π_1^{E}-hard.*

Proof. We reduce from the truth of arbitrary Π_1-formulae, which by Theorem 7 is Π_1^{E}-complete. Hence let φ be a Π_1-formula, i.e.,

$$\varphi = \forall f_1\cdots\forall f_n\exists x_1\cdots\exists x_m\,\theta$$

for function variables $f_1, \dots, f_n$, propositions $x_1, \dots, x_m$, and θ quantifier-free. Since the propositional quantifier block is existential, we can w.l.o.g. assume that θ is in 3CNF.[1]

The idea is to add $\forall g$ to the beginning of the formula, where g is a fresh binary function symbol, and to express in the reduction that g is the *nand* function, i.e., $g(b_1, b_2) = 1 - b_1 b_2$. In what follows, we use the constants 0 and 1, which can easily be simulated by adding new propositional quantifiers $\exists z_0 \exists z_1$ and unit clauses $\neg z_0 \wedge z_1$. To describe the behaviour of g, we add existentially quantified propositions d, d', e, e' and the following core clauses:

$$D_1 := g(0,0) \to d, \quad D_2 := g(0,0) \to e, \quad D_3 := g(d,0) \to d', \quad D_4 := g(0,e) \to e'.$$

Furthermore, every clause $C := (\ell_1 \vee \ell_2 \vee \ell_3)$ of θ is replaced by $e' \to g(d', C^*)$, where C^* is a *nand*-expression equivalent to $\neg C$, using g as a symbol for *nand*.[2] C^* has length $\mathcal{O}(|C|)$.

Call the resulting formula θ^*. To prove the correctness of the reduction, we show that θ is equivalent to $\theta' := \forall g \exists d \, \exists d' \, \exists e \, \exists e' (\bigwedge_{i=1}^{4} D_i \wedge \theta^*)$.

The easy direction is from right to left: Since g is universal, in particular we can assume that g is *nand*. As $g(0,0) = g(1,0) = g(0,1) = 1$, the propositions d, e, d', e' must all be true. Since also all clauses of the form $e' \to g(d', C^*)$ are true by assumption, C^* is false. Consequently, C is true.

For the converse direction, let g be arbitrary. We define suitable witnesses for d, e, d' and e'.

- If $g(0,0) = 0$, then we set $d, e, d', e' := 0$, which satisfies all clauses of the form $e' \to g(d', C^*)$, as well as $D_1, \dots, D_4$.
- If $g(0,1) = 0$, then we can similarly set $d, d', e := 1$ and $e' := 0$.
- Otherwise $g(0,0) = g(0,1) = 1$. Here, we must set $e, e', d := 1$.
  - If $g(1,0) = 0$, then we set $d' := 0$. Then $g(d', C^*) = g(0, C^*) = 1$ regardless of C^*.
  - If $g(1,0) = 1$, then we set $d' := 1$.
    - If $g(1,1) = 1$, then g is constant one, and the terms $g(d', C^*)$ are trivially true.
    - If $g(1,1) = 0$, then g is the actual *nand* function, and $g(d', C^*) \equiv \neg(1 \wedge C^*) \equiv C$ is true by assumption.

Finally, we replace θ by θ' in φ, move $\forall g$ to the front of the formula, and obtain simpleness of the formula by Corollary 6. ◀

The above results easily "relativize" to the case of more quantifier alternations before the final universal function quantifier block:

▶ **Theorem 28.** *Let $k > 0$ be odd. Then the truth of $\Sigma_{k+1}^{\mathsf{sc}}$ is $\Sigma_{k+1}^{\mathrm{E}}$-complete, and the truth of Π_k^{sc} is Π_k^{E}-complete.*

5.3 The Σ_1 cases with simpleness but no uniqueness

Curiously, while Π_1^{sc} is Π_1^{E}-complete, its dual fragment Σ_1^{sc} is likely easier than Σ_1^{E}, although harder than Σ_1^{usc}. We consider these final fragments in this subsection.

▶ **Theorem 29.** *Truth of formulae in Σ_1^{sc} or Σ_1^{sk} is* **PSPACE***-complete.*

Proof. The upper bound is given by Theorem 21. We show the hardness for Σ_1^{sc}, which implies the lower bound for Σ_1^{sk}. Let M be a single-tape Turing machine that decides some **PSPACE**-complete problem in deterministic space $p(n)$, where $p(n) \geq n$ is some polynomial.

[1] The approach is the same as for the classical reduction from SAT to 3SAT and can be found in standard textbooks (e.g. [1, Lemma 2.14]).

[2] We can choose for example $C^* := h(g(h(g(h(\ell_1), h(\ell_2))), h(\ell_3)))$, where $h(\varphi) = g(\varphi, \varphi)$.

W.l.o.g., we may assume that the computation of M halts in time $g(n)$ by reaching a unique rejecting or a unique accepting configuration, where $g(n)$ is some exponential function. For each input x, we compute a formula φ in logspace that is true iff M accepts x. The formula φ will be of the form

$$\exists f \, \forall v_1 \cdots \forall v_m \, \theta,$$

where θ is quantifier-free, simple and core, f is a function variable, and the v_i are propositions. Thus $\varphi \in \Sigma_1^{\mathsf{sc}}$.

If M has states Q and tape alphabet Γ, then a configuration of M is a triple (h, q, w), where $h \in [p(n)]$ denotes the head position on the tape, $q \in Q$ is the state of the machine, and $w \in \Gamma^{p(n)}$ is the tape content. We stipulate an arbitrary coding function $\langle\cdot\rangle : Q \cup \Gamma \to \{0,1\}^k$ that expands each state and each tape symbol to a fixed-width binary vector. For tape positions $j \in [p(n)]$, we use the unary encoding $\mathsf{bit}(j) := (0^{j-1}10^{p(n)-j})$. Using the coding function $\langle\cdot\rangle$, configurations of M can be now presented as binary strings of length $p(n) + k + kp(n)$.

The idea behind φ is as follows: The function f is used to encode a set of (binary encodings of) configurations of M. In order to take a head position, a state, and a tape content as an argument, the function f will have arity $p(n) + k + kp(n)$. In θ, we stipulate that f contains the initial configuration and is closed under transitions of M, but does not reach the unique rejecting configuration. Hence it expresses that M accepts x, as desired.

We will next describe θ more formally. Let M have initial state $q_0 \in Q$, and let $x = x_1 \cdots x_n$. First, we define the formula ψ_1 expressing that f contains the initial configuration:

$$\psi_1 := f(\mathsf{bit}(1); \langle q_0\rangle\, ; \langle x_1\rangle \cdots \langle x_n\rangle \; \langle\Box\rangle \cdots \langle\Box\rangle\,),$$

where $\Box \in \Gamma$ denotes the special symbol for blank. Next, ψ_2 states that f is closed under transitions of M (f may contain superfluous configurations, but this does not hurt the correctness of the reduction). Let $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\}$ be the transition function of M; e.g., if $\delta(q, a) = (q', b, -1)$, then M upon reading a in state q writes b, enters state q', and moves the head to the left. Define

$$\psi_2 := \forall \vec{v} \bigwedge_{\substack{j \in [p(n)] \\ \delta(q,a) = (q',a',i) \\ 1 \leq j+i \leq p(n)}} \Big(f(\mathsf{bit}(j); \langle q\rangle\, ; v_1 \cdots v_{k(j-1)} \, \langle a\rangle \, v_{kj+1} \cdots v_{kp(n)}) \\ \to f(\mathsf{bit}(j+i); \langle q'\rangle\, ; v_1 \cdots v_{k(j-1)} \, \langle a'\rangle \, v_{kj+1} \cdots v_{kp(n)})\Big),$$

where $\forall\vec{v}$ denotes $\forall v_1 \cdots \forall v_{kp(n)}$.

Finally, it remains to express that the rejecting configuration cannot be reached, which w.l.o.g. is a blank tape with M's head on the first position and in a designated state $q_r \in Q$.

$$\psi_3 := \neg f(\mathsf{bit}(1); \langle q_r\rangle\, ; \langle\Box\rangle \cdots \langle\Box\rangle\,)$$

By pulling the quantifiers in ψ_2 to the front, it is straightforward to see that $\exists f(\psi_1 \wedge \psi_2 \wedge \psi_3)$ is equivalent to a Σ_1-formula with only core clauses and with no nesting of functions, i.e., to a Σ_1^{sc}-formula. ◀

The proof of the following theorem is similar to that of Theorem 29. However, as an exponential time computation may require exponential space, some more care is required for the encodings. The computation is now encoded with a function that takes a tape address and the current timestep as arguments rather than the whole tape content. A detailed proof of the theorem can be found in Appendix D.

▶ **Theorem 30.** *Truth of formulae in Σ_1^{sh} is* **EXP***-complete.*

6 Summary

In this article, we studied the second-order quantifier hierarchy of Boolean logic. Boolean second-order logic, where quantifiers range over Boolean functions instead of mere propositions, can be seen as a generalization of logics such as DQBF that offer fine-grained control of dependencies between variables. Here, we turned to certain fragments where the propositional part is restricted to either Horn, Krom, or core formulae. Moreover, we introduced and considered two natural restrictions of second-order term constructions, namely simpleness (where proper function symbols cannot occur nested) and uniqueness (where all occurrences of a function have the same arguments). Using this terminology, DQBF is simple unique Σ_1.

We considered all possible combinations of these restrictions with respect to each level of the quantifier hierarchy, and obtained an almost complete classification of the computational complexity of the respective decision problem (cf. Table 1 on page 3). In almost all cases we obtained completeness results (with respect to logspace reductions). We showed that the complexity of Σ_1 and Π_1 formulae in Horn and/or Krom form collapse down to one of several classes that range from **NL** over **PSPACE** to **EXP**. Curiously, core Σ_1 stays $\mathsf{\Sigma}_1^{\mathrm{E}}$-hard if we lack simpleness, while core Π_1 stays $\mathsf{\Pi}_1^{\mathrm{E}}$-hard if we lack uniqueness. Moreover, Π_2 stays in **NL** if simple, unique, and Krom. For $k \geq 3$, for all considered restrictions to Σ_k (Π_k, resp.) the complexity either stays $\mathsf{\Sigma}_k^{\mathrm{E}}$-complete ($\mathsf{\Pi}_k^{\mathrm{E}}$-complete, resp.) or drops one level down to $\mathsf{\Sigma}_{k-1}^{\mathrm{E}}$ ($\mathsf{\Pi}_{k-1}^{\mathrm{E}}$, resp.) depending on uniqueness, simpleness, and whether k is even or odd. Furthermore, a direct corollary of the aforementioned results is that the complexity of $\Sigma_\omega^{\mathsf{usc}}$-formulae is **AEXP**(poly)-complete.

For the upper bounds, we mostly utilized generalizations of existing **NL** or **P** algorithms for classical Krom or Horn formulae. For the lower bounds, we introduced a number of different techniques; the common scheme being that one can exploit the ability to quantify functions to nullify the Horn and/or Krom restriction.

The most notable open case is that of simple unique Horn Π_1, which we conjecture to be **P**-complete, dually to the **P**-complete Σ_1 case (that is, DQBF-Horn [4]). Moreover, by Corollary 6, *non-simple* unique Π_1 has the same complexity. The final missing case, simple unique Π_2, likely reduces to these basic cases, but its complexity stays an open question for now as well.

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. URL: `http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264`.

2 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. `doi:10.1016/0020-0190(79)90002-4`.

3 Herbert Baier and Klaus W. Wagner. The Analytic Polynomial-Time Hierarchy. *Mathematical Logic Quarterly*, 44(4):529–544, 1998. URL: `http://onlinelibrary.wiley.com/doi/10.1002/malq.19980440412/abstract`.

4 Uwe Bubeck and Hans Kleine Büning. Dependency quantified horn formulas: Models and complexity. In *SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 198–211. Springer, 2006.

5 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. `doi:10.1145/322234.322243`.

6 Miika Hannula, Juha Kontinen, Martin Lück, and Jonni Virtema. On quantified propositional logics and the exponential time hierarchy. In Domenico Cantone and Giorgio Delzanno, editors,

Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016, volume 226 of *EPTCS*, pages 198–212, 2016. doi:10.4204/EPTCS.226.14.

7 Juris Hartmanis, Neil Immerman, and Vivian Sewelson. Sparse sets in NP-P: EXPTIME versus NEXPTIME. *Information and Control*, 65(2/3):158–181, 1985.

8 Markus Lohrey. Model-checking hierarchical structures. *J. Comput. Syst. Sci.*, 78(2):461–490, 2012. doi:10.1016/j.jcss.2011.05.006.

9 Martin Lück. Complete problems of propositional logic for the exponential hierarchy. *CoRR*, abs/1602.03050, 2016.

10 Pekka Orponen. Complexity classes of alternating machines with oracles. In Josep Díaz, editor, *Automata, Languages and Programming, 10th Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings*, volume 154 of *Lecture Notes in Computer Science*, pages 573–584. Springer, 1983. doi:10.1007/BFb0036938.

11 Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

12 Gary L. Peterson and John H. Reif. Multiple-person alternation. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 348–363. IEEE Computer Society, 1979. doi:10.1109/SFCS.1979.25.

13 Gary L. Peterson, John H. Reif, and Salman Azhar. Lower bounds for multiplayer noncooperative games of incomplete information. *Computers & Mathematics with Applications*, 41(7):957–992, 2001. doi:10.1016/S0898-1221(00)00333-3.

14 Christoph Scholl and Ralf Wimmer. Dependency quantified boolean formulas: An overview of solution methods and applications - extended abstract. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 2018. doi:10.1007/978-3-319-94144-8_1.

15 Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. A survey on applications of quantified boolean formulas. In *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019*, pages 78–84. IEEE, 2019. doi:10.1109/ICTAI.2019.00020.

16 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976. doi:10.1016/0304-3975(76)90061-X.

17 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

A Complexity toolbox

Alternating machines

We assume the reader to be familiar with basic complexity classes and notions such as Turing machines (TMs). We follow the definition of *alternating* TMs by Chandra et al. [5]. The states Q of such an alternating machine (ATM) are divided into disjoint sets $Q_\exists$ of *existential states* and $Q_\forall$ of *universal states*. Also, Q contains a designated *initial* state q_i, an *accepting* state q_a and a *rejecting* state q_r, where w.l.o.g. the initial state is always existential. A transition from an existential to a universal state, or vice versa, is called *alternation*. In this setting, a *non-deterministic* machine is one that never alternates, and a *deterministic* machine is one that provides at most one valid transition for every configuration.

A configuration is *accepting in k steps* if it contains no rejecting state, and furthermore either it contains an accepting state, or, provided $k > 0$, it contains an existential state and has a valid transition to a configurations that accepts in $k - 1$ steps, or contains an universal state and every valid transition leads to a configuration that accepts in $k-1$ steps. The *language decided by M* is the set of all inputs such that the initial configuration is accepting in k steps for some k.

As usual, the classes **EXP** and **NEXP** contain those problems which are decidable by a (non-)deterministic machine in time $2^{p(n)}$, for some polynomial p. Given a complexity class $\mathcal{C}$, its complement class is denoted by $\mathbf{co}\mathcal{C}$.

▶ **Definition 31.** *For $g(n) \geq 1$, the class $\mathbf{ATIME}(t(n), g(n))$ consists of the problems A for which there is an ATM deciding A in time $\mathcal{O}(t(n))$ with at most $g(n) - 1$ alternations on inputs of length n.*

▶ **Definition 32.** *For function classes $\mathcal{F}, \mathcal{G}$,*

$$\mathbf{ATIME}(\mathcal{F}, \mathcal{G}) := \bigcup_{f\in\mathcal{F}, g\in\mathcal{G}} \mathbf{ATIME}(f(n), g(n)).$$

▶ **Definition 33.**

$$\mathbf{AEXP} := \mathbf{ATIME}(2^{n^{\mathcal{O}(1)}}, 2^{n^{\mathcal{O}(1)}}), \qquad \mathbf{AEXP}(\text{poly}) := \mathbf{ATIME}(2^{n^{\mathcal{O}(1)}}, n^{\mathcal{O}(1)}).$$

Oracle machines

An *oracle Turing machine* is a Turing machine that additionally has an access to an *oracle* set B. The machine can query B by writing an instance x on a designated *oracle tape* and moving to a *query state* $q_?$. In the next configuration one of two states q_+ and q_- is assumed depending on whether $x \in B$ or not. There is no bound on the number of oracle queries during a computation of an oracle machine; the machine can erase the oracle tape and pose more queries.

If B is a language, then the usual complexity classes $\mathbf{P}, \mathbf{NP}, \mathbf{NEXP}$ etc. are generalized to $\mathbf{P}^B, \mathbf{NP}^B, \mathbf{NEXP}^B$ etc. where the definition is just changed from ordinary Turing machines to corresponding oracle machines with an oracle for B. If $\mathcal{C}$ is a class of languages, then $\mathbf{P}^{\mathcal{C}} := \bigcup_{B\in\mathcal{C}} \mathbf{P}^B$ and so on.

▶ **Definition 34** (The Polynomial Hierarchy [16]). *The levels of the polynomial hierarchy are defined inductively, where $k \geq 1$:*

- $\Sigma_0^{\mathrm{P}} = \Pi_0^{\mathrm{P}} = \Delta_0^{\mathrm{P}} := \mathbf{P}$.
- $\Sigma_k^{\mathrm{P}} := \mathbf{NP}^{\Sigma_{k-1}^{\mathrm{P}}}$, $\Pi_k^{\mathrm{P}} := \mathbf{coNP}^{\Sigma_{k-1}^{\mathrm{P}}}$, $\Delta_k^{\mathrm{P}} := \mathbf{P}^{\Sigma_{k-1}^{\mathrm{P}}}$.

▶ **Definition 35** (The Exponential Hierarchy [7]). *The levels of the exponential hierarchy are defined inductively, where $k \geq 1$:*

- $\Sigma_0^{\mathrm{E}} = \Pi_0^{\mathrm{E}} = \Delta_0^{\mathrm{E}} = \mathbf{EXP}$.
- $\Sigma_k^{\mathrm{E}} := \mathbf{NEXP}^{\Sigma_{k-1}^{\mathrm{P}}}$, $\Pi_k^{\mathrm{E}} := \mathbf{coNEXP}^{\Sigma_{k-1}^{\mathrm{P}}}$, $\Delta_k^{\mathrm{E}} := \mathbf{EXP}^{\Sigma_{k-1}^{\mathrm{P}}}$.

▶ **Theorem 36** ([5]). *For all $k \geq 1$:*

$$\Sigma_k^{\mathrm{P}} = \mathbf{ATIME}(n^{\mathcal{O}(1)}, k), \qquad \Pi_k^{\mathrm{P}} = \mathbf{co}\Sigma_k^{\mathrm{P}}.$$

Just as for the polynomial hierarchy, two competing definitions of Σ_k^{E} exist in the literature, one in terms of oracles and one as the class $\mathbf{ATIME}(2^{n^{\mathcal{O}(1)}}, k)$ [3, 8, 10].

▶ **Theorem 37** ([10]). *For all $k \geq 1$:*

$$\Sigma_k^{\mathrm{E}} = \mathbf{ATIME}(2^{n^{\mathcal{O}(1)}}, k), \qquad \Pi_k^{\mathrm{E}} = \mathbf{coATIME}(2^{n^{\mathcal{O}(1)}}, k).$$

A *logspace-reduction* from A to B is a logspace computable function f such that $x \in A \Leftrightarrow f(x) \in B$. If such f exists then A is *logspace-reducible* to B, in symbols $A \leq_{\mathrm{m}}^{\log} B$. If $A \in \mathcal{C}$ implies $A \leq_{\mathrm{m}}^{\log} B$, then B is $\leq_{\mathrm{m}}^{\log}$*-hard* for $\mathcal{C}$, and B is $\leq_{\mathrm{m}}^{\log}$*-complete* for $\mathcal{C}$ if $B \in \mathcal{C}$ and B is $\leq_{\mathrm{m}}^{\log}$-hard for $\mathcal{C}$. In this paper all reductions are logspace-reductions if not stated otherwise.

B Proof of Proposition 10

▶ **Proposition 10** (Free term elision). *Let $\varphi \in \mathsf{SO}_2^{\mathsf{u}}$ be a prenex formula, f a function variable not free in φ, and t a term free in φ. Then eliding t from f yields a formula equivalent to φ.*

Proof. Assume that φ, f and t are as above, and that $\mathrm{ar}(f) = n$ and g is a variable of arity $n-1$ that does not appear in φ. We prove that eliding the i-th argument of f yields an equivalent formula, where i is any position such that the i-th argument of f is t.

For a function F and $b \in \{0,1\}$, define the $(n-1)$-ary function

$$F_{|b}(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n) := F(a_1, \dots, a_{i-}, b, a_{i+1}, \dots, a_n).$$

Also, let $\varphi^\star$ be the formula φ with the i-th argument of f elided, i.e., f replaced by g and the i-th argument deleted in any occurrence of f as a term. For an interpretation I, define $I^\star$ like I except that $I^\star(g) := I(f)_{|I(t)}$. We show by induction on φ that $I(\varphi) = I^\star(\varphi^\star)$ for all interpretations I. It is easy to see that this proves the claim from the beginning, where neither f nor g appears free.

If φ does not contain t, and hence f, then we are done. Otherwise, if φ is of the form $f(t_1, \dots, t_{i-1}, t, t_{i+1}, \dots, t_n)$, then clearly

$$\begin{aligned} I(\varphi) &= I(f)(I(t_1), \dots, I(t_{i-1}), I(t), I(t_{i+1}), \dots, I(t_n)) \\ &= I(f)_{|I(t)}(I(t_1), \dots, I(t_{i-1}), I(t_{i+1}), \dots, I(t_n)) \\ &= I^\star(g)(I^\star(t_1), \dots, I^\star(t_{i-1}), I^\star(t_{i+1}), \dots, I^\star(t_n)) = I^\star(\varphi^\star). \end{aligned}$$

The inductive steps for applying function variables $h \neq f$, as well as for the Boolean connectives $\wedge$ and $\neg$, are straightforward. Also, the $\forall$-case can be reduced to $\exists$. It remains to consider the $\exists$-case. We divide this into the case where f is quantified and the case where any other function variable $h \neq f$ is quantified.

First, suppose $\varphi = \exists h \psi$, where $h \neq f$. Then whenever $I_H^h \vDash \psi$ for some I and H we have $(I^\star)_H^h = (I_H^h)^\star \vDash \psi^\star$, so $I^\star \vDash \varphi^\star$. Likewise, whenever $I_H^h \vDash \psi^\star$ for some I, then I_H^h is of the form $(J^\star)_H^h = (J_H^h)^\star$ for some J, so $J \vDash \varphi$.

Finally, let $\varphi = \exists f \psi$. If $I_F^f \vDash \psi$ for some I and F, then $(I_F^f)^\star \vDash \psi^\star$ by induction hypothesis. By definition, I^* and $(I_F^f)^\star$ agree everywhere except on f and g, and f does not occur in $\psi^\star$, so $I^* \vDash \exists g \psi^\star = \varphi^\star$ follows.

Suppose that conversely $I^\star \vDash \varphi^\star = \exists g \psi^\star$, so $(I^\star)_G^g \vDash \psi^\star$ for some I and G. As $(I^\star)_G^g = I_G^g$, also $I_G^g \vDash \psi^\star$. Define a function F from G as follows: Let $F(a_1, \dots, a_{i-1}, b, a_{i+1}, \dots, a_n) := G(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n)$ for both $b = 0$ and $b = 1$. Since f does not occur in $\psi^\star$, we can add it to any interpretation, so clearly $(I_F^f)_G^g \vDash \psi^\star$. Now notice that $G = F_{|0} = F_{|1} = F_{|I(t)}$. But then $(I_F^f)_G^g = (I_F^f)^\star$, so by induction hypothesis $I_F^f \vDash \psi$. Hence $I \vDash \exists f \psi = \varphi$. ◀

C Proof of Lemma 22

▶ **Lemma 22.** *Let $\mathcal{Q}\,\theta$ be a formula in CNF, with θ quantifier-free in CNF and $\mathcal{Q}$ being a sequence of quantifiers. Then $\mathcal{Q}\,\theta$ is equivalent to a logspace-computable formula $\exists \vec{f}\,\mathcal{Q}\,\forall \vec{y}\,\exists \vec{z}\,\theta'$ in CNF such that θ' is quantifier-free, $\vec{f}$ are function symbols, and $\vec{y}$, $\vec{z}$ are propositions. Moreover, if θ has uniqueness, then so has θ'.*

Proof. We begin with the case where $\mathcal{Q}$ is empty. The other cases are proved analogously.

Accordingly, let θ be of the form $\bigwedge_{i\in[n]} C_i$ with clauses $C_i = \ell^i_1 \vee \cdots \vee \ell^i_r$ and the ℓ^i_j being literals (i.e., terms or their negations). The idea of the proof is that all clauses C_i can be reformulated in terms of fresh Boolean functions h_i that act as disjunctions. Thus each clause becomes a single unit (and thus core) clause. Some auxiliary clauses are added in order to properly fix the disjunction as the interpretation of h_i.

We proceed as follows. First, for every literal ℓ in a clause C_i of θ, we introduce a fresh proposition p_ℓ that shall mirror ℓ and can appear inside h_i, since terms can only have other terms as their arguments, and not negations thereof. Next, we introduce a single proposition b the role of which we will explain below. Any clause $C_i = \ell^i_1 \vee \cdots \vee \ell^i_r$ is now replaced by the following conjunction $\xi(C_i)$ of core clauses:

$$\xi(C_i) := (b \leftrightarrow h_i(p_{\ell^i_1}, \ldots, p_{\ell^i_r})) \wedge \bigwedge_{k\in[r]} (p_{\ell^i_k} \to h_i(p_{\ell^i_1}, \ldots, p_{\ell^i_r}))$$

Let us start with the large conjunction on the right hand side: It ensures that the term $h_i(p_{\ell^i_1}, \ldots, p_{\ell^i_r})$ is true if any of its arguments is true. But in order to truly simulate C_i with h_i, we also need to achieve the converse. Otherwise h_i could still be a constant. However, we will quantify b universally, with the effect that the left hand side requires h_i to assume each value, zero and one, for some input. The value $h_i(p_{\ell^i_1}, \ldots, p_{\ell^i_r}) = 0$ can then only be assumed when all $p_{\ell^i_j}$ are zero, as required.

Furthermore, we need to impose some constraints on the proxies p_ℓ, so that p_ℓ in fact mirrors ℓ. In particular, exactly one of p_ℓ and $p_{\neg\ell}$ must be true. For every term t in θ, let g_t be another fresh binary function variable. Define

$$\begin{aligned}\tau(t) := \quad & (b \leftrightarrow g_t(p_t, p_{\neg t})) \wedge (p_t \to g_t(p_t, p_{\neg t})) \wedge (p_{\neg t} \to g_t(p_t, p_{\neg t})) \\ & \wedge (\neg p_t \vee \neg p_{\neg t}) \wedge (p_t \to t) \wedge (p_{\neg t} \to \neg t).\end{aligned}$$

Here, the first line again ensures that $g_t(p_t, p_{\neg t})$ is true if and only if p_t or $p_{\neg t}$ is true. So, when $b = 1$ then $g_t(p_t, p_{\neg t}) = 1$ and hence we know that at least one of p_t and $p_{\neg t}$ is true. The second line claims that at most one of them is true, and that this reflects the actual value of t. Observe that all used clauses are in core form.

Let now $t_1 \cdots t_s$ be a list of all terms occurring in the clauses of θ. Altogether, we translate $\theta = \bigwedge_{i\in[n]} C_i$ to φ as follows:

$$\varphi := \exists_{i\in[n]} h_i \; \exists_{i\in[s]} g_{t_i} \; \forall b \; \exists_{i\in[s]} p_{t_i} \; \exists_{i\in[s]} p_{\neg t_i} \bigwedge_{i\in[n]} \xi(C_i) \wedge \bigwedge_{i\in[s]} \tau(t_i)$$

Claim: θ and φ are logically equivalent.

- $\theta \vDash \varphi$: Suppose $I \vDash \theta$. We choose each h_i and g_t as the disjunction. Next, if $b = 0$, simply set all p_ℓ to zero. In turn, if $b = 1$, set p_ℓ to true if and only if $I \vDash \ell$. It is easy to check that this satisfies all $\xi(C_i)$ and $\tau(t_i)$. In particular, for each $h_i(p_{\ell^i_1}, \cdots, p_{\ell^i_r})$ there is $k \in [r]$ such that ℓ^i_k and hence $p_{\ell^i_k}$ must be true, as $I \vDash C_i$ by assumption.
- $\varphi \vDash \theta$: Suppose $I \vDash \varphi$. Then h_i and g_{t_i} are interpreted by some Boolean functions, and the p_ℓ by some truth values depending on b, such that all clauses in φ are true. In the case $b = 0$, the $h_i(\cdots)$ and $g_t(\cdots)$ must be false, and the same holds for all their arguments as well, due to the implications in $\xi(\cdots)$ and $\tau(\cdots)$. So $h_i(0, \ldots, 0) = g_t(0, 0) = 0$. In turn, in the case $b = 1$ it holds that $h_i(\cdots) = g_t(\cdots) = 1$, and hence at least one argument of each must have toggled its value. As a consequence, $\tau(t)$ forces that either p_t or $p_{\neg t}$ is true for every term t, and that p_ℓ is true iff $I \vDash \ell$. Likewise, for each $h_i(p_{\ell^i_1}, \ldots, p_{\ell^i_r})$ there is $k \in [r]$ such that $p_{\ell^i_k}$ and hence ℓ^i_k is true. In other words, all original clauses of θ are true in I.

The cases where θ contains quantifiers is proved analogously: b and the propositions $p_{t_i}, p_{\neg t_i}$ need to be quantified last in the prefix, as they depend on all other variables occurring in the formula, while the h_i and g_{t_i} are quantified first, since they can always just be set to the Boolean disjunction. Hence, the above proof works for arbitrary quantifier sequences $\mathcal{Q}$ and produces formulae of the form $\exists\vec{f}\mathcal{Q}\forall\vec{y}\exists\vec{z}$ as stated in the lemma. ◀

D Proof of Theorem 30

▶ **Theorem 30.** *Truth of formulae in Σ_1^{sh} is* **EXP***-complete.*

Proof. The upper bound is given by Theorem 20. For the lower bound, we modify the proof of Theorem 29 and encode all reachable configurations of an **EXP** computation using a single function variable f. However, since the computation can use exponential space, f now takes a tape *address*, rather than the whole tape content, as well as a current timestep in binary as an argument.

Let M be a single-tape TM that decides an **EXP**-complete problem, where M has states Q, initial state q_0, accepting state q_f, rejecting state q_r, tape alphabet Γ, and transition relation δ. This time, we consider as a configuration a word over $\Gamma' := \Gamma \cup (Q \times \Gamma)$. For example, $a(q,b)c$ means that the machine currently is in state q and reads b at tape position two. Suppose M runs in time $2^{p(n)}$ for some polynomial p, $p(n) \geq n$, and uses the tape positions $\{1, \ldots, 2^{p(n)} - 2\}$. For technical reasons, we "pad" configurations with blank symbols $\Box$ at positions 0 and $2^{p(n)} - 1$, but these cells will never be visited. Let $\langle\cdot\rangle : \Gamma' \to \{0,1\}^k$ be some fixed encoding. The function f is now of arity $k + kp(n) + kp(n)$. The intended meaning of $f(\langle\alpha\rangle\,; \mathsf{bin}(i); \mathsf{bin}(j))$ is that the ith symbol of the configuration on timestep j is α.

Let $x = x_1 \cdots x_n$ be the input. Let ℓ be minimal such that $n < 2^\ell$. We describe in the following formula that the first 2^ℓ symbols of the initial configuration are $\Box(q_0, x_1)x_2 \cdots x_n\Box \cdots \Box$ at timestep 0:

$$\psi_1 := f(\langle\Box\rangle\,; \mathsf{bin}(0); \mathsf{bin}(0)) \wedge f(\langle(q_0, x_1)\rangle\,; \mathsf{bin}(0); \mathsf{bin}(1)) \wedge \bigwedge_{i=2}^{n} f(\langle x_i\rangle\,; \mathsf{bin}(0); \mathsf{bin}(i)) \wedge \bigwedge_{i=n+1}^{2^\ell - 1} f(\langle\Box\rangle\,; \mathsf{bin}(0); \mathsf{bin}(i))$$

Then the next formula also fixes the remaining blank symbols $\Box$ on tape positions from 2^ℓ to $2^{p(n)} - 1$.

$$\psi_2 := \forall\vec{v} \bigwedge_{j=1}^{p(n)-\ell} f(\langle\Box\rangle\,; \mathsf{bin}(0); v_1, \ldots, v_{j-1}, 1, v_{j+1}, \ldots, v_{p(n)})$$

This is done by the third part of the arguments of f ranging over all numbers that have at least one of the first $p(n) - \ell$ bits set, which are $\{2^\ell, 2^\ell + 1, \ldots, 2^{p(n)} - 1\}$.

Next, we again state that M's rejecting configuration is *not* visited:

$$\psi_3 := \forall\vec{t}\forall\vec{u}\,\neg f(\langle(q_r, \Box)\rangle\,; \vec{t}; \vec{u})$$

Finally, it remains to express in formulae that f is closed under transitions of M. As in Theorem 29, this is the only part of the formula where we introduce non-unit clauses, which now will rather be Horn instead of core. For this, we use another function variable suc ("successor"), which has arity $2p(n)$, and for which every term of the form $\mathsf{suc}(\mathsf{bin}(m), \mathsf{bin}(m+1))$ is true. We show how to enforce this later; for now, we use it to impose the aforementioned closure condition on f.

We consider the set of valid *windows* of M. A window is a sixtuple $(a_1a_2a_3; a'_1a'_2a'_3) \in (\Gamma')^6$. For example, $(a(q,b)c; ad(q,c))$ means that M in state q when reading b writes d and moves to the right. Cells not currently visited by the head do not change (except for the head moving onto a cell), so $(abc; abc)$ and $(abc; (q,a)bc)$ are valid windows but $(abc; abd)$ is not. The set W of valid windows is finite and only depends on the transition function of M. The following formula states that, whenever $(a_1a_2a_3; a'_1a'_2a'_3)$ is a valid window, the middle tape cell must become (or stay) a'_2.

$$\psi_4 := \forall\vec{t}\vec{s}\vec{u}\vec{v}\vec{w} \bigwedge_{(a_1a_2a_3;a'_1a'_2a'_3)\in W} \Big(\big(\mathsf{suc}(\vec{t};\vec{s}) \land \mathsf{suc}(\vec{u};\vec{v}) \land \mathsf{suc}(\vec{v};\vec{w}) \\ \land f(\langle a_1\rangle;\vec{t};\vec{u}) \land f(\langle a_2\rangle;\vec{t};\vec{v}) \land f(\langle a_3\rangle;\vec{t};\vec{w})\big) \to f(\langle a'_2\rangle;\vec{s};\vec{v})\Big)$$

Here, $\vec{t}$ and $\vec{s}$ encode consecutive timesteps, and $\vec{u}\vec{v}\vec{w}$ are adjacent positions. The first and last position must be separately fixed to $\Box$ because they are never in the middle of a window:

$$\psi_5 := \forall\vec{t}\big(f(\langle\Box\rangle;\vec{t};\langle 0\rangle) \land f(\langle\Box\rangle;\vec{t};\left\langle 2^{p(n)}-1\right\rangle)\big)$$

Next, we specify suc and finish the reduction:

$$\varphi := \exists\mathsf{suc}\,\exists f\,\Big(\big(\forall\vec{v}\bigwedge_{i=0}^{p(n)-1}\mathsf{suc}(v_1,\ldots,v_i,0,1^{p(n)-i-1};v_1,\ldots,v_i,1,0^{p(n)-i-1})\big) \land \bigwedge_{i=1}^{5}\psi_i\Big)$$

Note that, just like f, the relation encoded by suc might contain more tuples than necessary, but again this does not hurt the reduction. It is easy to see that the formula can be transformed into a Σ_1 formula with simple matrix in Horn CNF. The Horn property of the formula hinges on ψ_4, for which it is crucial that M is deterministic. For this reason, this reduction cannot be generalized to, say, **NEXP**. ◀

Domain Theory in Constructive and Predicative Univalent Foundations

Tom de Jong
University of Birmingham, UK
https://www.cs.bham.ac.uk/~txd880
t.dejong@pgr.bham.ac.uk

Martín Hötzel Escardó
University of Birmingham, UK
https://www.cs.bham.ac.uk/~mhe
m.escardo@cs.bham.ac.uk

Abstract

We develop domain theory in constructive univalent foundations without Voevodsky's resizing axioms. In previous work in this direction, we constructed the Scott model of PCF and proved its computational adequacy, based on directed complete posets (dcpos). Here we further consider algebraic and continuous dcpos, and construct Scott's D_∞ model of the untyped λ-calculus. A common approach to deal with size issues in a predicative foundation is to work with *information systems* or *abstract bases* or *formal topologies* rather than dcpos, and *approximable relations* rather than Scott continuous functions. Here we instead accept that dcpos may be large and work with type universes to account for this. For instance, in the Scott model of PCF, the dcpos have carriers in the second universe $\mathcal{U}_1$ and suprema of directed families with indexing type in the first universe $\mathcal{U}_0$. Seeing a poset as a category in the usual way, we can say that these dcpos are large, but locally small, and have small filtered colimits. In the case of algebraic dcpos, in order to deal with size issues, we proceed mimicking the definition of accessible category. With such a definition, our construction of Scott's D_∞ again gives a large, locally small, algebraic dcpo with small directed suprema.

2012 ACM Subject Classification Theory of computation → Constructive mathematics; Theory of computation → Type theory

Keywords and phrases domain theory, constructivity, predicativity, univalent foundations

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.28

Related Version A version of this paper with full proofs is available at https://arxiv.org/abs/2008.01422.

1 Introduction

In domain theory [1] one considers posets with suitable completeness properties, possibly generated by certain elements called *compact*, or more generally generated by a certain *way-below* relation, giving rise to algebraic and continuous domains. As is well known, domain theory has applications to programming language semantics [42, 40, 33], higher-type computability [27], topology, topological algebra and more [19, 18].

In this work we explore the development of domain theory from the univalent point of view [46, 49]. This means that we work with the stratification of types as singletons, propositions, sets, 1-groupoids, etc., and that we work with univalence. At present, higher inductive types other than propositional truncation are not needed. Often the only consequences of univalence needed here are functional and propositional extensionality. An exception is the fundamental notion *has size*: if we want to know that it is a proposition, then univalence is necessary, but this knowledge is not needed for our purposes (Section 3). Full details of our univalent type theory are given in Section 2.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 28; pp. 28:1–28:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Additionally, we work constructively (we don't assume excluded middle or choice axioms) and predicatively (we don't assume Voevodky's resizing principles [47, 48, 49], and so, in particular, powersets are large). Most of the work presented here has been formalized in the proof assistant Agda [7, 17, 12] (see Section 7 for details). In our predicative setting, it is extremely important to check universe levels carefully, and the use of a proof assistant such as Agda has been invaluable for this purpose.

In previous work in this direction [10] (extended by Brendan Hart [20]), we constructed the Scott model of PCF and proved its computational adequacy, based on directed complete posets (dcpos). Here we further consider algebraic and continuous dcpos, and construct Scott's D_∞ model of the untyped λ-calculus [40].

A common approach to deal with size issues in a predicative foundation is to work with *information systems* [41], *abstract bases* [1] or *formal topologies* [38, 9] rather than dcpos, and *approximable relations* rather than (Scott) continuous functions. Here we instead accept that dcpos may be large and work with type universes to account for this. For instance, in our development of the Scott model of PCF [42, 33], the dcpos have carriers in the second universe $\mathcal{U}_1$ and suprema of directed families with indexing type in the first universe $\mathcal{U}_0$. Seeing a poset as a category in the usual way, we can say that these dcpos are large, but locally small, and have small filtered colimits. In the case of algebraic dcpos, in order to deal with size issues, we proceed mimicking the definition of accessible category [29]. With such a definition, our construction of Scott's D_∞ again gives a large, locally small, algebraic dcpo with small directed suprema.

Organization

Section 2: Foundations. *Section 3*: (Im)predicativity. *Section 4*: Basic domain theory, including directed complete posets, continuous functions, lifting, Ω-completeness, exponentials, powersets as dcpos. *Section 5*: Limit and colimits of dcpos, Scott's D_∞. *Section 6*: Way-below relation, bases, compact element, continuous and algebraic dcpos, ideal completion, retracts, examples. *Section 7*: Conclusion and future work.

Related Work

Domain theory has been studied predicatively in the setting of *formal topology* [38, 9] in [39, 30, 31, 28] and the more recent categorical paper [24]. In this predicative setting, one avoids size issues by working with abstract bases or formal topologies rather than dcpos, and approximable relations rather than Scott continuous functions. Hedberg [21] presented these ideas in Martin-Löf Type Theory and formalized them in the proof assistant ALF. A modern formalization in Agda based on Hedberg's work was recently carried out in Lidell's master thesis [26].

Our development differs from the above line of work in that it studies dcpos directly and uses type universes to account for the fact that dcpos may be large. There are two Coq formalizations of domain theory in this direction [5, 13]. Both formalizations study ω-chain complete preorders, work with setoids, and make use of Coq's impredicative sort `Prop`. Our development avoids the use of setoids thanks to the adoption of the univalent point of view. Moreover, we work predicatively and we work with directed sets rather than ω-chains, as we intend our theory to be also applicable to topology and algebra [19, 18].

There are also constructive accounts of domain theory aimed at program extraction [4, 32]. Both [4] and [32] study ω-chain complete posets (ω-cpos) and define notions of ω-continuity for them. Interestingly, Bauer and Kavkler [4] note that there can only be non-trivial examples of ω-continuous ω-cpos when Markov's Principle holds [4, Proposition 6.2]. This leads the authors of [32] to weaken the definition of ω-continuous ω-cpo by using the double negation of existential quantification in the definition of the way-below relation [32, Remark 3.2]. In light of this, it is interesting to observe that when we study directed complete posets (dcpos) rather than ω-cpos, and continuous dcpos rather than ω-continuous ω-cpos, we can avoid Markov's Principle or a weakened notion of the way-below relation to obtain non-trivial continuous dcpos (see for instance Examples 58, 59 and 82).

Another approach is the field of *synthetic domain theory* [37, 36, 22, 34, 35]. Although the work in this area is constructive, it is still impredicative, based on topos logic, but more importantly it has a focus different from that of regular domain theory: the aim is to isolate a few basic axioms and find models in (realizability) toposes where "every object is a domain and every morphism is continuous". These models often validate additional axioms, such as Markov's Principle and countable choice, and moreover falsify excluded middle. Our development has a different goal, namely to develop regular domain theory constructively and predicatively, but in a foundation compatible with excluded middle and choice, while not relying on them or Markov's Principle or countable choice.

2 Foundations

We work in intensional Martin-Löf Type Theory with type formers $+$ (binary sum), Π (dependent products), Σ (dependent sum), Id (identity type), and inductive types, including $\mathbf{0}$ (empty type), $\mathbf{1}$ (type with exactly one element $\star : \mathbf{1}$), $\mathbf{N}$ (natural numbers). Moreover, we have type universes (for which we typically write $\mathcal{U}$, $\mathcal{V}$, $\mathcal{W}$ or $\mathcal{T}$) with the following closure conditions. We assume a universe $\mathcal{U}_0$ and two operations: for every universe $\mathcal{U}$ a successor universe $\mathcal{U}^+$ with $\mathcal{U} : \mathcal{U}^+$, and for every two universes $\mathcal{U}$ and $\mathcal{V}$ another universe $\mathcal{U} \sqcup \mathcal{V}$ such that for any universe $\mathcal{U}$, we have $\mathcal{U}_0 \sqcup \mathcal{U} \equiv \mathcal{U}$ and $\mathcal{U} \sqcup \mathcal{U}^+ \equiv \mathcal{U}^+$. Moreover, $(-) \sqcup (-)$ is idempotent, commutative, associative, and $(-)^+$ distributes over $(-) \sqcup (-)$. We write $\mathcal{U}_1 :\equiv \mathcal{U}_0^+$, $\mathcal{U}_2 :\equiv \mathcal{U}_1^+, \dots$ and so on. If $X : \mathcal{U}$ and $Y : \mathcal{V}$, then $X + Y : \mathcal{U} \sqcup \mathcal{V}$ and if $X : \mathcal{U}$ and $Y : X \to \mathcal{V}$, then the types $\Sigma_{x:X} Y(x)$ and $\Pi_{x:X} Y(x)$ live in the universe $\mathcal{U} \sqcup \mathcal{V}$; finally, if $X : \mathcal{U}$ and $x, y : X$, then $\mathsf{Id}_X(x, y) : \mathcal{U}$. The type of natural numbers $\mathbf{N}$ is assumed to be in $\mathcal{U}_0$ and we postulate that we have copies $\mathbf{0}_{\mathcal{U}}$ and $\mathbf{1}_{\mathcal{U}}$ in every universe $\mathcal{U}$. All our examples go through with just two universes $\mathcal{U}_0$ and $\mathcal{U}_1$, but the theory is more easily developed in a general setting.

In general we adopt the same conventions of [46]. In particular, we simply write $x = y$ for the identity type $\mathsf{Id}_X(x, y)$ and use $\equiv$ for the judgemental equality, and for dependent functions $f, g : \Pi_{x:X} A(x)$, we write $f \sim g$ for the pointwise equality $\Pi_{x:X} f(x) = g(x)$.

Within this type theory, we adopt the univalent point of view [46]. A type X is a *proposition* (or *truth value* or *subsingleton*) if it has at most one element, i.e. the type $\mathsf{is\text{-}prop}(X) :\equiv \prod_{x,y:X} x = y$ is inhabited. A major difference between univalent foundations and other foundational systems is that we *prove* that types are propositions or properties. For instance, we can show (using function extensionality) that the axioms of directed complete poset form a proposition. A type X is a *set* if any two elements can be identified in at most one way, i.e. the type $\prod_{x,y:X} \mathsf{is\text{-}prop}(x = y)$ is inhabited.

We will assume two extensionality principles:

(i) *Propositional extensionality*: if P and Q are two propositions, then we postulate that $P = Q$ exactly when both $P \to Q$ and $Q \to P$ are inhabited.

(ii) *Function extensionality*: if $f, g : \prod_{x:X} A(x)$ are two (dependent) functions, then we postulate that $f = g$ exactly when $f \sim g$.

Function extensionality has the important consequence that the propositions form an exponential ideal, i.e. if X is a type and $Y : X \to \mathcal{U}$ is such that every $Y(x)$ is a proposition, then so is $\Pi_{x:X} Y(x)$. In light of this, universal quantification is given by Π-types in our type theory.

In Martin-Löf Type Theory, an element of $\prod_{x:X} \sum_{y:Y} \phi(x,y)$, by definition, gives us a function $f : X \to Y$ such that $\prod_{x:X} \phi(x, f(x))$. In some cases, we wish to express the weaker "for every $x : X$, there exists some $y : Y$ such that $\phi(x,y)$" without necessarily having an assignment of x's to y's. A good example of this is when we define directed families later (see Definition 7). This is achieved through the propositional truncation.

Given a type $X : \mathcal{U}$, we postulate that we have a proposition $\|X\| : \mathcal{U}$ with a function $|-| : X \to \|X\|$ such that for every proposition $P : \mathcal{V}$ in any universe $\mathcal{V}$, every function $f : X \to P$ factors (necessarily uniquely, by function extensionality) through $|-|$. Existential quantification $\exists_{x:X} Y(x)$ is given by $\|\Sigma_{x:X} Y(x)\|$. One should note that if we have $\exists_{x:X} Y(x)$ and we are trying to prove some proposition P, then we may assume that we have $x : X$ and $y : Y(x)$ when constructing our inhabitant of P. Similarly, we can define disjunction as $P \vee Q :\equiv \|P + Q\|$.

3 Impredicativity

We now explain what we mean by (im)predicativity in univalent foundations.

▶ **Definition 1** (Has size, `has-size` in [16]). *A type $X : \mathcal{U}$ is said to* have size $\mathcal{V}$ *for some universe $\mathcal{V}$ when we have $Y : \mathcal{V}$ that is equivalent to X, i.e. X has-size $\mathcal{V} :\equiv \sum_{Y:\mathcal{V}} Y \simeq X$.*

Here, the symbol $\simeq$ refers to Voevodsky's notion of equivalence [16, 46]. Notice that the type X has-size $\mathcal{V}$ is a proposition if and only if the univalence axiom holds [16].

▶ **Definition 2** (Type of propositions $\Omega_{\mathcal{U}}$). *The type of propositions in a universe $\mathcal{U}$ is $\Omega_{\mathcal{U}} :\equiv \sum_{P:\mathcal{U}} \mathsf{is\text{-}prop}(P) : \mathcal{U}^+$.*

Observe that $\Omega_{\mathcal{U}}$ itself lives in the successor universe $\mathcal{U}^+$. We often think of the types in some fixed universe $\mathcal{U}$ as *small* and accordingly we say that $\Omega_{\mathcal{U}}$ is *large*. Similarly, the powerset of a type $X : \mathcal{U}$ is large. Given our predicative setup, we must pay attention to universes when considering powersets.

▶ **Definition 3** ($\mathcal{V}$-powerset $\mathcal{P}_{\mathcal{V}}(X)$, $\mathcal{V}$-subsets). *Let $\mathcal{V}$ be a universe and $X : \mathcal{U}$ type. We define the* $\mathcal{V}$-powerset *$\mathcal{P}_{\mathcal{V}}(X)$ as $X \to \Omega_{\mathcal{V}} : \mathcal{V}^+ \sqcup \mathcal{U}$. Its elements are called* $\mathcal{V}$-subsets *of X.*

▶ **Definition 4** ($\in, \subseteq$). *Let x be an element of a type X and let A be an element of $\mathcal{P}_{\mathcal{V}}(X)$. We write $x \in A$ for the type $\mathsf{pr}_1(A(x))$. The first projection pr_1 is needed because $A(x)$, being of type $\Omega_{\mathcal{V}}$, is a pair. Given two $\mathcal{V}$-subsets A and B of X, we write $A \subseteq B$ for $\prod_{x:X}(x \in A \to x \in B)$.*

Functional and propositional extensionality imply that $A = B \iff A \subseteq B$ and $B \subseteq A$.

▶ **Definition 5** (Total type $\mathbb{T}(A)$). *Given a $\mathcal{V}$-subset A of a type X, we write $\mathbb{T}(A)$ for the* total type *$\sum_{x:X} x \in A$.*

One could ask for a *resizing axiom* asserting that $\Omega_{\mathcal{U}}$ has size $\mathcal{U}$, which we call *the propositional impredicativity of $\mathcal{U}$*. A closely related axiom is *propositional resizing*, which asserts that every proposition $P : \mathcal{U}^+$ has size $\mathcal{U}$. Without the addition of such resizing axioms, the type theory is said to be *predicative*. As an example of the use of impredicativity in mathematics, we mention that the powerset has unions of arbitrary subsets if and only if propositional resizing holds [16, `existence-of-unions-gives-PR`].

We mention that the resizing axioms are actually theorems when classical logic is assumed. This is because if $P \vee \neg P$ holds for every proposition in $P : \mathcal{U}$, then the only propositions (up to equivalence) are $\mathbf{0}_{\mathcal{U}}$ and $\mathbf{1}_{\mathcal{U}}$, which have equivalent copies in $\mathcal{U}_0$, and $\Omega_{\mathcal{U}}$ is equivalent to a type $\mathbf{2}_{\mathcal{U}} : \mathcal{U}$ with exactly two elements. The existence of a computational interpretation of propositional impredicativity axioms for univalent foundations is an open problem, however [45, 43].

4 Basic Domain Theory

Our basic ingredient is the notion of *directed complete poset* (dcpo). In set-theoretic foundations, a dcpo can be defined to be a poset that has least upper bounds of all directed families. A naive translation of this to our foundation would be to proceed as follows. Define a poset in a universe $\mathcal{U}$ to be a type $P : \mathcal{U}$ with a reflexive, transitive and antisymmetric relation $- \sqsubseteq - : P \times P \to \mathcal{U}$. According to the univalent point of view, we also require that the type P is a *set* and the values $p \sqsubseteq q$ of the order relation are *subsingletons*. Then we could say that the poset $(P, \sqsubseteq)$ is *directed complete* if every directed family $I \to X$ with indexing type $I : \mathcal{U}$ has a least upper bound. The problem with this definition is that there are no interesting examples in our constructive and predicative setting. For instance, assume that the poset $\mathbf{2}$ with two elements $0 \sqsubseteq 1$ is directed complete, and consider a proposition $A : \mathcal{U}$ and the directed family $A + \mathbf{1} \to \mathbf{2}$ that maps the left component to 1 and the right component to 0. By case analysis on its hypothetical supremum, we conclude that the negation of A is decidable. This amounts to weak excluded middle, which is known to be equivalent to De Morgan's Law, and doesn't belong to the realm of constructive mathematics. To try to get an example, we may move to the poset Ω_0 of propositions in the universe $\mathcal{U}_0$, ordered by implication. This poset does have all suprema of families $I \to \Omega_0$ indexed by types I in the *first universe* $\mathcal{U}_0$, given by existential quantification. But if we consider a directed family $I \to \Omega_0$ with I in the *same universe* as Ω_0 lives, namely the *second universe* $\mathcal{U}_1$, existential quantification gives a proposition in the *third universe* $\mathcal{U}_2$ and so doesn't give its supremum. In this example, we get a poset such that

(i) the carrier lives in the universe $\mathcal{U}_1$,
(ii) the order has truth values in the universe $\mathcal{U}_0$, and
(iii) suprema of directed families indexed by types in $\mathcal{U}_0$ exist.

Regarding a poset as a category in the usual way, we have a large, but locally small, category with small filtered colimits (directed suprema). This is typical of all the examples we have considered so far in practice, such as the dcpos in the Scott model of PCF and Scott's D_∞ model of the untyped λ-calculus. We may say that the predicativity restriction increases the universe usage by one. However, for the sake of generality, we formulate our definition of dcpo with the following universe conventions:

(i) the carrier lives in a universe $\mathcal{U}$,
(ii) the order has truth values in a universe $\mathcal{T}$, and
(iii) suprema of directed families indexed by types in a universe $\mathcal{V}$ exist.

So our notion of dcpo has three universe parameters $\mathcal{U}, \mathcal{V}, \mathcal{T}$. We will say that the dcpo is *locally small* when $\mathcal{T}$ is not necessarily the same as $\mathcal{V}$, but the order has truth values of size $\mathcal{V}$. Most of the time we mention $\mathcal{V}$ explicitly and leave $\mathcal{U}$ and $\mathcal{T}$ to be understood from the context.

▶ **Definition 6** (Poset). *A* poset $(P, \sqsubseteq)$ *is a set* $P : \mathcal{U}$ *together with a proposition-valued binary relation* $\sqsubseteq : P \to P \to \mathcal{T}$ *satisfying:*

(i) reflexivity*:* $\prod_{p:P} p \sqsubseteq p$*;*
(ii) antisymmetry*:* $\prod_{p,q:P} p \sqsubseteq q \to q \sqsubseteq p \to p = q$*;*
(iii) transitivity*:* $\prod_{p,q,r:P} p \sqsubseteq q \to q \sqsubseteq r \to p \sqsubseteq r$*.*

▶ **Definition 7** (Directed family). *Let $(P, \sqsubseteq)$ be a poset and I any type. A family $\alpha : I \to P$ is* directed *if it is inhabited (i.e. $\|I\|$ is pointed) and $\Pi_{i,j:I} \exists_{k:I} \alpha_i \sqsubseteq \alpha_k \times \alpha_j \sqsubseteq \alpha_k$.*

▶ **Definition 8** ($\mathcal{V}$-directed complete poset, $\mathcal{V}$-dcpo). *Let $\mathcal{V}$ be a type universe. A* $\mathcal{V}$-directed complete poset *(or* $\mathcal{V}$-dcpo*, for short) is a poset $(P, \sqsubseteq)$ such that every directed family $I \to P$ with $I : \mathcal{V}$ has a supremum in P.*

We will sometimes leave the universe $\mathcal{V}$ implicit, and simply speak of "a dcpo". On other occasions, we need to carefully keep track of universe levels. To this end, we make the following definition.

▶ **Definition 9** ($\mathcal{V}$-$\mathsf{DCPO}_{\mathcal{U},\mathcal{T}}$). *Let $\mathcal{V}$, $\mathcal{U}$ and $\mathcal{T}$ be universes. We write $\mathcal{V}$-$\mathsf{DCPO}_{\mathcal{U},\mathcal{T}}$ for the type of $\mathcal{V}$-dcpos with carrier in $\mathcal{U}$ and order taking values in $\mathcal{T}$.*

▶ **Definition 10** (Pointed dcpo). *A dcpo D is* pointed *if it has a least element, which we will denote by $\bot_D$, or simply $\bot$.*

▶ **Definition 11** (Locally small). *A $\mathcal{V}$-dcpo D is* locally small *if we have $\sqsubseteq_{\mathsf{small}} : D \to D \to \mathcal{V}$ such that $\prod_{x,y:D}(x \sqsubseteq_{\mathsf{small}} y) \simeq (x \sqsubseteq_D y)$.*

▶ **Example 12** (Powersets as pointed dcpos). Powersets give examples of pointed dcpos. The subset inclusion $\subseteq$ makes $\mathcal{P}_{\mathcal{V}}(X)$ into a poset and given a (not necessarily directed) family $A_{(-)} : I \to \mathcal{P}_{\mathcal{V}}(X)$ with $I : \mathcal{V}$, we may consider its supremum in $\mathcal{P}_{\mathcal{V}}(X)$ as given by $\lambda x.\exists_{i:I}\, x \in A_i$. Note that $(\exists_{i:I}\, x \in A_i) : \mathcal{V}$ for every $x : X$, so this is well-defined. Finally, $\mathcal{P}_{\mathcal{V}}$ has a least element, the empty set: $\lambda x.\mathbf{0}_{\mathcal{V}}$. Thus, $\mathcal{P}_{\mathcal{V}}(X) : \mathcal{V}$-$\mathsf{DCPO}_{\mathcal{V}^+ \sqcup \mathcal{U}, \mathcal{V} \sqcup \mathcal{U}}$. When $\mathcal{V} \equiv \mathcal{U}$ (as in Example 59), we get the simpler, locally small $\mathcal{P}_{\mathcal{U}}(X) : \mathcal{U}$-$\mathsf{DCPO}_{\mathcal{U}^+,\mathcal{U}}$. ⌟

Fix two $\mathcal{V}$-dcpos D and E.

▶ **Definition 13** (Continuous function). *A function $f : D \to E$ is* (Scott) continuous *if it preserves directed suprema, i.e. if $I : \mathcal{V}$ and $\alpha : I \to D$ is directed, then $f(\bigsqcup \alpha)$ is the supremum in E of the family $f \circ \alpha$.*

▶ **Lemma 14.** *If $f : D \to E$ is continuous, then it is monotone, i.e. $x \sqsubseteq_D y$ implies $f(x) \sqsubseteq_E f(y)$.*

▶ **Lemma 15.** *If $f : D \to E$ is continuous and $\alpha : I \to D$ is directed, then so is $f \circ \alpha$.*

▶ **Definition 16** (Strict function). *Suppose that D and E are pointed. A continuous function $f : D \to E$ is* strict *if $f(\bot_D) = \bot_E$.*

4.1 Lifting

▶ **Construction 17** ($\mathcal{L}_{\mathcal{V}}(X)$, η_X, cf. [10, 15]). Let $X : \mathcal{U}$ be a set. For any universe $\mathcal{V}$, we construct a pointed $\mathcal{V}$-dcpo $\mathcal{L}_{\mathcal{V}}(X) : \mathcal{V}$-$\mathsf{DCPO}_{\mathcal{V}^+ \sqcup \mathcal{U}, \mathcal{V}^+ \sqcup \mathcal{U}}$, known as the *lifting* of X. Its carrier is given by the type $\sum_{P:\mathcal{V}} \mathsf{is\text{-}prop}(P) \times (P \to X)$ of *partial elements* of X.

Given a partial element $(P, i, \varphi) : \mathcal{L}_{\mathcal{V}}(X)$, we write $(P, i, \varphi){\downarrow}$ for P and say that the partial element is defined if P holds. Moreover, we often leave the second component implicit, writing (P, φ) for (P, i, φ).

The order is given by $l \sqsubseteq_{\mathcal{L}_{\mathcal{V}}(X)} m :\equiv (l{\downarrow} \to l = m)$, and it has a least element given by $(\mathbf{0}, \mathbf{0}\text{-}\mathsf{is\text{-}prop}, \mathsf{unique\text{-}from\text{-}}\mathbf{0})$ where $\mathbf{0}$-$\mathsf{is\text{-}prop}$ is a witness that the empty type is a proposition and $\mathsf{unique\text{-}from\text{-}}\mathbf{0}$ is the unique map from the empty type.

Given a directed family $\left(Q_{(-)}, \varphi_{(-)}\right) : I \to \mathcal{L}_{\mathcal{V}}(X)$, its supremum is given by $(\exists_{i:I} Q_i, \psi)$, where ψ is such that

$$\begin{tikzcd} \sum_{i:I} Q_i \arrow[rr, "(i,q)\mapsto\varphi_i(q)"] \arrow[dr, "|-|"'] & & D \\ & \exists_{i:I} Q_i \arrow[ur, "\psi"'] & \end{tikzcd}$$

commutes. (This is possible, because the top map is weakly constant (i.e. any of its values are equal) and D is a set [25, Theorem 5.4].)

Finally, we write $\eta_X : X \to \mathcal{L}_{\mathcal{V}}(X)$ for the embedding $x \mapsto (\mathbf{1}, \mathbf{1}\text{-is-prop}, \lambda u.x)$. ⌟

Note that we require X to be a set, so that $\mathcal{L}_{\mathcal{V}}(X)$ is a poset, rather than an ∞-category. In practice, we often have $\mathcal{V} \equiv \mathcal{U}$ (see for instance Example 58, Section 5.2, or the Scott model of PCF [10]), but we develop the theory for the more general case. We can describe the order on $\mathcal{L}_{\mathcal{V}}(X)$ more explicitly, as follows.

▶ **Lemma 18.** *If we have elements (P, φ) and (Q, ψ) of $\mathcal{L}_{\mathcal{V}}(X)$, then $(P, \varphi) \sqsubseteq (Q, \psi)$ holds if and only if we have $f : P \to Q$ such that $\prod_{p:P} \varphi(p) = \psi(f(p))$.*

Observe that this exhibits $\mathcal{L}_{\mathcal{V}}(X)$ as locally small. We will show that $\mathcal{L}_{\mathcal{V}}(X)$ is the *free* pointed $\mathcal{V}$-dcpo on a set X, but to do that, we first need a lemma.

▶ **Lemma 19.** *Let D be a pointed $\mathcal{V}$-dcpo. Then D has suprema of families indexed by propositions in $\mathcal{V}$, i.e. if $P : \mathcal{V}$ is a proposition, then any $\alpha : P \to D$ has a supremum $\bigvee \alpha$.*

Moreover, if E is a (not necessarily pointed) $\mathcal{V}$-dcpo and $f : D \to E$ is continuous, then $f(\bigvee \alpha)$ is the supremum of the family $f \circ \alpha$.

Proof. Let D be a pointed $\mathcal{V}$-dcpo, $P : \mathcal{V}$ a proposition and $\alpha : P \to D$ a function. Now define $\beta : \mathbf{1}_{\mathcal{V}} + P \to D$ by $\mathsf{inl}(\star) \mapsto \bot_D$ and $\mathsf{inr}(p) \mapsto \alpha(p)$. Then, β is easily seen to be directed and so it has a well-defined supremum in D, which is also the supremum of α. The second claim follows from the fact that β is directed, so continuous maps must preserve its supremum. ◀

▶ **Lemma 20.** *Let $X : \mathcal{U}$ be a set and let (P, φ) be an arbitrary element of $\mathcal{L}_{\mathcal{V}}(X)$. Then $(P, \varphi) = \bigvee_{p:P} \eta_X(\varphi(p))$.*

▶ **Theorem 21.** *The lifting $\mathcal{L}_{\mathcal{V}}(X)$ gives the free $\mathcal{V}$-dcpo on a set X. Put precisely, if $X : \mathcal{U}$ is a set, then for every $\mathcal{V}$-dcpo $D : \mathcal{V}\text{-DCPO}_{\mathcal{U}',\mathcal{T}'}$ and function $f : X \to D$, there is a unique continuous function $\overline{f} : \mathcal{L}_{\mathcal{V}}(X) \to D$ such that*

$$\begin{tikzcd} X \arrow[rr, "f"] \arrow[dr, "\eta_X"'] & & D \\ & \mathcal{L}_{\mathcal{V}}(X) \arrow[ur, dashed, "\overline{f}"'] & \end{tikzcd}$$

commutes.

There is yet another way in which the lifting is a free construction, cf. [10, Section 4.3]. What is noteworthy about this is that freely adding subsingleton suprema automatically gives all directed suprema.

▶ **Definition 22** ($\Omega_{\mathcal{V}}$-complete). *A poset $(P, \sqsubseteq)$ is* $\Omega_{\mathcal{V}}$-complete *if it has suprema for all families indexed by a proposition in $\mathcal{V}$.*

▶ **Theorem 23.** *The lifting $\mathcal{L}_{\mathcal{V}}(X)$ gives the free $\Omega_{\mathcal{V}}$-complete poset on a set X. Put precisely, if $X : \mathcal{U}$ is a set, then for every $\Omega_{\mathcal{V}}$-complete poset $(P, \sqsubseteq)$ with $P : \mathcal{U}'$ and $\sqsubseteq$ taking values in $\mathcal{T}'$ and function $f : X \to P$, there exists a unique monotone $\overline{f} : \mathcal{L}_{\mathcal{V}}(X) \to P$ preserving all suprema indexed by propositions in $\mathcal{V}$, such that $\overline{f} \circ \eta_X = f$.*

Finally, a variation of Construction 17 freely adds a least element to a dcpo.

▶ **Construction 24** ($\mathcal{L}'_{\mathcal{V}}(D)$)**.** Let $D : \mathcal{V}\text{-}\mathsf{DCPO}_{\mathcal{U},\mathcal{T}}$ be a $\mathcal{V}$-dcpo. We construct a *pointed* $\mathcal{V}$-dcpo $\mathcal{L}'_{\mathcal{V}}(D) : \mathcal{V}\text{-}\mathsf{DCPO}_{\mathcal{V}^+ \sqcup \mathcal{U}, \mathcal{V} \sqcup \mathcal{T}}$. Its carrier is given by the type $\sum_{P:\mathcal{V}} \mathsf{is\text{-}prop}(P) \times (P \to D)$.

The order is given by $(P, \varphi) \sqsubseteq_{\mathcal{L}'_{\mathcal{V}}(D)} (Q, \psi) :\equiv \sum_{f:P\to Q} \prod_{p:P} \varphi(p) = \psi(f(p))$ and has a least element $(\mathbf{0}, \mathbf{0}\text{-}\mathsf{is\text{-}prop}, \mathsf{unique\text{-}from\text{-}}\mathbf{0})$.

Now let $\alpha :\equiv \big(Q_{(-)}, \varphi_{(-)}\big) : I \to \mathcal{L}'_{\mathcal{V}}(D)$ be a directed family. Consider $\Phi : \left(\sum_{i:I} Q_i\right) \to D$ given by $(i, q) \mapsto \varphi_i(q)$. The supremum of α is given by $(\exists_{i:I} Q_i, \psi)$, where ψ takes a witness that $\sum_{i:I} Q_i$ is inhabited to the directed (for which we needed $\exists_{i:I} Q_i$) supremum $\bigsqcup \Phi$ in D.

Finally, we write $\eta'_D : D \to \mathcal{L}'_{\mathcal{V}}(D)$ for the continuous map $x \mapsto (\mathbf{1}, \mathbf{1}\text{-}\mathsf{is\text{-}prop}, \lambda u.x)$. ⌟

▶ **Theorem 25.** *The construction $\mathcal{L}'_{\mathcal{V}}(D)$ gives the free pointed $\mathcal{V}$-dcpo on a $\mathcal{V}$-dcpo D. Put precisely, if $D : \mathcal{V}\text{-}\mathsf{DCPO}_{\mathcal{U},\mathcal{T}}$ is a $\mathcal{V}$-dcpo, then for every pointed $\mathcal{V}$-dcpo $E : \mathcal{V}\text{-}\mathsf{DCPO}_{\mathcal{U}',\mathcal{T}'}$ and continuous function $f : D \to E$, there is a unique strict continuous function $\overline{f} : \mathcal{L}'_{\mathcal{V}}(D) \to E$ such that $\overline{f} \circ \eta'_D = f$.*

4.2 Exponentials

▶ **Construction 26** (E^D)**.** Let D and E be two $\mathcal{V}$-dcpos. We construct another $\mathcal{V}$-dcpo E^D as follows. Its carrier is given by the type of continuous functions from D to E.

These functions are ordered pointwise, i.e. if $f, g : D \to E$, then

$$f \sqsubseteq_{E^D} g :\equiv \prod_{x:D} f(x) \sqsubseteq_E g(x).$$

Accordingly, directed suprema are also given pointwise. Explicitly, let $\alpha : I \to E^D$ be a directed family. For every $x : D$, we have the family $\alpha_x : I \to E$ given by $i \mapsto \alpha_i(x)$. This is a directed family in E and so we have a well-defined supremum $\bigsqcup \alpha_x : E$ for every $x : D$. The supremum of α is then given by $x \mapsto \bigsqcup \alpha_x$, where one should check that this assignment is indeed continuous.

Finally, if E is pointed, then so is E^D, because, in that case, the function $x \mapsto \bot_E$ is the least continuous function from D to E. ⌟

▶ Remark 27. In general, the universe levels of E^D can be quite large and complicated. For if $D : \mathcal{V}\text{-}\mathsf{DCPO}_{\mathcal{U},\mathcal{T}}$ and $D : \mathcal{V}\text{-}\mathsf{DCPO}_{\mathcal{U}',\mathcal{T}'}$, then $E^D : \mathcal{V}\text{-}\mathsf{DCPO}_{\mathcal{V}^+ \sqcup \mathcal{U} \sqcup \mathcal{T} \sqcup \mathcal{U}' \sqcup \mathcal{T}', \mathcal{U} \sqcup \mathcal{T}'}$. Even if $\mathcal{V} = \mathcal{U} \equiv \mathcal{T} \equiv \mathcal{U}' \equiv \mathcal{T}'$, the carrier of E^D still lives in the "large" universe $\mathcal{V}^+$. (Actually, this scenario cannot happen non-trivially in a predicative setting, since non-trivial dcpos cannot be "small" [11].) Even so, as observed in [10], if we take $\mathcal{U} \equiv \mathcal{T} \equiv \mathcal{U}' \equiv \mathcal{T}' \equiv \mathcal{U}_1$ and $\mathcal{V} \equiv \mathcal{U}_0$, then D, E, E^D are all elements of $\mathcal{U}_0\text{-}\mathsf{DCPO}_{\mathcal{U}_1,\mathcal{U}_1}$.

5 Scott's D_∞

We now construct, predicatively, Scott's famous pointed dcpo D_∞ which is isomorphic to its own function space $D_\infty^{D_\infty}$ (Theorem 39). We follow Scott's original paper [40] rather closely, but with two differences. Firstly, we explicitly keep track of the universe levels to make sure that our constructions go through predicatively. Secondly, [40] describes sequential (co)limits, while we study the more general directed (co)limits (Section 5.1) and then specialize to sequential (co)limits later (Section 5.2).

5.1 Limits and Colimits

▶ **Definition 28** (Deflation). *Let D be a dcpo. An endofunction $f : D \to D$ is a* deflation *if $f(x) \sqsubseteq x$ for all $x : D$.*

▶ **Definition 29** (Embedding-projection pair). *Let D and E be two dcpos. An* embedding-projection pair *from D to E consists of two continuous functions $\varepsilon : D \to E$ (the* embedding*) and $\pi : E \to D$ (the* projection*) such that:*

(i) *ε is a section of π;*
(ii) *$\varepsilon \circ \pi$ is a deflation.*

For the remainder of this section, fix the following setup. Let $\mathcal{V}$, $\mathcal{U}$, $\mathcal{T}$ and $\mathcal{W}$ be type universes. Let $(I, \sqsubseteq)$ be a directed preorder with $I : \mathcal{V}$ and $\sqsubseteq$ taken values in $\mathcal{W}$. Suppose that we have:

(i) for every $i : I$, a $\mathcal{V}$-dcpo $D_i : \mathcal{V}\text{-DCPO}_{\mathcal{U},\mathcal{T}}$;
(ii) for every $i, j : I$ with $i \sqsubseteq j$, an embedding-projection pair $(\varepsilon_{i,j}, \pi_{i,j})$ from D_i to D_j;

such that

(i) for every $i : I$, we have $\varepsilon_{i,i} = \pi_{i,i} = \mathsf{id}$;
(ii) for every $i, j, k : I$ with $i \sqsubseteq j \sqsubseteq k$, we have $\varepsilon_{i,k} \sim \varepsilon_{j,k} \circ \varepsilon_{i,j}$ and $\pi_{i,k} \sim \pi_{i,j} \circ \pi_{j,k}$.

▶ **Construction 30** (D_∞). Given the above inputs, we construct another $\mathcal{V}$-dcpo $D_\infty : \mathcal{V}\text{-DCPO}_{\mathcal{U}\sqcup\mathcal{V}\sqcup\mathcal{W},\mathcal{U}\sqcup\mathcal{T}}$ as follows. Its carrier is given by the type:

$$\sum_{\sigma : \prod_{i:I} D_i} \prod_{j:I, i \sqsubseteq j} \pi_{i,j}(\sigma_j) = \sigma_i.$$

These functions are ordered pointwise, i.e. if $\sigma, \tau : I \to D_i$, then

$$\sigma \sqsubseteq_{D_\infty} \tau :\equiv \prod_{i:I} \sigma_i \sqsubseteq_{D_i} \tau_i.$$

Accordingly, directed suprema are also given pointwise. Explicitly, let $\alpha : A \to D_\infty$ be a directed family. For every $i : I$, we have the family $A \to D_i$ given by $a \mapsto (\alpha(a))_i$, and denoted by α_i. One can show that α_i is directed and so we have a well-defined supremum $\bigsqcup \alpha_i : D_i$ for every $i : I$. The supremum of α is then given by the function $i : I \mapsto \bigsqcup \alpha_i$, where one should check that $\pi_{i,j}(\bigsqcup \alpha_j) = \bigsqcup \alpha_i$ holds whenever $i \sqsubseteq j$. ⌟

▶ Remark 31. We allow for general universe levels here, which is why D_∞ lives in the relatively complicated universe $\mathcal{U} \sqcup \mathcal{V} \sqcup \mathcal{W}$. In concrete examples, such as in Section 5.2, the situation simplifies to $\mathcal{V} = \mathcal{W} = \mathcal{U}_0$ and $\mathcal{U} = \mathcal{T} = \mathcal{U}_1$.

▶ **Construction 32** ($\pi_{i,\infty}$). For every $i : I$, we have a continuous function $\pi_{i,\infty} : D_\infty \to D_i$, given by $\sigma \mapsto \sigma_i$. ⌟

▶ **Construction 33** ($\varepsilon_{i,\infty}$). For every $i, j : I$, consider the function

$$\kappa : D_i \to \left(\sum_{k:I} i \sqsubseteq k \times j \sqsubseteq k\right) \to D_j$$
$$\kappa_x(k, l_i, l_j) = \pi_{i,j}(\varepsilon_{i,k}(x)).$$

Using directedness of $(I, \sqsubseteq)$, we can show that for every $x : D_i$ the map κ_x is weakly constant (i.e. all its values are equal). Therefore, we can apply [25, Theorem 5.4] and factor κ_x through $\exists_{k:I}(i \sqsubseteq k \times j \sqsubseteq k)$. But $(I, \sqsubseteq)$ is directed, so $\exists_{k:I}(i \sqsubseteq k \times j \sqsubseteq k)$ is a singleton. Thus, we obtain a function $\rho_{i,j} : D_i \to D_j$ such that: if we are given $k : I$ with $(l_i, l_j) : i \sqsubseteq k \times j \sqsubseteq k$, then $\rho_{i,j}(x) = \kappa_x(k, l_i, l_j)$.

Finally, this allows us to construct for every $i : I$, a continuous function $\varepsilon_{i,\infty} : D_i \to D_\infty$ by mapping $x : D_i$ to the function $\lambda(j : I).\rho_{i,j}(x)$. ⌟

▶ **Theorem 34.** *For every $i : I$, the pair $(\varepsilon_{i,\infty}, \pi_{i,\infty})$ is an embedding-projection pair.*

▶ **Lemma 35.** *Let $i, j : I$ such that $i \sqsubseteq j$. Then $\pi_{i,j} \circ \pi_{j,\infty} \sim \pi_i$, and $\varepsilon_{j,\infty} \circ \varepsilon_{i,j} \sim \varepsilon_{i,\infty}$.*

▶ **Theorem 36.** *The dcpo D_∞ with the maps $(\pi_{i,\infty})_{i:I}$ is the limit of $\left((D_i)_{i:I}, (\pi_{i,j})_{i \sqsubseteq j}\right)$.*

▶ **Theorem 37.** *The dcpo D_∞ with the maps $(\varepsilon_{i,\infty})_{i:I}$ is the colimit of $\left((D_i)_{i:I}, (\varepsilon_{i,j})_{i \sqsubseteq j}\right)$.*

The (co)limit property is in the category of continuous maps of dcpos.

5.2 Scott's Example Using Self-exponentiation

We now show that we can construct Scott's D_∞ [40] predicatively. Formulated precisely, we construct a pointed $D_\infty : \mathcal{U}_0\text{-}\mathsf{DCPO}_{\mathcal{U}_1,\mathcal{U}_1}$ such that D_∞ is isomorphic to its self-exponential $D_\infty^{D_\infty}$.

We employ the machinery from Section 5.1. Following [40, pp. 126–127], we inductively define pointed dcpos $D_n : \mathcal{U}_0\text{-}\mathsf{DCPO}_{\mathcal{U}_1,\mathcal{U}_1}$ for every natural number n:

(i) $D_0 :\equiv \mathcal{L}_{\mathcal{U}_0}(\mathbf{1}_{\mathcal{U}_0})$;
(ii) $D_{n+1} :\equiv D_n^{D_n}$.

Next, we inductively define embedding-projection pairs (ε_n, π_n) from D_n to D_{n+1}:

(i) $\varepsilon_0 : D_0 \to D_1$ is given by mapping $x : D_0$ to the continuous function that is constantly x; $\pi_0 : D_1 \to D_0$ is given by evaluating a continuous function $f : D_0 \to D_0$ at $\bot$;
(ii) $\varepsilon_{n+1} : D_{n+1} \to D_{n+2}$ takes a continuous function $f : D_n \to D_n$ to the continuous composite $D_{n+1} \xrightarrow{\pi_n} D_n \xrightarrow{f} D_n \xrightarrow{\varepsilon_n} D_{n+1}$; $\pi_{n+1} : D_{n+2} \to D_{n+1}$ takes a continuous function $f : D_{n+1} \to D_{n+1}$ to the continuous composite $D_n \xrightarrow{\varepsilon_n} D_{n+1} \xrightarrow{f} D_{n+1} \xrightarrow{\pi_n} D_n$.

In order to apply the machinery from Section 5.1, we will need embedding-projection pairs $(\varepsilon_{n,m}, \pi_{n,m})$ from D_n to D_m whenever $n \le m$. Let n and m be natural numbers with $n \le m$ and let k be the natural number $m - n$. We define the pairs by induction on k:

(i) if $k = 0$, then we set $\varepsilon_{n,n} = \pi_{n,n} = \mathsf{id}$;
(ii) if $k = l + 1$, then $\varepsilon_{n,m} = \varepsilon_{n+l} \circ \varepsilon_{n,n+l}$ and $\pi_{n,m} = \pi_{n,n+l} \circ \pi_{n+l}$.

So, Constructions 30, 32 and 33 give us $D_\infty : \mathcal{U}_0\text{-}\mathsf{DCPO}_{\mathcal{U}_1,\mathcal{U}_1}$ with embedding-projection pairs $(\varepsilon_{n,\infty}, \pi_{n,\infty})$ from D_n to D_∞ for every natural number n.

▶ **Lemma 38.** *Let n be a natural number. The function $\pi_n : D_{n+1} \to D_n$ is strict. Hence, so is $\pi_{n,m}$ whenever $n \le m$.*

▶ **Theorem 39.** *The dcpo D_∞ is pointed and isomorphic to $D_\infty^{D_\infty}$.*

▶ Remark 40. Of course, Theorem 39 is only interesting in case $D_\infty \not\simeq \mathbf{1}$. Fortunately, D_∞ has (infinitely) many elements besides $\bot_{D_\infty}$. For instance, we can consider $x_0 :\equiv \eta(\star) : D_0$ and $\sigma_0 : \prod_{n:\mathbf{N}} D_n$ given by $\sigma_0(n) :\equiv \varepsilon_{0,n}(x_0)$. Then, σ_0 is an element of D_∞ not equal to $\bot_{D_\infty}$, because $x_0 \neq \bot_{D_0}$.

6 Continuous and Algebraic Dcpos

We next consider dcpos generated by certain elements called compact, or more generally generated by a certain way-below relation, giving rise to algebraic and continuous domains.

6.1 The Way-below Relation

▶ **Definition 41** (Way-below relation, $x \ll y$)**.** *Let D be a $\mathcal{V}$-dcpo and $x, y : D$. We say that x is* way below *y, denoted by $x \ll y$, if for every $I : \mathcal{V}$ and directed family $\alpha : I \to D$, whenever we have $y \sqsubseteq \bigsqcup \alpha$, then there exists some element $i : I$ such that $x \sqsubseteq \alpha_i$ already. Symbolically,*

$$x \ll y :\equiv \prod_{I:\mathcal{V}} \prod_{\alpha:I\to D} \Big(\text{is-directed}(\alpha) \to y \sqsubseteq \bigsqcup \alpha \to \exists_{i:I}\, x \sqsubseteq \alpha_i\Big).$$

▶ **Lemma 42.** *The way-below relation enjoys the following properties.*
 (i) *It is proposition-valued.*
 (ii) *If $x \ll y$, then $x \sqsubseteq y$.*
 (iii) *If $x \sqsubseteq y \ll v \sqsubseteq w$, then $x \ll w$.*
 (iv) *It is antisymmetric.*
 (v) *It is transitive.*

▶ **Lemma 43.** *Let D be a dcpo. Then $x \sqsubseteq y$ implies $\prod_{z:D}(z \ll x \to z \ll y)$.*

▶ **Definition 44** (Compact)**.** *Let D be a dcpo. An element $x : D$ is called* compact *if $x \ll x$.*

▶ **Example 45.** The least element of a pointed dcpo is always compact.

▶ **Example 46** (Compact elements in $\mathcal{L}_{\mathcal{V}}(X)$)**.** Let $X : \mathcal{U}$ be a set. An element $(P, \varphi) : \mathcal{L}_{\mathcal{V}}(X)$ is compact if and only if P is decidable.

▶ **Definition 47** (Kuratowski finite)**.** *A type X is* Kuratowski finite *if there exists some natural number $n : \mathbf{N}$ and a surjection $e : \mathsf{Fin}(n) \twoheadrightarrow X$.*

That is, X is *Kuratowski finite* if its elements can be finitely enumerated, possibly with repetitions.

▶ **Example 48** (Compact elements in $\mathcal{P}_{\mathcal{U}}(X)$)**.** Let $X : \mathcal{U}$ be a set. An element $A : \mathcal{P}_{\mathcal{U}}(X)$ is compact if and only if its total type $\mathbb{T}A$ is Kuratowski finite.

6.2 Continuous Dcpos

Classically, a continuous dcpo is a dcpo where every element is the directed join of the set of elements way below it [3]. Predicatively, we must be careful, because if x is an element of a dcpo D, then $\sum_{y:D} y \ll x$ is typically large, so its directed join need not exist for size reasons. Our solution is to use a predicative version of bases [1] that accounts for size issues. For the special case of algebraic dcpos, our situation is the poset analogue of accessible categories [2]. Indeed, in category theory requiring smallness is common, even in impredicative settings, see for instance [23], where continuous dcpos are generalized to continuous categories.

▶ **Definition 49** (Basis, approximating family)**.** *A* basis *for $\mathcal{V}$-dcpo D is a function $\beta : B \to D$ with $B : \mathcal{V}$ such that for every $x : D$ there exists some $\alpha : I \to B$ with $I : \mathcal{V}$ such that*
 (i) *$\beta \circ \alpha$ is directed and its supremum is x;*
 (ii) *$\beta(\alpha_i) \ll x$ for every $i : I$.*
We summarise these requirements by saying that α is an approximating family *for x.*

Moreover, we require that $\ll$ is small when restricted to the basis. That is, we have $\ll^B : B \to B \to \mathcal{V}$ such that $(\beta(b) \ll \beta(b')) \simeq \big(b \ll^B b'\big)$ for every $b, b' : B$.

▶ **Definition 50** (Continuous dcpo)**.** *A dcpo D is* continuous *if there exists some basis for it.*

We postpone giving examples of continuous dcpos until we have developed the theory further, but the interested reader may look ahead to Examples 58, 59 and 82.

A useful property of bases is that it allows us to express the order fully in terms of the way-below relation, giving a converse to Lemma 43.

▶ **Lemma 51.** *Let D be a dcpo with basis $\beta : B \to D$. Then $x \sqsubseteq y$ holds if and only if $\prod_{b:B}(\beta(b) \ll x \to \beta(b) \ll y)$.*

▶ **Lemma 52.** *Let D be a $\mathcal{V}$-dcpo with a basis $\beta : B \to D$. Then $\sqsubseteq$ is small when restricted to the basis, i.e. $\beta(b_1) \sqsubseteq \beta(b_2)$ has size $\mathcal{V}$ for every two elements $b_1, b_2 : B$. Hence, we have $\sqsubseteq^B : B \to B \to \mathcal{V}$ such that $\prod_{b_1,b_2:B}\big(b_1 \sqsubseteq^B b_2\big) \simeq (\beta(b_1) \sqsubseteq \beta(b_2))$.*

The most significant properties of a basis are the interpolation properties. We consider nullary, unary and binary versions here. The binary interpolation property actually follows fairy easily from the unary one, but we still record it, because we wish to show that bases are examples of the abstract bases that we define later (cf. Example 62). Our proof of unary interpolation is a predicative version of [14].

▶ **Lemma 53** (Nullary interpolation)**.** *Let D be a dcpo with a basis $\beta : B \to D$. For every $x : D$, there exists some $b : B$ such that $\beta(b) \ll x$.*

▶ **Lemma 54** (Unary interpolation)**.** *Let D be a $\mathcal{V}$-dcpo with basis $\beta : B \to D$ and let $x, y : D$. If $x \ll y$, then there exists some $b : B$ such that $x \ll \beta(b) \ll y$.*

▶ **Lemma 55** (Binary interpolation)**.** *Let D be a $\mathcal{V}$-dcpo with basis $\beta : B \to D$ and let $x, y, z : D$. If $x, y \ll z$, then there exists some $b : B$ such that $x, y \ll \beta(b) \ll z$.*

6.3 Algebraic Dcpos

We now turn to a particular class of continuous dcpos, called algebraic dcpos.

▶ **Definition 56** (Algebraic dcpo)**.** *A dcpo D is* algebraic *if there exists some basis $\beta : B \to D$ for it such that $\beta(b)$ is compact for every $b : B$.*

▶ **Lemma 57.** *Let D be a $\mathcal{V}$-dcpo. Then D is algebraic if and only if there exists $\beta : B \to D$ with $B : \mathcal{V}$ such that*

(i) *every element $\beta(b)$ is compact;*

(ii) *for every $x : D$, there exists $\alpha : I \to B$ with $I : \mathcal{V}$ such that $\beta \circ \alpha$ is directed and $x = \bigsqcup \beta \circ \alpha$.*

▶ **Example 58** ($\mathcal{L}_{\mathcal{U}}(X)$ is algebraic)**.** Let $X : \mathcal{U}$ be a set and consider $\mathcal{L}_{\mathcal{U}}(X) : \mathcal{U}\text{-}\mathsf{DCPO}_{\mathcal{U}^+,\mathcal{U}^+}$. The basis $[\bot, \eta_X] : (\mathbf{1}_{\mathcal{U}} + X) \to \mathcal{L}_{\mathcal{U}}(X)$ exhibits $\mathcal{L}_{\mathcal{U}}(X)$ as an algebraic dpco.

Proof. By Example 46, the elements $\bot$ and $\eta_X(x)$ (with $x : X$) are all compact, so it remains to show that $\mathbf{1}_{\mathcal{U}} + X$ is indeed a basis. Recalling Lemmas 19 and 20, we can write any element $(P, \varphi) : \mathcal{L}_{\mathcal{V}}(X)$ as the directed join $\bigsqcup([\bot, \eta_X] \circ \alpha)$ with $\alpha :\equiv [\mathsf{id}, \varphi] : (\mathbf{1}_{\mathcal{U}} + P) \to (\mathbf{1}_{\mathcal{U}} + X)$. By Lemma 57 the proof is finished. ◁

▶ **Example 59** ($\mathcal{P}_{\mathcal{U}}(X)$ is algebraic)**.** Let $X : \mathcal{U}$ be a set and consider $\mathcal{P}_{\mathcal{U}}(X) : \mathcal{U}\text{-}\mathsf{DCPO}_{\mathcal{U}^+,\mathcal{U}}$. The basis $\iota : \mathsf{List}(X) \to \mathcal{P}_{\mathcal{U}}(X)$ that maps a finite list to a Kuratowski finite subset exhibits $\mathcal{P}_{\mathcal{U}}(X)$ as an algebraic dpco.

▶ **Example 60** (Scott's D_∞ is algebraic)**.** The pointed dcpo $D_\infty : \mathcal{U}_0\text{-}\mathsf{DCPO}_{\mathcal{U}_1,\mathcal{U}_1}$ with $D_\infty \cong D_\infty^{D_\infty}$ from Section 5.2 is algebraic.

6.4 Ideal Completion

Finally, we consider how to build dcpos from posets, or more generally from abstract bases, using the rounded ideal completion [1, Section 2.2.6]. Given our definition of the notion of dcpo, the reader might expect us to define ideals using families rather than subsets. However, we use subsets for extensionality reasons. Two subsets A and B of some X are equal exactly when $x \in A \iff x \in B$ for every $x : X$. However, given two (directed) families $\alpha : I \to X$ and $\beta : J \to X$, it is of course not the case (it does not even typecheck) that $\alpha = \beta$ when $\Pi_{i:I}\exists_{j:J}\alpha_i = \beta_j$ and $\Pi_{j:J}\exists_{i:I}\beta_j = \alpha_i$ hold. We could try to construct the ideal completion by quotienting the families, but then it seems impossible to define directed suprema in the ideal completion without resorting to choice.

▶ **Definition 61** (Abstract basis). *A pair $(B, \prec)$ with $B : \mathcal{V}$ and $\prec$ taking values in $\mathcal{V}$ is called a $\mathcal{V}$-*abstract basis *if:*

(i) *$\prec$ is proposition-valued;*
(ii) *$\prec$ is transitive;*
(iii) *$\prec$ satisfies nullary interpolation, i.e. for every $x : B$, there exists some $y : B$ with $y \prec x$;*
(iv) *$\prec$ satisfies binary interpolation, i.e. for every $x, y : B$ with $x \prec y$, there exists some $z : B$ with $x \prec z \prec y$.*

▶ **Example 62.** Let D be a $\mathcal{V}$-dcpo with a basis $\beta : B \to D$, By Lemmas 42, 53 and 55, the pair $\left(B, \ll^{B}\right)$ is an example of a $\mathcal{V}$-abstract basis.

▶ **Example 63.** Any preorder $(P, \sqsubseteq)$ with $P : \mathcal{V}$ and $\sqsubseteq$ taking values in $\mathcal{V}$ is a $\mathcal{V}$-abstract basis, since reflexivity implies both interpolation properties.

For the remainder of this section, fix some arbitrary $\mathcal{V}$-abstract basis $(B, \prec)$.

▶ **Definition 64** (Directed subset). *Let A be a $\mathcal{V}$-subset of B. Then A is* directed *if A is inhabited (i.e. $\exists_{x:B}\, x \in A$ holds) and for every $x, y \in A$, there exists some $z \in A$ such that $x, y \sqsubseteq z$.*

▶ **Definition 65** (Ideal, lower set). *Let A be a $\mathcal{V}$-subset of B. Then A is an* ideal *if A is a directed subset of B and A is a* lower set, *i.e. if $x \prec y$ and $y \in A$, then $x \in A$ as well.*

▶ **Construction 66** (Rounded ideal completion $\mathsf{Idl}(B, \prec)$). We construct a $\mathcal{V}$-dcpo, known as the *(rounded) ideal completion* $\mathsf{Idl}(B, \prec) : \mathcal{V}\text{-}\mathsf{DCPO}_{\mathcal{V}^+,\mathcal{V}}$ of $(B, \prec)$. The carrier is given by the type $\sum_{I:B\to\mathcal{V}} \text{is-ideal}(I)$ of ideals on $(B, \prec)$. The order is given by subset inclusion $\subseteq$. If we have a directed family $\alpha : A \to \mathsf{Idl}(B, \prec)$ of ideals (with $A : \mathcal{V}$), then the subset given by $\lambda x.\exists_{a:A} x \in \alpha_a$ is again an ideal and the supremum of α in $\mathsf{Idl}(B, \prec)$. ⌟

▶ **Lemma 67** (Rounded ideals). *The ideals of* $\mathsf{Idl}(B, \prec)$ *are* rounded. *That is, if $I : \mathsf{Idl}(B, \prec)$ and $x \in I$, then there exists some $y \in I$ with $x \prec y$.*

▶ **Definition 68** (Principal ideal $\downarrow x$). *We write $\downarrow(-) : B \to \mathsf{Idl}(B, \prec)$ for the map that takes $x : B$ to the* principal ideal *$\lambda y.y \prec x$.*

▶ **Lemma 69.** *Let $I : \mathsf{Idl}(B, \prec)$ be an ideal. Then I may be expressed as the supremum of the directed family $(x, p) : \mathbb{T}(I) \mapsto \downarrow x : \mathsf{Idl}(B, \prec)$, which we will denote by $I = \bigsqcup_{x \in I} \downarrow x$.*

We wish to prove that $\mathsf{Idl}(B, \prec)$ is continuous with basis $\downarrow(-) : B \to \mathsf{Idl}(B, \prec)$. To this end, it is useful to express $\ll_{\mathsf{Idl}(B,\prec)}$ in more elementary terms.

▶ **Lemma 70.** *Let $I, J : \mathsf{Idl}(B, \prec)$ be two ideals. Then $I \ll J$ holds if and only there exists $x \in J$ such that $I \subseteq {\downarrow} x$.*

▶ **Theorem 71.** *The map $\downarrow(-) : B \to \mathsf{Idl}(B, \prec)$ is a basis for $\mathsf{Idl}(B, \prec)$. Thus, $\mathsf{Idl}(B, \prec)$ is a continuous $\mathcal{V}$-dcpo.*

▶ **Lemma 72.** *If $\prec$ is reflexive, then the compact elements of $\mathsf{Idl}(B, \prec)$ are exactly the principal ideals and $\mathsf{Idl}(B, \prec)$ is algebraic.*

▶ **Theorem 73.** *The ideal completion is the free dcpo on a small poset. That is, if we have a poset $(P, \sqsubseteq)$ with $P : \mathcal{V}$ and $\sqsubseteq$ taking values in $\mathcal{V}$, then for every $D : \mathcal{V}\text{-}\mathsf{DCPO}_{\mathcal{U},\mathcal{T}}$ and monotone function $f : P \to D$, there is a unique continuous function $\overline{f} : \mathsf{Idl}(P, \sqsubseteq) \to D$ such that*

$$\begin{tikzcd}
P \arrow[rr, "f"] \arrow[dr, "\downarrow(-)"'] & & D \\
& \mathsf{Idl}(P, \sqsubseteq) \arrow[ur, dashed, "\overline{f}"'] &
\end{tikzcd}$$

commutes.

▶ **Definition 74** (Continuous retract, section, retraction)**.** *A $\mathcal{V}$-dcpo D is a* continuous retract *of another $\mathcal{V}$-dcpo E if we have continuous functions $s : D \to E$ (the* section*) and $r : E \to D$ (the* retraction*) such that $r(s(x)) = x$ for every $x : D$.*

▶ **Theorem 75.** *If E is a dcpo with basis $\beta : B \to D$ and D is a continuous retract of E with retraction r, then $r \circ \beta$ is a basis for D.*

We now turn to locally small dcpos, as they allow us to find canonical approximating families, which is used in the proof of Theorem 78.

▶ **Lemma 76.** *Let D be a $\mathcal{V}$-dcpo with basis $\beta : B \to D$. The following are equivalent:*
(i) *D is locally small;*
(ii) *$\beta(b) \ll x$ has size $\mathcal{V}$ for every $x : D$ and $b : B$.*

▶ **Lemma 77.** *Let D be a $\mathcal{V}$-dcpo with basis $\beta : B \to D$. If D is locally small, then an element $x : D$ is the supremum of the large directed family $\left(\sum_{b:B} \beta(b) \ll x\right) \xrightarrow{\mathsf{pr}_1} B \xrightarrow{\beta} D$. Moreover, if D is locally small, then this directed family is small.*

▶ **Theorem 78.** *Let D be a $\mathcal{V}$-dcpo with basis $\beta : B \to D$ and suppose that D is locally small. Then D is a continuous retract of the algebraic $\mathcal{V}$-dcpo $\mathsf{Idl}\big(B, \sqsubseteq^B\big)$ (recall Lemma 52).*

One may wonder how restrictive the condition that D is locally small is. We note that if X is a set, then $\mathcal{L}_{\mathcal{V}}(X)$ (by Lemma 18) and $\mathcal{P}_{\mathcal{V}}(X)$ are examples of locally small $\mathcal{V}$-dcpos. A natural question is what happens with exponentials. In general, E^D may fail to be locally small even when both D and E are. However, we do have the following result.

▶ **Lemma 79.** *Let D and E be $\mathcal{V}$-dcpos. Suppose that D is continuous and E is locally small. Then E^D is locally small.*

Moreover, the (co)limit of locally small dcpos is locally small.

▶ **Lemma 80.** *Given a system $(D_i, \varepsilon_{i,j}, \pi_{i,j})$ as in Section 5.1, if every D_i is locally small, then so is D_∞.*

Finally, the requirement that D is locally small is necessary, in the following sense.

▶ **Lemma 81.** *Let D be a $\mathcal{V}$-dcpo with basis $\beta : B \to D$. Suppose that D is a continuous retract of $\mathsf{Idl}\big(B, \sqsubseteq^B\big)$. Then D is locally small.*

We end this section by describing an example of a continuous dcpo, built using the ideal completion, that is not algebraic. In fact, this dcpo has no compact elements at all.

▶ **Example 82** (A continuous dcpo that is not algebraic)**.** We can inductively define a type $\mathbb{D}$ and an order $\prec$ representing dyadic rationals $m/2^n$ in the interval $(-1, 1)$ for integers m, n. Then we prove that $\prec$ is proposition-valued, transitive, irreflexive, trichotomous, dense and that it has no endpoints. Using these properties, we can show that $(\mathbb{D}, \prec)$ is a $\mathcal{U}_0$-abstract basis. Thus, taking the rounded ideal completion, we get $\mathsf{Idl}(\mathbb{D}, \prec) : \mathcal{U}_0\text{-}\mathsf{DCPO}_{\mathcal{U}_1,\mathcal{U}_0}$, which is continuous with basis $\downarrow(-) : \mathbb{D} \to \mathsf{Idl}(\mathbb{D}, \prec)$ by Theorem 71. But $\mathsf{Idl}(\mathbb{D}, \prec)$ cannot be algebraic, since none of its elements are compact. Indeed suppose that we had an ideal I with $I \ll I$. By Lemma 70, there would exist $x \in I$ with $I \sqsubseteq \downarrow x$. But this implies $x \prec x$, but $\prec$ is irreflexive, so this is impossible.

7 Conclusion and Future Work

We have developed domain theory constructively and predicatively in univalent foundations, including Scott's D_∞ model of the untyped λ-calculus, as well as notions of continuous and algebraic dcpos. We avoid size issues in our predicative setting by having large dcpos with joins of small directed families. Often we find it convenient to work with locally small dcpos, whose orders have small truth values.

In future work, we wish to give a predicative account of the theory of algebraic and continuous exponentials, which is a rich and challenging topic even classically. We also intend to develop applications to topology and locale theory. First steps on formal topology and frames in cubical type theory [6, 8] are developed in Tosun's thesis [44], and our notion of continuous dcpo should be applicable to tackle local compactness and exponentiability.

It is also important to understand when classical theorems do not have constructive and predicative counterparts. For instance, Zorn's Lemma doesn't imply excluded middle but it implies propositional resizing [11] and we are working on additional examples.

We have formalized the following in Agda [12], in addition to the Scott model of PCF and its computational adequacy [10, 20]:

1. dcpos,
2. limits and colimits of dcpos, Scott's D_∞,
3. lifting and exponential constructions,
4. pointed dcpos have subsingleton joins (in the right universe),
5. way-below relation, continuous, algebraic dcpos, interpolation properties,
6. abstract bases and rounded ideal completions (including its universal property),
7. continuous dcpos are continuous retract of their ideal completion, and hence of algebraic dcpos,
8. ideal completion of dyadics, giving an example of a non-algebraic, continuous dcpo.

In the near future we intend to complete our formalization to also include Theorems 21, 23 and 25, Examples 59 and 60, and Lemma 79.

References

1 S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.

2 Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories*, volume 189 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1994. doi:10.1017/CBO9780511600579.

3 Roberto M. Amadio and Pierre-Louis Curien. *Domains and Lambda-Calculi*, volume 46 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998. doi:10.1017/CBO9780511983504.

4 Andrej Bauer and Iztok Kavkler. A constructive theory of continuous domains suitable for implementation. *Annals of Pure and Applied Logic*, 159(3):251–267, 2009. doi:10.1016/j.apal.2008.09.025.

5 Nick Benton, Andrew Kennedy, and Carsten Varming. Some domain theory and denotational semantics in Coq. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2009)*, volume 5674 of *Lecture Notes in Computer Science*, pages 115–130. Springer, 2009. doi:10.1007/978-3-642-03359-9_10.

6 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:34. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.TYPES.2015.5.

7 The Agda community. Agda wiki. https://wiki.portal.chalmers.se/agda/pmwiki.php.

8 Thierry Coquand, Simon Huber, and Anders Mörtberg. On Higher Inductive Types in Cubical Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 255–264. Association for Computing Machinery, 2018. doi:10.1145/3209108.3209197.

9 Thierry Coquand, Giovanni Sambin, Jan Smith, and Silvio Valentini. Inductively generated formal topologies. *Annals of Pure and Applied Logic*, 124(1–3):71–106, 2003. doi:10.1016/s0168-0072(03)00052-6.

10 Tom de Jong. The Scott model of PCF in univalent type theory, November 2019. arXiv:1904.09810.

11 Tom de Jong. Domain theory in predicative Univalent Foundations. Abstract for a talk at *HoTT/UF 2020* on 7 July, 2020. URL: https://hott-uf.github.io/2020/HoTTUF_2020_paper_14.pdf.

12 Tom de Jong. Formalisation of domain theory in constructive and predicative univalent foundations. https://github.com/tomdjong/TypeTopology, June 2020. Agda development.

13 Robert Dockins. Formalized, Effective Domain Theory in Coq. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving (ITP 2014)*, volume 8558 of *Lecture Notes in Computer Science*, pages 209–225. Springer, 2014. doi:10.1007/978-3-319-08970-6_14.

14 Martín H. Escardó. The proof of the interpolation property of the way-below relation of a continuous poset. 18 March, 2000. URL: https://www.cs.bham.ac.uk/~mhe/papers/interpolation.pdf.

15 Martín H. Escardó and Cory M. Knapp. Partial Elements and Recursion via Dominances in Univalent Type Theory. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CSL.2017.21.

16 Martín Hötzel Escardó. Introduction to Univalent Foundations of Mathematics with Agda, February 2020. arXiv:1911.00580.

17 Martín Hötzel Escardó. Various new theorems in constructive univalent mathematics written in Agda. https://github.com/martinescardo/TypeTopology, June 2020. Agda development.

18 G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2003. doi:10.1017/CBO9780511542725.

19 Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D. Lawson, Michael W. Mislove, and Dana S. Scott. *A Compendium of Continuous Lattices*. Springer Berlin Heidelberg, 1980. doi:10.1007/978-3-642-67678-9.

20 Brendan Hart. Investigating Properties of PCF in Agda. MSci project, School of Computer Science, University of Birmingham, 2020. Report and Agda code available at https://github.com/BrendanHart/Investigating-Properties-of-PCF.

21 Michael Hedberg. A type-theoretic interpretation of constructive domain theory. *Journal of Automated Reasoning*, 16(3):369–425, 1996. doi:10.1007/BF00252182.

22 J. M. E. Hyland. First steps in synthetic domain theory. In A. Carboni, M. C. Peddicchio, and G. Rosolini, editors, *Category Theory*, volume 1488 of *Lecture Notes in Mathematics*, pages 131–156. Springer, 1991. doi:10.1007/bfb0084217.

23 Peter Johnstone and André Joyal. Continuous categories and exponentiable toposes. *Journal of Pure and Applied Algebra*, 25(3):255–296, 1982. doi:10.1016/0022-4049(82)90083-4.

24 Tatsuji Kawai. Predicative theories of continuous lattices, June 2020. arXiv:2006.05642.

25 Nicolai Kraus, Martín Escardó, Thierry Coquand, and Thorsten Altenkirch. Notions of Anonymous Existence in Martin-Löf Type Theory. *Logical Methods in Computer Science*, 13(1), 2017. doi:10.23638/LMCS-13(1:15)2017.

26 David Lidell. Formalizing domain models of the typed and the untyped lambda calculus in Agda. Master's thesis, Chalmers University of Technology and University of Gothenburg, 2020. Final version in preparation. Agda code available at https://github.com/DoppeD/ConsistentCwfsOfDomains.

27 John Longley and Dag Normann. *Higher-Order Computability*. Springer, 2015. doi:10.1007/978-3-662-47992-6.

28 Maria Emilia Maietti and Silvio Valentini. Exponentiation of Scott formal topologies. In A. Jung M. Escardó, editor, *Proceedings of the Workshop on Domains VI*, volume 73, pages 111–131, 2004. doi:10.1016/j.entcs.2004.08.005.

29 Michael Makkai and Robert Paré. *Accessible categories: the foundations of categorical model theory*, volume 104 of *Contemporary Mathematics*. American Mathematical Society, 1989. doi:10.1090/conm/104.

30 Sara Negri. Continuous lattices in formal topology. In Eduardo Giménez and Christine Paulin-Mohring, editors, *Types for Proofs and Programs (TYPES 1996)*, volume 1512 of *Lecture Notes in Computer Science*, pages 333–353. Springer, 1998. doi:10.1007/BFb0097800.

31 Sara Negri. Continuous domains as formal spaces. *Mathematical Structures in Computer Science*, 12(1):19–52, 2002. doi:10.1017/S0960129501003450.

32 Dirk Pattinson and Mina Mohammadian. Constructive Domains with Classical Witnesses, June 2020. arXiv:1910.04948.

33 G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977. doi:10.1016/0304-3975(77)90044-5.

34 Bernhard Reus. Formalizing synthetic domain theory — the basic definitions. *Journal of Automated Reasoning*, 23(3-4):411–444, 1999. doi:10.1023/A:1006258506401.

35 Bernhard Reus and Thomas Streicher. General synthetic domain theory — a logical approach. *Mathematical Structures in Computer Science*, 9(2):177—-223, 1999. doi:10.1017/S096012959900273X.

36 G. Rosolini. Categories and effective computations. In David H. Pitt, Axel Poigné, and David E. Rydeheard, editors, *Category Theory and Computer Science*, volume 283 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 1987.

37 Giuseppe Rosolini. *Continuity and effectiveness in topoi*. PhD thesis, University of Oxford, 1986.

38 Giovanni Sambin. Intuitionistic formal spaces—a first communication. In *Mathematical logic and its applications*, pages 187–204. Springer, 1987. doi:10.1007/978-1-4613-0897-3_12.

39 Giovanni Sambin, Silvio Valentini, and Paolo Virgili. Constructive domain theory as a branch of intuitionistic pointfree topology. *Theoretical Computer Science*, 159(2):319–341, 1996. doi:10.1016/0304-3975(95)00169-7.

40 Dana S. Scott. Continuous lattices. In F.W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lecture Notes in Mathematics*, pages 97–136. Springer, 1972. doi:10.1007/BFB0073967.

41 Dana S. Scott. Lectures on a mathematical theory of computation. In Manfred Broy and Gunther Schmidt, editors, *Theoretical Foundations of Programming Methodology: Lecture Notes of an International Summer School, directed by F. L. Bauer, E. W. Dijkstra and C. A. R. Hoare*, volume 91 of *NATO Advanced Study Institutes Series*, pages 145–292. Springer, 1982. doi:10.1007/978-94-009-7893-5_9.

42 Dana S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121(1):411–440, 1993. doi:10.1016/0304-3975(93)90095-B.

43 Andrew W. Swan. Choice, collection and covering in cubical sets. Talk in the electronic *HoTTEST* seminar, 6 November. Slides at https://www.uwo.ca/math/faculty/kapulkin/seminars/hottestfiles/Swan-2019-11-06-HoTTEST.pdf. Video recording at https://www.youtube.com/watch?v=r9KbEOzyr1g, 2019.

44 Ayberk Tosun. Formal Topology in Univalent Foundations. Master's thesis, Chalmers University of Technology and University of Gothenburg, 2020. Agda code available at: https://github.com/ayberkt/formal-topology-in-UF. doi:20.500.12380/301098.

45 Taichi Uemura. Cubical Assemblies, a Univalent and Impredicative Universe and a Failure of Propositional Resizing. In Peter Dybjer, José Espírito Santo, and Luís Pinto, editors, *24th International Conference on Types for Proofs and Programs (TYPES 2018)*, volume 130 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:20. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.TYPES.2018.7.

46 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

47 Vladimir Voevodsky. Resizing rules — their use and semantic justification. Slides from a talk at TYPES, Bergen, 11 September, 2011. URL: https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/2011_Bergen.pdf.

48 Vladimir Voevodsky. An experimental library of formalized mathematics based on the univalent foundations. *Mathematical Structures in Computer Science*, 25(5):1278–1294, 2015.

49 Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. UniMath — a computer-checked library of univalent mathematics. Available at https://github.com/UniMath/UniMath.

A Cyclic Proof System for HFL$_\mathbb{N}$

Mayuko Kori
Department of Informatics, The Graduate University for Advanced Studies (SOKENDAI), Hayama, Japan
National Institute of Informatics, Tokyo, Japan
mkori@nii.ac.jp

Takeshi Tsukada
Graduate School of Science, Chiba University, Japan
tsukada@math.s.chiba-u.ac.jp

Naoki Kobayashi
The University of Tokyo, Japan
koba@is.s.u-tokyo.ac.jp

Abstract

A cyclic proof system allows us to perform inductive reasoning without explicit inductions. We propose a cyclic proof system for HFL$_\mathbb{N}$, which is a higher-order predicate logic with natural numbers and alternating fixed-points. Ours is the first cyclic proof system for a higher-order logic, to our knowledge. Due to the presence of higher-order predicates and alternating fixed-points, our cyclic proof system requires a more delicate global condition on cyclic proofs than the original system of Brotherston and Simpson. We prove the decidability of checking the global condition and soundness of this system, and also prove a restricted form of standard completeness for an infinitary variant of our cyclic proof system. A potential application of our cyclic proof system is semi-automated verification of higher-order programs, based on Kobayashi et al.'s recent work on reductions from program verification to HFL$_\mathbb{N}$ validity checking.

2012 ACM Subject Classification Theory of computation → Proof theory

Keywords and phrases Cyclic proof, higher-order logic, fixed-point logic, sequent calculus

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.29

Related Version A full version of the paper is available at `https://arxiv.org/abs/2010.14891`.

Funding This work was supported by JSPS KAKENHI Grant Number JP15H05706, JP20H00577, and JP20H05703.
Mayuko Kori: The first author is supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST.

Acknowledgements We would like to thank anonymous referees for useful comments.

1 Introduction

There have recently been extensive studies on cyclic proof systems. They allow a proof to be cyclic, as long as it satisfies a certain sanity condition called the "global trace condition." Cyclic proofs enable inductive reasoning without explicit inductions, which would be useful for proof automation. Various cyclic proof systems have been proposed [3, 7, 13] for first-order logics, and some of them have been applied to automated program verification [1, 2, 14].

In the present paper, we propose a cyclic proof system for HFL$_\mathbb{N}$, a higher-order logic with natural numbers and least/greatest fixpoint operators on higher-order predicates. HFL$_\mathbb{N}$ has been introduced by Kobayashi et al. [9, 10] as an extension of HFL [16], and shown to be useful for higher-order program verification. Verification of various temporal properties of higher-order programs can naturally be reduced to validity checking for HFL$_\mathbb{N}$ formulas. For example, consider the following OCaml program:

```
let rec g n = if n=0 then () else g (n-1) in
let rec f h m = h m; f h (m+1) in f g 0
```

The property "`f` is infinitely often called" is then expressed as the following formula:

$$\big(\nu F.\lambda h.\lambda m.h\ m \wedge F\ h\ (m+1)\big)\ \big(\mu G.\lambda n.(n = 0 \vee (n \neq 0 \wedge G\ (n-1)))\big)\ 0.$$

Here, $\nu F.\lambda h.\cdots$ and $\mu G.\lambda n.\cdots$ respectively represent the greatest and least predicates such that $F = \lambda h.\cdots$ and $G = \lambda n.\cdots$. Notice the close correspondence between the program and the formula: the functions f and g correspond to the predicates F and G, and function calls correspond to applications of the predicates; an interested reader may wish to consult [9, 10] to learn how program verification problems can be translated to HFL$_\mathbb{N}$ formulas. Our cyclic proof system for HFL$_\mathbb{N}$ presented in this paper would, therefore, be useful for semi-automated verification of higher-order programs.

A key issue in the design of our cyclic proof system is how to formalize a decidable "global trace condition," to guarantee the soundness of cyclic proofs in the presence of higher-order predicates and alternating fixed-points. Our global trace condition has been inspired by, and is actually very similar to that of Doumane [7]. The decidability of our global trace condition is, however, non-trivial, due to the presence of higher-order predicates. Inspired by the approach of Brotherston and Simpson [3], we reduce the global trace condition to the containment problem for Büchi automata.

We also consider an infinitary version of our proof system, and prove that the infinitary proof system is complete for sequents without higher-order variables. The restriction to sequents without higher-order variables is sufficient for the aforementioned application of our proof system to higher-order program verification.

The rest of this paper is structured as follows. Section 2 reviews the syntax and semantics of HFL$_\mathbb{N}$. Section 3 defines our cyclic proof system. Sections 4 and 5 respectively prove the decidability of the global trace condition and the soundness of our cyclic proof system. Section 6 discusses the restricted form of completeness of the infinitary variant of our proof system. Section 7 discusses related work, and Section 8 concludes the paper.

2 HFL$_\mathbb{N}$: Higher-Order Fixed-Point Arithmetic

This section introduces the target logic HFL$_\mathbb{N}$, which is a higher-order logic with natural numbers and alternating fixed-points. It has been introduced by Kobayashi et al. [10, 17] as an extension of higher-order modal fixed-point logic (HFL) [16].

2.1 Syntax of HFL$_\mathbb{N}$

HFL$_\mathbb{N}$ is simply typed. The syntax of *types* is given as follows:

$$A ::= \mathbf{N} \mid T \qquad\qquad T ::= \mathbf{\Omega} \mid A \to T.$$

$\mathbf{N}$ is the type of natural numbers, $\mathbf{\Omega}$ is the type of propositions and $A \to T$ is a function type. Occurrences of $\mathbf{N}$ are restricted to argument positions for a technical reason (see below).

Let $\mathcal{V}$ be a countably infinite set of *variables*, ranged over by $x, y, z, f, X, Y, Z, \ldots$. The syntax of *terms* and *formulas* is given by:

$$\begin{array}{llcl} \text{term} & s, t & ::= & x \mid \mathbf{Z} \mid \mathbf{S}t \\ \text{formula} & \varphi, \psi & ::= & s = t \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid x \mid \lambda x^A.\varphi \mid \varphi\,\psi \mid \varphi\,t \mid \mu x^T.\varphi \mid \nu x^T.\varphi. \end{array}$$

We shall often omit the type annotations. The syntax of terms is standard: $\mathbf{Z}$ represents zero and $\mathbf{S}$ is the successor function. A closed term must be of the form $\mathbf{S}^n\mathbf{Z}$, which is often identified with the natural number n. Constructors of formulas are logical ones ($s = t$, $\varphi \vee \psi$ and $\varphi \wedge \psi$), those from the λ-calculus (variable x, abstraction $\lambda x.\varphi$, and application $\varphi\,\psi$ and $\varphi\,t$), and fixed-point operators (least fixed-point $\mu x.\varphi$ and greatest fixed-point $\nu x.\varphi$). Some standard constructs such as summation $t_1 + t_2 = s$, multiplication $t_1 \times t_2 = s$, truth $\top$ and quantifiers $\forall$ and $\exists$ are definable; see examples later in this subsection. The set of *free variables* is defined as usual; the binders are λx, μx and νx.

A *sequent* is a pair (Γ, Δ) of finite sequences of formulas, written as $\Gamma \vdash \Delta$. For a finite sequence Γ of formulas, $FV(\Gamma)$ denotes the union of the sets of free variables of formulas in Γ. The free variables of a sequent $\Gamma \vdash \Delta$ is $FV(\Gamma) \cup FV(\Delta)$.

The type system is presented in Figure 1. Here $\mathcal{H}$ is a *type environment*, which is a map from a finite subset of variables to the set of types. The typing rules should be easy to understand. The types of fixed-point formulas $\mu x.\varphi$ and $\nu y.\psi$, as well as the types of the variables x and y, are restricted to T (i.e., they cannot be $\mathbf{N}$). A formula φ is *well-typed* if $\mathcal{H} \vdash \varphi : T$ for some $\mathcal{H}$ and T. In the sequel, we shall consider only well-typed formulas.

For a sequent $\Gamma \vdash \Delta$, we write $\mathcal{H} \mid \Gamma \vdash \Delta$ if $\mathcal{H} \vdash \varphi : \mathbf{\Omega}$ for every φ in Γ or Δ. A sequent is *well-typed* if $\mathcal{H} \mid \Gamma \vdash \Delta$ for some $\mathcal{H}$.

We give some examples of formulas and explain their intuitive meaning.

▶ **Example 1.** The truth $\top$ and falsity $\bot$ can be defined as fixed-points:

$$\top \;:=\; \nu x^{\mathbf{\Omega}}.x \qquad \text{and} \qquad \bot \;:=\; \mu x^{\mathbf{\Omega}}.x.$$

The former means that $\top$ is the greatest element in the set of truth values (i.e. elements in the semantic domain of $\mathbf{\Omega}$) such that $x = x$. Similarly $\bot$ is the least truth value. The greatest $\top_T$ and least $\bot_T$ value of type T are defined in a similar way. ⌟

▶ **Example 2.** The quantifiers over natural numbers are definable. Let *forall* be a predicate of type $(\mathbf{N} \to \mathbf{\Omega}) \to \mathbf{N} \to \mathbf{\Omega}$ defined by

$$\mathit{forall} \quad := \quad \nu X.\lambda p^{\mathbf{N}\to\mathbf{\Omega}}.\lambda x^{\mathbf{N}}.p\,x \wedge X\,p\,(\mathbf{S}\,x).$$

Then $\mathit{forall}\,\varphi\,n$ holds if and only if $\varphi\,m$ holds for every $m \geq n$. To see this, notice that $(\mathit{forall}\,\varphi\,n) = (\varphi\,n) \wedge (\mathit{forall}\,\varphi\,(n+1))$ since *forall* is a fixed-point. By iteratively applying this equation, we have

$$(\mathit{forall}\,\varphi\,n) \quad = \quad (\varphi\,n) \wedge (\varphi\,(n+1)) \wedge (\varphi\,(n+2)) \wedge \cdots$$

and thus $\mathit{forall}\,\varphi\,n$ if and only if $\forall m \geq n.\varphi\,m$.[1] Then the universal quantifier over natural numbers is defined by

$$\forall x.\varphi \quad := \quad \mathit{forall}\,(\lambda x.\varphi)\,\mathbf{Z}.$$

The existential quantifier can be defined similarly. A direct definition is

$$\exists x^{\mathbf{N}}.\varphi \quad := \quad \Big(\mu Y^{\mathbf{N}\to\mathbf{\Omega}}.\lambda x.\varphi \vee Y\,(\mathbf{S}x)\Big)\,\mathbf{Z}.$$

The quantifiers over T are easier because of monotonicity (see next subsection); we define $\forall x^T.\varphi := (\lambda x.\varphi)\,\bot_T$ and $\exists x^T.\varphi := (\lambda x.\varphi)\,\top_T$. ⌟

[1] The reader may notice that this intuitive argument does not explain why we should use ν instead of μ. Here we skip this subtle issue. An interested reader may compare the interpretations of *forall* and its μ-variant, following the definition in the next subsection.

For terms:

$$\frac{}{\mathcal{H}, x : A \vdash x : A} \qquad \frac{}{\mathcal{H} \vdash \mathbf{Z} : \mathbf{N}} \qquad \frac{\mathcal{H} \vdash t : \mathbf{N}}{\mathcal{H} \vdash \mathbf{S}t : \mathbf{N}}$$

For formulas:

$$\frac{\mathcal{H} \vdash s : \mathbf{N} \qquad \mathcal{H} \vdash t : \mathbf{N}}{\mathcal{H} \vdash s = t : \mathbf{\Omega}}$$

$$\frac{\mathcal{H} \vdash \varphi : \mathbf{\Omega} \qquad \mathcal{H} \vdash \psi : \mathbf{\Omega}}{\mathcal{H} \vdash \varphi \vee \psi : \mathbf{\Omega}} \qquad \frac{\mathcal{H} \vdash \varphi : \mathbf{\Omega} \qquad \mathcal{H} \vdash \psi : \mathbf{\Omega}}{\mathcal{H} \vdash \varphi \wedge \psi : \mathbf{\Omega}}$$

$$\frac{\mathcal{H}, x : A \vdash \varphi : T}{\mathcal{H} \vdash \lambda x^{A}.\varphi : A \to T} \qquad \frac{\mathcal{H} \vdash \varphi : T' \to T \qquad \mathcal{H} \vdash \psi : T'}{\mathcal{H} \vdash \varphi\, \psi : T}$$

$$\frac{\mathcal{H} \vdash \varphi : \mathbf{N} \to T \qquad \mathcal{H} \vdash t : \mathbf{N}}{\mathcal{H} \vdash \varphi\, t : T}$$

$$\frac{\mathcal{H}, x : T \vdash \varphi : T}{\mathcal{H} \vdash \mu x^{T}.\varphi : T} \qquad \frac{\mathcal{H}, x : T \vdash \varphi : T}{\mathcal{H} \vdash \nu x^{T}.\varphi : T}$$

Figure 1 Typing Rules for HFL$_{\mathbb{N}}$.

▶ **Example 3.** Let *sum* be the summation, i.e. the predicate such that $sum\, n\, m\, k$ holds if and only if $n + m = k$. This predicate can be defined as follows:

$$\mu sum.\, \lambda x^{\mathbf{N}}.\lambda y^{\mathbf{N}}.\lambda z^{\mathbf{N}}.(x = \mathbf{Z} \wedge y = z) \vee (\exists x'.\exists z'.x = \mathbf{S}x' \wedge sum\, x'\, y\, z' \wedge z = \mathbf{S}z').$$

This formula represents the standard inductive definition of the summation: $\mathbf{Z} + y = y$ and $x' + y = z' \implies \mathbf{S}x' + y = \mathbf{S}z'$. One can define the multiplication in a similar way, representing the standard inductive definition of the multiplication using *sum*. ⌟

▶ **Example 4.** The inequality $s < t$ on terms is also definable. The idea is to appeal to the following fact: if $n < m$, then $n + 1 = m$ or $(n + 1) < m$. This justifies

$$(s < t) \quad := \quad (\mu X.\lambda y^{\mathbf{N}}.(\mathbf{S}y = t) \vee X\,(\mathbf{S}y))\, s.$$

Here X is a variable of type $\mathbf{N} \to \mathbf{\Omega}$ and $X\, s'$ means $s' < t$. Then the inequality $s \neq t$ can be defined as $(s < t) \vee (t < s)$. ⌟

▶ Remark 5. HFL$_{\mathbb{N}}$ does not have negation $\neg$. The absence of negation plays an important role in the interpretation of fixed-point operators, as we shall see in the next subsection. For a formula φ with no free variables except for those of type $\mathbf{N}$, the negation $\neg\varphi$ can be obtained by replacing each logical connective with its De Morgan dual,

$$\wedge \leftrightsquigarrow \vee, \qquad \mu \leftrightsquigarrow \nu, \quad \text{and} \quad = \leftrightsquigarrow \neq,$$

as discussed in [12]. ⌟

2.2 Semantics of HFL$_{\mathbb{N}}$

This subsection introduces the interpretations of types and formulas.

The interpretation of a type A is a poset $[\![A]\!] = ([\![A]\!], \leq_A)$. It is inductively defined by

$$\begin{aligned} [\![\mathbf{N}]\!] &:= \mathbb{N} & x \leq_{\mathbf{N}} y &:\Leftrightarrow x = y \\ [\![\mathbf{\Omega}]\!] &:= \{\top, \bot\} & x \leq_{\mathbf{\Omega}} y &:\Leftrightarrow x = \bot \text{ or } y = \top \\ [\![A \to T]\!] &:= \{f : [\![A]\!] \to [\![T]\!] \mid f \text{ is monotone}\} \qquad & f \leq_{A \to T} g &:\Leftrightarrow \forall x \in [\![A]\!].f(x) \leq_T g(x). \end{aligned}$$

$$
\begin{aligned}
[\![\mathcal{H} \vdash x : A]\!](\rho) &= \rho(x) \\
[\![\mathcal{H} \vdash \mathbf{Z} : \mathbf{N}]\!](\rho) &= 0 \\
[\![\mathcal{H} \vdash \mathbf{S}t : \mathbf{N}]\!](\rho) &= 1 + [\![\mathcal{H} \vdash t : \mathbf{N}]\!](\rho) \\
[\![\mathcal{H} \vdash s = t : \mathbf{\Omega}]\!](\rho) &= ([\![\mathcal{H} \vdash s : \mathbf{N}]\!](\rho) = [\![\mathcal{H} \vdash t : \mathbf{N}]\!](\rho)) \\
[\![\mathcal{H} \vdash \varphi \vee \psi : \mathbf{\Omega}]\!](\rho) &= ([\![\mathcal{H} \vdash \varphi : \mathbf{\Omega}]\!](\rho) \vee [\![\mathcal{H} \vdash \psi : \mathbf{\Omega}]\!](\rho)) \\
[\![\mathcal{H} \vdash \varphi \wedge \psi : \mathbf{\Omega}]\!](\rho) &= ([\![\mathcal{H} \vdash \varphi : \mathbf{\Omega}]\!](\rho) \wedge [\![\mathcal{H} \vdash \psi : \mathbf{\Omega}]\!](\rho)) \\
[\![\mathcal{H} \vdash \lambda x^A.\varphi : A \to T]\!](\rho) &= \lambda v \in [\![A]\!].[\![\mathcal{H}, x : A \vdash \varphi : T]\!](\rho[x \mapsto v]) \\
[\![\mathcal{H} \vdash \varphi\,\psi : T]\!](\rho) &= ([\![\mathcal{H} \vdash \varphi : A \to T]\!](\rho))\,([\![\mathcal{H} \vdash \psi : A]\!](\rho)) \\
[\![\mathcal{H} \vdash \varphi\,t : T]\!](\rho) &= ([\![\mathcal{H} \vdash \varphi : \mathbf{N} \to T]\!](\rho))\,([\![\mathcal{H} \vdash t : \mathbf{N}]\!](\rho)) \\
[\![\mathcal{H} \vdash \mu x^T.\varphi : T]\!](\rho) &= \mathrm{lfp}([\![\mathcal{H} \vdash \lambda x^T.\varphi : T \to T]\!](\rho)) \\
[\![\mathcal{H} \vdash \nu x^T.\varphi : T]\!](\rho) &= \mathrm{gfp}([\![\mathcal{H} \vdash \lambda x^T.\varphi : T \to T]\!](\rho))
\end{aligned}
$$

Figure 2 Interpretation of terms and formulas.

Note that

- $[\![A \to T]\!]$ is the space of *monotone* functions, and
- $[\![T]\!]$ is a complete lattice for every T.

On the contrary $[\![A]\!]$ is not necessarily a complete lattice since $[\![\mathbf{N}]\!]$ is not.

The interpretation $[\![\mathcal{H}]\!]$ of a type environment $\mathcal{H}$ is the set of mappings ρ such that $\rho(x) \in [\![\mathcal{H}(x)]\!]$ for every x in the domain of $\mathcal{H}$. The set $[\![\mathcal{H}]\!]$ forms a poset with respect to the point-wise ordering. An element of $[\![\mathcal{H}]\!]$ is called a *valuation*. We write $\rho[x \mapsto v]$ for the mapping defined by $\rho[x \mapsto v](x) = v$ and $\rho[x \mapsto v](y) = \rho(y)$ for $y \neq x$.

Assume $\mathcal{H} \vdash \varphi : A$ (where φ is a formula or a term). Its interpretation $[\![\mathcal{H} \vdash \varphi : A]\!]$ is a monotone function $[\![\mathcal{H}]\!] \longrightarrow [\![A]\!]$. The definition is in Figure 2. Here *lfp*(f) and *gfp*(f) are the least and greatest fixed-points of the function f. The interpretations of the fixed-point operators are well-defined since every monotone function $f : P \longrightarrow P$ on a complete lattice P has both the least and greatest fixed-points. We shall write $[\![\mathcal{H} \vdash \varphi : A]\!](\rho)$ simply $[\![\varphi]\!]_\rho$ when no confusion can arise.

▶ **Definition 6.** *Let $\mathcal{H} \mid \Gamma \vdash \Delta$ be a sequent and ρ be a valuation in $[\![\mathcal{H}]\!]$. Then we write $\Gamma \models_\rho \Delta$ if $\bigwedge_{\varphi \in \Gamma} [\![\varphi]\!]_\rho \leq \bigvee_{\psi \in \Delta} [\![\psi]\!]_\rho$, or equivalently, if $[\![\varphi]\!]_\rho = \bot$ for some $\varphi \in \Gamma$ or $[\![\psi]\!]_\rho = \top$ for some $\psi \in \Delta$. A sequent $\mathcal{H} \mid \Gamma \vdash \Delta$ is* valid *if $\Gamma \models_\rho \Delta$ for all valuations ρ, and we denote it briefly by $\Gamma \models \Delta$.*

3 A Cyclic Proof System for HFL$_{\mathbb{N}}$

In this section, we introduce a cyclic proof system for HFL$_{\mathbb{N}}$. A cyclic proof is a proof diagram which can contain cycles and should satisfy a certain condition in order to ensure the soundness. Subsection 3.1 describes derivations and the soundness condition to define cyclic proofs and Subsection 3.2 shows that the induction rule based on prefixed-points is admissible. In Subsection 3.3, we show some examples of cyclic proofs.

3.1 Definition of the Cyclic Proof System

Figure 3 shows our deduction rules, which are based on Gentzen's sequent calculus. Here $\varphi[\psi/x]$ represents the capture-avoiding substitution and $\Gamma[\varphi/x]$ represents the sequence of formulas which is the result of applying the substitution $[\varphi/x]$ to all formulas in Γ. The rules should be easy to understand since most of the rules are standard. We explain uncommon rules. The rule (Mono) is based on the fact that each formula defines a monotone function, i.e. $[\![\psi]\!] \sqsubseteq [\![\chi]\!]$ implies $[\![\varphi[\psi/x]]\!] \sqsubseteq [\![\varphi[\chi/x]]\!]$. Since only a formula of type $\mathbf{\Omega}$ can appear in a sequent, $[\![\psi]\!] \sqsubseteq [\![\chi]\!]$ is expressed as $\psi\,\vec{y} \vdash \chi\,\vec{y}$ for fresh $\vec{y}$. The rules (λL) and (λR) are justified by the fact that the β-equivalence $(\lambda x.\varphi)\,\psi = \varphi[\psi/x]$ preserves semantics. The rules (σL) and (σR) express the fact that $\sigma x.\varphi$ (where σ is μ or ν) is a fixed-point and thus $\sigma x.\varphi = \varphi[\sigma x.\varphi/x]$. The rule (Nat) says that a variable of type $\mathbf{N}$ indeed represents a natural number, and (P1) and (P2) correspond to the axioms $(\mathbf{Z} = \mathbf{S}x) \to \bot$ and $(\mathbf{S}x = \mathbf{S}y) \to (x = y)$.

- Identity rules

$$\frac{}{\varphi \vdash \varphi}\ \text{(Axiom)} \qquad \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma, \varphi \vdash \Delta}{\Gamma \vdash \Delta}\ \text{(Cut)}$$

- Structural rules

$$\frac{\Gamma \vdash \Delta}{\Gamma, \varphi \vdash \Delta}\ \text{(Wk L)} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \varphi, \Delta}\ \text{(Wk R)}$$

$$\frac{\Gamma, \varphi, \varphi \vdash \Delta}{\Gamma, \varphi \vdash \Delta}\ \text{(Ctr L)} \qquad \frac{\Gamma \vdash \varphi, \varphi, \Delta}{\Gamma \vdash \varphi, \Delta}\ \text{(Ctr R)}$$

$$\frac{\Gamma, \psi, \varphi, \Gamma' \vdash \Delta}{\Gamma, \varphi, \psi, \Gamma' \vdash \Delta}\ \text{(Ex L)} \qquad \frac{\Gamma \vdash \Delta, \psi, \varphi, \Delta'}{\Gamma \vdash \Delta, \varphi, \psi, \Delta'}\ \text{(Ex R)}$$

$$\frac{\Gamma \vdash \Delta}{\Gamma[\varphi/x] \vdash \Delta[\varphi/x]}\ \text{(Subst)}$$

$$\frac{\Gamma, \psi\,\vec{y} \vdash \chi\,\vec{y}, \Delta}{\Gamma, \varphi[\psi/x^T] \vdash \varphi[\chi/x^T], \Delta}\ \vec{y} \cap FV(\Gamma, \psi, \chi, \Delta) = \emptyset,\ \text{(Mono)}$$

- Logical rules $(\sigma = \mu, \nu)$

$$\frac{\Gamma[t/x, s/y] \vdash \Delta[t/x, s/y]}{\Gamma[s/x, t/y], s = t \vdash \Delta[s/x, t/y]}\ (= L) \qquad \frac{}{\Gamma \vdash t = t, \Delta}\ (= R)$$

$$\frac{\Gamma, \varphi \vdash \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta}\ (\vee L) \qquad \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta}\ (\vee R)$$

$$\frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta}\ (\wedge L) \qquad \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta}\ (\wedge R)$$

$$\frac{\Gamma, \varphi[\psi/x]\,\vec{\psi} \vdash \Delta}{\Gamma, (\lambda x.\varphi)\,\psi\,\vec{\psi} \vdash \Delta}\ (\lambda L) \qquad \frac{\Gamma \vdash \varphi[\psi/x]\,\vec{\psi}, \Delta}{\Gamma \vdash (\lambda x.\varphi)\,\psi\,\vec{\psi}, \Delta}\ (\lambda R)$$

$$\frac{\Gamma, \varphi[\sigma x.\varphi/x]\,\vec{\psi} \vdash \Delta}{\Gamma, (\sigma x.\varphi)\,\vec{\psi} \vdash \Delta}\ (\sigma L) \qquad \frac{\Gamma \vdash \varphi[\sigma x.\varphi/x]\,\vec{\psi}, \Delta}{\Gamma \vdash (\sigma x.\varphi)\,\vec{\psi}, \Delta}\ (\sigma R)$$

- Natural number rules
  $N \equiv \mu X.\lambda x.(x = \mathbf{Z}) \vee (\exists x'.x = \mathbf{S}x' \wedge X\,x')$

$$\frac{\Gamma, N\,x^{\mathbf{N}} \vdash \Delta}{\Gamma \vdash \Delta}\ \text{(Nat)} \qquad \frac{}{\mathbf{S}s = \mathbf{Z} \vdash}\ \text{(P1)} \qquad \frac{\Gamma, s = t \vdash \Delta}{\Gamma, \mathbf{S}s = \mathbf{S}t \vdash \Delta}\ \text{(P2)}$$

Figure 3 Deduction Rules.

Although every leaf of an ordinary proof tree is an axiom, this is not the case in cyclic proof systems. A *(finite) derivation tree* is a tree obtained by using the rules in Figure 3, whose leaves are not necessarily axioms. A leaf that is not an axiom is called *open*.

▶ **Definition 7** (Pre-proof). *A* pre-proof *consists of a finite derivation tree $\mathcal{D}$ and a function $\mathcal{R}$ that assigns to each open leaf n in $\mathcal{D}$ a non-leaf node $\mathcal{R}(n)$ that has the same sequent as n.*

A pre-proof induces the infinite derivation tree by iteratively replacing an open leaf n with $\mathcal{R}(n)$. This correspondence would be helpful to understand the definitions below.

A pre-proof is unsound in general, i.e., the root sequent of a pre-proof may be invalid. We introduce a sanity condition called the *global trace condition*, and define cyclic proofs as pre-proofs that satisfy this condition.

Given a pre-proof $(\mathcal{D}, \mathcal{R})$, its *path* is a (finite or infinite) sequence $(n_i)_{i=1,2,\dots}$ of nodes of $\mathcal{D}$ such that, for every i,

- if n_i is an open leaf, then $n_{i+1} = \mathcal{R}(n_i)$, and
- otherwise n_{i+1} is a premise of n_i.

Given a path $(n_i)_{i=1,2,\dots}$, a *pre-trace* in this path is a sequence $(\tau_i)_{i=1,2,\dots}$ of occurrences of formulas such that, for every i, τ_i is an occurrence of a formula in the sequent n_i and τ_{i+1} is a "relevant occurrence" of τ_i in the sequent n_{i+1}. The latter condition means that τ_{i+1} originates from τ_i in a bottom-up construction of the proof. For example, if n_i and n_{i+1} are respectively the conclusion and premise of $(\wedge L)$, i.e., if n_i is the sequent $\Gamma, \varphi \wedge \psi \vdash \Delta$ and n_{i+1} is $\Gamma, \varphi, \psi \vdash \Delta$, then (i) φ and ψ in n_{i+1} are relevant occurrences of $\varphi \wedge \psi$ in n_i, and (ii) each formula in Γ (resp. Δ) of n_{i+1} is a relevant occurrence of the corresponding formula in Γ (resp. Δ) of n_i. For another example, if n_i is the conclusion of $(\vee L)$, n_{i+1} is the left premise and τ_i is the occurrence of $\varphi \vee \psi$, then τ_{i+1} is the occurrence of φ. The concrete definition, which we omit here, is lengthy but straightforward; a possible exception is (Mono), in which $\psi\,\vec{y}$ and $\chi\,\vec{y}$ are defined as relevant occurrences of $\varphi[\psi/x]$ and $\varphi[\chi/x]$ respectively. See Appendix A for more detail. A pre-trace is a *trace* if, for infinitely many i, τ_i is the principal occurrence[2] of a logical rule.

The global trace condition requires existence of a "good" trace for each infinite path. The appropriate notion of "good" traces depends on the logic. In [3], a trace is "good" if it contains infinitely many principal occurrences of (μL) or (νR). In other words, a "good" trace contains infinitely many expansions of μ. This fairly simple condition comes from the restriction of usage of fixed-points: their logic does not allow alternation of fixed-points, e.g. $\mu P.\varphi$ is allowed only if the free predicate variables of φ are bound by μ. Allowing nested fixed-points makes the definition of "good" traces more complicated; the definition in [7] refers to the most significant fixed-point operator among those that are expanded infinitely many times. The higher-order nature of $\mathrm{HFL}_{\mathbb{N}}$ requires us to more precisely track the usage of fixed-point operators.

The following definition is inspired by the winning criterion of game semantics of $\mathrm{HFL}_{\mathbb{N}}$ [4, 10, 15]. The idea is to track which occurences of fixed-point operators are unfolded infinitely in depth by annotating each occurrence with a sequence that grows with each unfolding. By abuse of notation, a path is written as a sequence $(\Gamma_i \vdash \Delta_i)_{i=1,2,\dots}$ of sequents. We often identify an occurrence of a formula with the formula. For example, $\tau_i \equiv \varphi$ means that τ_i is an occurrence of φ.

[2] An occurrence of a formula in the conclusion of a rule in Figure 3 is *principal* if it belongs to neither Γ nor Δ.

▶ **Definition 8** (μ-trace, ν-trace)**.** *Let $(\tau_i)_{i\geq 0}$ be a trace. We assign a sequence of natural numbers to every fixed-point operator in τ_i for all i by the following algorithm. We use σ as a metavariable of fixed-point operators $\{\mu, \nu\}$. We write σ_p for the fixed-point operator to which the sequence p is assigned.*

- *For every σ in τ_0, we assign ϵ to σ.*
- *If τ_i is the principal occurrence of $(\sigma L/R)$, then τ_i with annotation is $\sigma_p x.\varphi$ (where fixed-point operators in φ are also annotated). Then τ_{i+1} with annotation is $\varphi[\sigma_{p.k}x.\varphi/x]$ where k is a natural number that has not been used in this annotation process.*[3]
- *Otherwise, the sequence of a fixed-point operator in τ_{i+1} comes from the corresponding operator in τ_i. For example, if τ_i is the principal occurrence of $(\lambda L/R)$ and τ_i with annotation is $(\lambda x.\varphi)\,\psi$, then τ_{i+1} with annotation is $\varphi[\psi/x]$.*

Let $p[0:n]$ denote the sequence consisting of the first n elements of p. We call $(\tau_i)_{i\geq 0}$ a μ-trace (resp. ν-trace) *if there is an infinite sequence p such that $p[0:n]$ is assigned to μ (resp. ν) in some τ_i for every natural number n.*

In the definiton above, for each trace $(\tau_i)_{i\geq 0}$, there is at most one infinite trace p that satisfies the condition above; hence, no trace can be both a μ-trace and a ν-trace; see Lemma 22.

▶ **Example 9.** Let us consider the following pre-proof $(\mathcal{D}, \mathcal{R})$.

$$\dfrac{\dfrac{\dfrac{\dfrac{(\star)\ \vdash (\nu f.\lambda g.g\,(f\,g))\,(\mu x.\lambda a.a)}{\vdash (\lambda a.a)\,((\nu f.\lambda g.g\,(f\,g))\,(\mu x.\lambda a.a))}\,(\lambda R)}{\vdash (\mu x.\lambda a.a)\,((\nu f.\lambda g.g\,(f\,g))\,(\mu x.\lambda a.a))}\,(\mu R)}{\vdash (\lambda h.h\,((\nu f.\lambda g.g\,(f\,g))\,h))\,(\mu x.\lambda a.a)}\,(\lambda R)}{(\star)\ \vdash (\nu f.\lambda g.g\,(f\,g))\,(\mu x.\lambda a.a)}\,(\nu R)$$

In the diagram, the function $\mathcal{R}$ is indicated by the $\star$ marks: the open leaf $(\star)$ is mapped to the other node marked $(\star)$. This pre-proof has a unique path, and the path has a unique trace $(\tau_i)_{i\geq 0}$ (since each sequent consists of a single formula).

We assign a sequence of natural numbers to each occurrence of a fixed-point operator in the trace $(\tau_i)_{i\geq 0}$. Both fixed-point operators in τ_0 are annotated by the empty sequence:

$$\tau_0 \quad \equiv \quad (\nu_\epsilon f.\lambda g.g\,(f\,g))\,(\mu_\epsilon x.\lambda a.a).$$

The first rule expands ν, and we annotate the recursive occurrence of this ν with a fresh natural number, say 0:

$$\tau_1 \quad \equiv \quad (\lambda h.h\,((\nu_0 f.\lambda g.g\,(f\,g))\,h))\,(\mu_\epsilon x.\lambda a.a).$$

The next rule is (λR) and we just substitute the annotated formula $(\mu_\epsilon x.\lambda a.a)$ for h:

$$\tau_2 \quad \equiv \quad (\mu_\epsilon x.\lambda a.a)\,((\nu_0 f.\lambda g.g\,(f\,g))\,(\mu_\epsilon x.\lambda a.a)).$$

Then we expand μ and annotate its recursive occurrences (if there were any) with 1; actually, since x does not appear in the body $\lambda a.a$, the resulting formula does not have label 1.

$$\tau_3 \quad \equiv \quad (\lambda a.a)\,((\nu_0 f.\lambda g.g\,(f\,g))\,(\mu_\epsilon x.\lambda a.a)).$$

Applying the β-reduction, we have

$$\tau_4 \quad \equiv \quad (\nu_0 f.\lambda g.g\,(f\,g))\,(\mu_\epsilon x.\lambda a.a).$$

[3] One can weaken the freshness requirement for k: the minimal requirement is that the sequence $p.k$ has not been used.

The current node is the open leaf, and the next node is determined by $\mathcal{R}$. The annotation is copied: $\tau_5 \equiv \tau_4$. The next rule is (νR) and we name the recursive occurrences 0.2, extending the annotation 0 by a fresh number 2:

$$\tau_6 \quad \equiv \quad (\lambda h.h\,((\nu_{0.2} f.\lambda g.g\,(f\,g))\,h))\,(\mu_\epsilon x.\lambda a.a).$$

By continuing this argument, we have

$$\tau_{1+5k} \quad \equiv \quad (\lambda h.h\,((\nu_p f.\lambda g.g\,(f\,g))\,h))\,(\mu_\epsilon x.\lambda a.a)$$

where $p = 0.2.4.\ldots.(2k)$. Note that the annotation of ν grows but that of μ does not. Hence this trace is a ν-trace but not a μ-trace. ⌟

A trace $(\tau_i)_{i\geq 0}$ is a *left trace* (resp. *right trace*) if τ_0 occurs on the left (resp. right) side of $\vdash$. Note that every τ_i occurs on the same side as τ_0.

▶ **Definition 10** (Cyclic proof). *A* cyclic proof *is a pre-proof that satisfies the* global trace condition*: for every infinite path, a tail of the path has a left μ-trace or right ν-trace.*

▶ **Example 11.** The pre-proof in Example 9 is a cyclic proof. ⌟

▶ Remark 12. One may find our global trace condition (cf. Definitions 8 and 10) complicated and wonder if it is possible to replace it with a simpler conidition such as the parity condition. A recent result [15, Theorem 25] suggests a negative answer: It shows that the validity of $\mathrm{HFL}_{\mathbb{N}}$ formulas cannot be captured by parity games, but games with more complicated winning criteria. Our global trace condition is inspired by the criteria. ⌟

3.2 Some Admissible Rules

This subsection shows that some familiar rules for quantifiers and inductions are admissible in our cyclic proof system.

As we saw in Subsection 2.1, formulas with quantifiers can be expressed by fixed-points. The proposition below enables us to use quantifier rules in our cyclic proof system.

▶ **Proposition 13.** *If there is a cyclic proof of $\Gamma \vdash \Delta$ derived by the rules in Figure 3 plus the following quantifier rules, then there exists a cyclic proof of $\Gamma \vdash \Delta$ without the quantifier rules.*

$$\frac{\Gamma, \varphi[\psi/x] \vdash \Delta}{\Gamma, \forall x.\varphi \vdash \Delta}\ (\forall L) \qquad\qquad \frac{\Gamma \vdash \varphi, \Delta}{\Gamma \vdash \forall x.\varphi, \Delta}\ x \notin FV(\Gamma, \Delta)\ (\forall R)$$

$$\frac{\Gamma, \varphi \vdash \Delta}{\Gamma, \exists x.\varphi \vdash \Delta}\ x \notin FV(\Gamma, \Delta)\ (\exists L) \qquad\qquad \frac{\Gamma \vdash \varphi[\psi/x], \Delta}{\Gamma \vdash \exists x.\varphi, \Delta}\ (\exists R)$$

Proof. See [11]. ◀

We can also embed explicit induction rules, so-called Park's fixed-point rules:

$$\frac{\Gamma, \varphi[\chi/x]\,\vec{y} \vdash \chi\,\vec{y}, \Delta \qquad \Gamma, \chi\,\vec{\psi} \vdash \Delta}{\Gamma, (\mu x.\varphi)\,\vec{\psi} \vdash \Delta}\ \vec{y} \cap FV(\Gamma, \varphi[\chi/x], \Delta) = \emptyset\ \text{(Pre)}$$

$$\frac{\Gamma, \chi\,\vec{y} \vdash \varphi[\chi/x]\,\vec{y}, \Delta \qquad \Gamma \vdash \chi\,\vec{\psi}, \Delta}{\Gamma \vdash (\nu x.\varphi)\,\vec{\psi}, \Delta}\ \vec{y} \cap FV(\Gamma, \varphi[\chi/x], \Delta) = \emptyset\ \text{(Post)}$$

They are inspired by Knaster-Tarski's fixed-point theorem: these rules replace pre/postfixed-points with least/greatest fixed-points. The rule (Pre) is sound because χ is a prefixed-point by the left premise and $\mu x.\varphi$ is the least one. The same argument holds for (Post).

▶ **Proposition 14.** *If there is a cyclic proof of* $\Gamma \vdash \Delta$ *derived by the rules in Figure 3 + (Pre) + (Post), then there exists a cyclic proof of* $\Gamma \vdash \Delta$ *without (Pre) and (Post).*

Proof. Given a cyclic proof Π that may use (Pre) and (Post), we first construct a pre-proof Π' by removing (Pre) and (Post). For every instance of (Pre) of the form:

$$\dfrac{\Gamma, \varphi[\chi/x]\,\vec{y} \vdash \chi\,\vec{y}, \Delta \qquad \Gamma, \chi\,\vec{\psi} \vdash \Delta}{\Gamma, (\mu x.\varphi)\,\vec{\psi} \vdash \Delta}\ (\text{Pre})$$

we will replace it with the following diagram.

$$\dfrac{\Pi \qquad \dfrac{\Gamma, \chi\,\vec{\psi} \vdash \Delta}{\Gamma, (\mu x.\varphi)\,\vec{\psi}, \chi\,\vec{\psi} \vdash \Delta}\ (\text{Wk L})}{\Gamma, (\mu x.\varphi)\,\vec{\psi} \vdash \Delta}\ (\text{Cut})$$

$$\Pi := \dfrac{\dfrac{\dfrac{\Gamma, \varphi[\chi/x]\,\vec{y} \vdash \chi\,\vec{y}, \Delta}{\Gamma, (\mu x.\varphi)\,\vec{y}, \varphi[\chi/x]\,\vec{y} \vdash \chi\,\vec{y}, \Delta}\ (\text{Wk L}) \qquad \dfrac{\dfrac{\dfrac{(\star)\ \Gamma, (\mu x.\varphi)\,\vec{y} \vdash \chi\,\vec{y}, \Delta}{\Gamma, \varphi[\mu x.\varphi/x]\,\vec{y} \vdash \varphi[\chi/x]\,\vec{y}, \Delta}\ (\text{Mono})}{\Gamma, \varphi[\mu x.\varphi/x]\,\vec{y} \vdash \chi\,\vec{y}, \varphi[\chi/x]\,\vec{y}, \Delta}\ (\text{Wk R})}{\Gamma, (\mu x.\varphi)\,\vec{y} \vdash \chi\,\vec{y}, \varphi[\chi/x]\,\vec{y}, \Delta}\ (\mu\text{L})}{(\star)\ \Gamma, (\mu x.\varphi)\,\vec{y} \vdash \chi\,\vec{y}, \Delta}\ (\text{Cut})}{\Gamma, (\mu x.\varphi)\,\vec{\psi} \vdash \chi\,\vec{\psi}, \Delta}\ (\text{Subst})$$

(Post) can also be removed in the same manner.

We show that the resulting pre-proof Π' is a cyclic proof. For each infinite path π in Π', if some tail of π goes through only one cycle as the above one from $(\star)$ to $(\star)$ then we can trace the left μ in $\mu x.\varphi$ or the right ν in $\nu x.\varphi$. Otherwise, there exists a corresponding infinite path in Π, which satisfies the global trace condition. Therefore Π' is a cyclic proof of $\Gamma \vdash \Delta$. ◀

3.3 Examples

This subsection presents two examples of cyclic proofs.

▶ **Example 15** (Well-foundedness of a tree). In this example, we write $\mathbf{N}^*$ for the type of finite sequences of natural numbers. We also use the concatenation operation $(-) \cdot (-)$. This $\mathbf{N}^*$ can be expressed by $\mathbf{N}$ and thus this additional type does not increase the expressivity.

A *tree* is a subset of finite sequences of natural numbers that represents the complement of the tree; the idea is to regard ϵ as the root and p as the parent of $p \cdot i$. Let $f^{\mathbf{N}^* \to \mathbf{\Omega}}$ be a term representing a tree. We define Φ and Ψ as follows:

$$\Phi := \mu w^{(\mathbf{N}^* \to \mathbf{\Omega}) \to \mathbf{\Omega}}.\lambda k^{\mathbf{N}^* \to \mathbf{\Omega}}.k\,\epsilon \vee \forall i^{\mathbf{N}}.w\,(\lambda z^{\mathbf{N}^*}.k\,(i \cdot z))$$
$$\Psi := \mu v^{\mathbf{N}^* \to \mathbf{\Omega}}.\lambda z^{\mathbf{N}^*}.f\,z \vee \forall i^{\mathbf{N}}.v\,(z \cdot i)$$

Then both $\Phi\,f$ and $\Psi\,\epsilon$ represent well-foundedness of f, i.e. whether there is no infinite path in f. Φ checks whether the tree k satisfies well-foundedness. This returns true if k has only one node or all the immediate children of the root satisfy well-foundedness. Ψ checks whether the subtree of f whose root node is z satisfies well-foundedness. This returns true if z is a leaf or all subtrees under z satisfy well-foundedness.

A cyclic proof of $\Phi\,f \vdash \Psi\,\epsilon$ is given as follows:

$$\dfrac{\dfrac{}{f\,l \vdash f\,l}\,(\text{Axiom}) \qquad \dfrac{\dfrac{\dfrac{(\dagger)\quad \Phi\,(\lambda z.f\,(l\cdot n\cdot z)) \vdash \Psi\,(l\cdot n) \qquad (\star)\quad Y_\Phi\,(\mathbf{S}n) \vdash Y_\Psi\,(\mathbf{S}n)}{\Phi\,(\lambda z.f\,(l\cdot n\cdot z)) \wedge Y_\Phi\,(\mathbf{S}n) \vdash \Psi\,(l\cdot n) \wedge Y_\Psi\,(\mathbf{S}n)}\,(\wedge L, R)}{(\star)\quad Y_\Phi\,n \vdash Y_\Psi\,n}\,(\nu L, R)}{Y_\Phi\,\mathbf{Z} \vdash Y_\Psi\,\mathbf{Z}}\,(\text{Subst})}{\dfrac{f\,l \vee \forall i.\Phi\,(\lambda z.f\,(l\cdot i\cdot z)) \vdash f\,l \vee \forall i.\Psi\,(l\cdot i)}{\dfrac{(\dagger)\quad \Phi\,(\lambda z.f\,(l\cdot z)) \vdash \Psi\,l}{\Phi\,f \vdash \Psi\,\epsilon}\,(\text{Subst}) \text{ and } f \equiv \lambda z.f\,(\epsilon\cdot z)}\,(\mu L, R)}\,(\vee L, \vee R, Wk)$$

where $Y_\Phi \equiv (\nu Y.\lambda i.\Phi\,(\lambda z.f\,(l\cdot i\cdot z)) \wedge Y\,(\mathbf{S}i))$ and $Y_\Psi \equiv (\nu Y.\lambda i.\Psi\,(l\cdot i) \wedge Y\,(\mathbf{S}i))$. Here we omit (Subst) for open leaves; hence cycles of $(\star)$ and $(\dagger)$ are valid, although two nodes for each label have different sequents. Note that $Y_\Phi\,\mathbf{Z} \equiv \forall i.\Phi\,(\lambda z.f\,(l\cdot i\cdot z))$ and $Y_\Psi \equiv \forall i.\Psi\,(l\cdot i)$.

For all infinite paths, if the path includes $(\dagger) \to (\dagger)$ infinitely then we can trace the left μ in Φ and otherwise we can trace the right ν in $\forall i.\Psi\,(l\cdot i)$. ⌟

The example below demonstrates an application of our cyclic proof system to program verification, based on the reduction of Kobayashi et al. [10, 17] from program verification to $\text{HFL}_\mathbb{N}$ validity checking.

▶ **Example 16** (Example 2.4 and 3.3 in [17]). Consider the following OCaml-like program.

```
let rec repeat f x =
  if x = 0 then () else if * then repeat f (f x) else repeat f (x-1)
in let y = input() in
  repeat (fun x -> x-y) n
```

In this program, * represents a non-deterministic Boolean value and `input()` means a user input. We aim to verify that an appropriate input `y` makes this program eventually terminate.

To verify this, we have to check $\vdash \text{input}\,0\,(g\,n)$ where

$$\begin{aligned} \text{repeat} &:= \mu R.\lambda f.\lambda x.(x = 0) \vee (\exists x'.x = x' + 1 \wedge f\,x\,(R\,f) \wedge R\,f\,x') \\ \text{sub} &:= \lambda y.\lambda x.\lambda k.k\,(x - y) \\ g &:= \lambda z.\lambda y.\text{repeat}\,(\text{sub}\,y)\,z \\ \text{input} &:= \mu I.\lambda x.\lambda k.(k\,x) \vee (I\,(x + 1)\,k) \end{aligned}$$

where $(-)$ is defined naturally by using μ. Here, functions on integers of type $\mathbf{N} \to \mathbf{N}$ in the program have been turned into predicates of type $\mathbf{N} \to (\mathbf{N} \to \mathbf{\Omega}) \to \mathbf{\Omega}$, which are obtained by CPS translation; for example, the function `fun x->x-y` has been turned into $\text{sub}\,y$ ($\equiv \lambda x.\lambda k.k(x - y)$). Note that $\text{input}\,0\,(g\,n)$ is equivalent to $\exists y.g\,n\,y$, which models an angelic non-determinsm of `input()` in the program.

The goal sequent can be proved by the following cyclic proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 = 0}\,(\text{=R})}{\vdash \text{repeat}\,(\text{sub}\,1)\,0}\,(\mu\text{R}, \vee\text{R})}{n = 0 \vdash \text{repeat}\,(\text{sub}\,1)\,n}\,(\text{=L}) \qquad \Pi}{(\star, \dagger)\ \mathbf{N}\,n \vdash \text{repeat}\,(\text{sub}\,1)\,n}\,(\mu\text{L})}{\vdash g\,n\,1}\,(\lambda\text{R, Nat})}{\vdash g\,n\,1, \text{input}\,2\,(g\,n)}\,(\text{Wk R})}{\vdash \text{input}\,1\,(g\,n)}\,(\mu\text{R}, \vee\text{R})}{\vdash g\,n\,0, \text{input}\,1\,(g\,n)}\,(\text{Wk R})}{\vdash \text{input}\,0\,(g\,n)}\,(\mu\text{R}, \vee\text{R})$$

$$\Pi := \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\star)\ \mathbf{N}\,n' \vdash \mathrm{repeat}\,(\mathrm{sub}\,1)\,n'}{\mathbf{N}\,n' \vdash (\mathrm{sub}\,1)\,(n'+1)\,(\mathrm{repeat}\,(\mathrm{sub}\,1))}\,(\lambda\mathrm{R}) \qquad (\dagger)\ \mathbf{N}\,n' \vdash \mathrm{repeat}\,(\mathrm{sub}\,1)\,n'}{\mathbf{N}\,n' \vdash (\mathrm{sub}\,1)\,(n'+1)\,(\mathrm{repeat}\,(\mathrm{sub}\,1)) \land \mathrm{repeat}\,(\mathrm{sub}\,1)\,n'}\,(\land\mathrm{R})}{\mathbf{N}\,n' \vdash \mathrm{repeat}\,(\mathrm{sub}\,1)\,(n'+1)}\,(\mu\mathrm{R})}{\mathbf{N}\,n', n = n'+1 \vdash \mathrm{repeat}\,(\mathrm{sub}\,1)\,n}\,(=\mathrm{L})}{\exists n'.\mathbf{N}\,n' \land n = n'+1 \vdash \mathrm{repeat}\,(\mathrm{sub}\,1)\,n}\,(\exists\mathrm{L},\land\mathrm{L})$$

This satisfies the global trace condition because we can trace the left μ in $\mathbf{N}$ for every infinite path. ⌟

4 Decidability of the Global Trace Condition

For our cyclic proof system to be useful, there should exist an algorithm to check whether a given proof candidate is indeed a cyclic proof. It is easy to check whether a given candidate is a pre-proof, but it is non-trivial to check whether the pre-proof also satisfies the global trace condition. In this section we prove that the global condition is indeed decidable. We follow the approach in [3], which reduces the global trace condition of a pre-proof to Büchi automata containment. One automaton accepts all infinite paths of the pre-proof and the other accepts those that satisfy the global trace condition. Then whether the pre-proof is a cyclic proof corresponds to the inclusion between these automata.

Let us briefly recall the definition of Büchi automata. A *(nondeterministic) Büchi automaton* is a tuple $(Q, \Sigma, \delta, Q_0, F)$ where Q is a finite set of elements called *states*, Σ is a finite set of symbols, $\delta : Q \times \Sigma \times Q$ is a *transition relation*, $Q_0 \subseteq Q$ is a set of *initial states*, and $F \subseteq \delta$ is a set of accepting transition rules.[4] Given an infinite word $w = (a_i)_{i \geq 0} \in \Sigma^\omega$, a *run* over w is a sequence $(q_i)_{i\geq 0}$ of states such that $q_0 \in Q_0$ and $(q_i, a_i, q_{i+1}) \in \delta$ for all $i \geq 0$. This run is *accepting* if $(q_i, a_i, q_{i+1}) \in F$ for infinitely many i. The automaton *accepts* an infinite word if there is an accepting run over the word.

Let $(\mathcal{D}, \mathcal{R})$ be a given pre-proof. The alphabet Σ is the set of nodes of $\mathcal{D}$. Then an infinite path is represented as an infinite word, and it is easy to construct an automaton $\mathcal{A}_{path}$ that accepts all the paths of $\mathcal{D}$. We define an automaton $\mathcal{A}_{gtc}$ that checks if a given path satisfies the global trace condition.

The idea of the automaton is as follows. Recall Definition 8, in which we assign a sequence of natural numbers to each occurrence of a fixed-point operator. One cannot directly simulate the annotation process by an automaton since the set of sequences of natural numbers is infinite. However, at the end of Definition 8, we focus on an infinite sequence p of natural numbers, and sequences that are not prefixes of p can be safely ignored. Furthermore, it suffices to remember exactly one finite sequence by the following argument. Consider the case that $\tau_i \equiv \sigma_q x.\varphi$ where q is a prefix of p, and $\tau_{i+1} \equiv \varphi[\sigma_{q.k} x.\varphi/x]$.

- If $q.k$ is not a prefix of p, then we can safely forget the annotation $q.k$.
- If $q.k$ is a prefix of p, then we can safely forget the annotation q.

To see the latter, observe that other expansions of σ_q generate annotations $q.k'$ with $k' \neq k$ by freshness of k'. Hence $q.k'$ is not a prefix of p and thus we can forget it. So any extension of q that will be generated afterward can be safely ignored, and thus we can forget q itself.

The above argument motivates the following definition.

[4] This differs from the standard definition, in which the acceptance condition is specified by the set of accepting *states*. It is not difficult to see that this change does not affect the expressive power.

▶ **Definition 17.** *A* marked formula $\check{\varphi}$ *is a formula in which some occurrences of fixed-point operators* σ *are* marked*; a marked fixed-point operator is written as* $\sigma_\bullet$*. We write* $|\check{\varphi}|$ *for the (standard) formula obtained by removing marks. An* occurrence-with-marks $\check{\tau}$ *is a pair* $(\tau, \check{\varphi})$ *of an occurrence* τ *and a marked formula* $\check{\varphi}$ *such that* $\tau \equiv |\check{\varphi}|$*.*

We define $\mathcal{A}_{gtc}$. The set of states of $\mathcal{A}_{gtc}$ consists of occurrences-with-marks of $\mathcal{D}$ and a distinguished (initial) state $*$. Most rules just simulate Definition 8. For example, if τ is the principal occurrence in the conclusion of (λL), $(\lambda x.\check{\varphi})\,\check{\psi}$ is a marked formula such that $\tau \equiv (\lambda x.|\check{\varphi}|)\,|\check{\psi}|$ and n is the premise, then $((\tau, (\lambda x.\check{\varphi})\,\check{\psi}), n, (\tau', \check{\varphi}[\check{\psi}/x]))$ where τ' is the unique occurrence in v that is relevant to τ. Important transition rules are those dealing with $(\sigma L/R)$. Consider the state $\check{\tau} = (\tau, \check{\varphi})$ where τ is the principal occurrence of $(\sigma L/R)$. Let n be the premise and τ' be the unique relevant occurrence in n.

- Case $\check{\varphi} \equiv (\sigma x.\check{\psi})\,\vec{\check{\chi}}$: The automaton just unfolds the fixed-point operator, i.e.,

  $$((\tau, (\sigma x.\check{\psi})\,\vec{\check{\chi}}),\ n,\ (\tau', \check{\psi}[(\sigma x.\check{\psi})/x]\,\vec{\check{\chi}})) \in \delta.$$

- Case $\check{\varphi} \equiv (\sigma_\bullet x.\check{\psi})\,\vec{\check{\chi}}$: Note that there may be other copies of $\sigma_\bullet x.\check{\psi}$ in $\vec{\check{\chi}}$ or $\check{\psi}$. So the automaton has to choose which copy should be tracked, and the transition is nondeterministic. If the automaton decides to track the occurrence of $\sigma_\bullet x.\check{\psi}$ being unfolded, it removes all marks in $\vec{\check{\chi}}$ and tracks the recursive calls of the head occurrence by marking them:

  $$((\tau, (\sigma_\bullet x.\check{\psi})\,\vec{\check{\chi}}),\ n,\ (\tau', |\check{\psi}|[(\sigma_\bullet x.|\check{\psi}|)/x]\,|\vec{\check{\chi}}|)) \in \delta.$$

  Otherwise it removes the mark of the fixed-point operator being unfolded and unfolds it:

  $$((\tau, (\sigma_\bullet x.\check{\psi})\,\vec{\check{\chi}}),\ n,\ (\tau', \check{\psi}[(\sigma x.\check{\psi})/x]\,\vec{\check{\chi}})), \in \delta.$$

  If the rule is (μL) or (νR), then the former transition is an accepting transition. All other transitions for $(\sigma L/R)$ and other rules are not accepting.

The initial state either ignores the input $((*, n, *) \in \delta)$ or nondeterministically chooses an occurrence-with-marks $((*, n, \check{\tau}) \in \delta$ if $\check{\tau}$ is an occurrence in n and has exactly one marked fixed-point operator).

▶ **Lemma 18.** *Let* $(\mathcal{D}, \mathcal{R})$ *be a pre-proof. For every infinite path* π*,* $\pi \in \mathcal{L}(\mathcal{A}_{gtc})$ *if and only if* π *satisfies the global trace condition.*

Proof. Assume that π satisfies the global trace condition. Let $((\tau_i)_{i\geq 0}, p)$ be the pair of a trace (with annotations) and an infinite sequence of natural numbers that witnesses the global trace condition. For each i, let q_i be the longest prefix of p among the annotations in τ_i. Let us mark σ_{q_i} in τ_i and write $\check{\varphi}_i$ for the resulting marked formula. Then $(\tau_i, \check{\varphi}_i)_{i\geq 0}$ is an accepting run.

We prove the converse. Assume a (possibly non-accepting) run, which determines a trace $(\tau_i)_{i\geq 0}$. We annotate $(\tau_i)_{i\geq 0}$ following Definition 8. Let q_i be the sequence assigned to the marked operator in τ_i and p be the limit of $(q_i)_{i\geq 0}$. The transition rules ensure the well-definedness of q_i and p. If the run is accepting, p is infinite since $(q_i)_{i\geq 0}$ must grow infinitely many times. Furthermore it is a left μ-trace or right ν-trace as all accepting transitions are for (μL) or (νR). ◀

We have the following theorem as a corollary.

▶ **Theorem 19.** *The validity checking of a pre-proof* $(\mathcal{D}, \mathcal{R})$ *is decidable.*

5 Soundness

The soundness proof follows the proof strategy of [3]. We prove the claim by contraposition. Assume that the conclusion $\Gamma \vdash \Delta$ of a cyclic proof is invalid. Then there exists a valuation ρ such that $\Gamma \not\models_\rho \Delta$. Since all rules are *locally sound* (i.e. if the premises are valid under a valuation, the conclusion is also valid under the valuation), there exists an infinite path whose sequents are invalid under ρ. Using the global trace condition, we construct an infinite decreasing chain of ordinals, a contradiction.

To be precise, we impose on the infinite path a condition slightly stronger than the invalidity under ρ. To describe the condition, we need fixed-point operators $\mu^\alpha x.\varphi$ and $\nu^\alpha x.\varphi$ annotated by ordinals. The semantics of $\mu^\alpha x.\varphi$ is given by

$$[\![\mu^0 x^T.\varphi]\!](\rho) := \bot_T \quad \text{and} \quad [\![\mu^\alpha x^T.\varphi]\!](\rho) := \bigsqcup_{\beta<\alpha} [\![(\lambda x.\varphi)\,(\mu^\beta x^T.\varphi)]\!](\rho).$$

The definition of $\nu^\alpha x.\varphi$ is similar (but uses the dual operations).

During the construction of the path, we give ordinal annotations to occurrences μ on the left side of $\vdash$ and to occurrences of ν on the right side. The annotation processes are essentially the same as that in Definition 8. Again, the most important case is that τ_i is the principal occurrence of (μL) or (νR). Consider the (μL) case. Then $\tau_i \equiv (\mu^\alpha x.\varphi)\,\vec{\psi}$ (where μ in φ and $\vec{\psi}$ are also annotated). By the assumption of the invalidity of the sequent under ρ, we have $[\![(\mu^\alpha x.\varphi)\,\vec{\psi}]\!](\rho) = \top$. The annotation to τ_{i+1} is $\varphi[\mu^\beta x.\varphi/x]\,\vec{\psi}$ where β is the minimum ordinal such that $[\![\varphi[\mu^\beta x.\varphi]\,\vec{\psi}]\!](\rho) = \top$. By this process, fixed-point operators annotated with the same sequence of natural numbers have the same ordinal annotation. In other words, it defines a function from sequences q of natural numbers appearing in the trace to ordinal numbers α_q. Furthermore one can show that $\alpha_{q.k} < \alpha_q$ (provided that $q.k$ appears in the trace). Therefore, if the path has a left μ-trace witnessed by an infinite sequence p, then $\alpha_{p[0:1]} > \alpha_{p[0:2]} > \alpha_{p[0:3]} > \dots$ is an infinite decreasing chain of ordinals.

The above argument shows the following theorem. A detailed proof is in [11].

▶ **Theorem 20** (Soundness). *If there is a cyclic proof of $\Gamma \vdash \Delta$, then $\Gamma \vdash \Delta$ is valid.*

6 Completeness of Infinitary Variant

Our cyclic proof system is incomplete. Existence of a proof in our cyclic proof system is computably enumerable, but the valid sequents in $\mathrm{HFL}_{\mathbb{N}}$ are not because $\mathrm{HFL}_{\mathbb{N}}$ includes Peano arithmetic. The aim of this section is to find a complete proof system. Following [3], we study the infinitary variant of our cyclic proof system, in which a proof is an infinite tree instead of a finite tree with cycles. More precisely, an *infinitary proof* is a (possibly) infinite derivation tree (without open leaves) of which all infinite paths satisfy the global trace condition. Soundness proof of the previous section is applicable to the infinitary system as well. Here we discuss its completeness.

▶ **Theorem 21.** *Let $\Gamma \vdash \Delta$ be a sequent without higher-order free variables. That means, every free variable of the sequent is of type $\mathbf{N}$ or of type $\mathbf{N}^k \to \mathbf{\Omega}$ for some $k \geq 0$. If $\Gamma \models \Delta$, then $\Gamma \vdash \Delta$ is provable in the infinitary proof system.*

We give a sketch of the proof. A detailed proof is given in Appendix B. We can assume without loss of generality that the sequent has no free variable of type $\mathbf{N}$. We start from a (finite) pre-proof consisting only of the root node, which is an open leaf, and iteratively expand each open leaf by applying rules $(\vee L/R)$, $(\wedge L/R)$, $(\lambda L/R)$, and $(\sigma L/R)$. The only restriction is that the expansion must be "fair": the fairness condition we impose is that (i) each open leaf will eventually be expanded, and (ii) for every path, each formula in a sequent will eventually appear in the principal position (unless the path is terminated by the axiom rule). This process generates a growing sequence of (finite) pre-proofs, and the infinitary proof is defined as its limit.

Now it suffices to show the global trace condition of the above constructed candidate proof, but this is the hardest part of the proof. Assume an infinite path π. Since $\Gamma \models \Delta$, for each valuation ρ, there exists $\varphi \in \Gamma$ such that $[\![\varphi]\!](\rho) = \bot$ or $\psi \in \Delta$ such that $[\![\varphi]\!](\rho) = \top$. Then, by a similar argument to the proof of soundness, existence of a left ν-trace or a right μ-trace leads to a contradiction. It is worth noting that the construction is "dual": whereas in the soundness proof we ensure $[\![\tau_i]\!](\rho) = \top$ for a left-trace $(\tau_i)_{i\geq 0}$, here $[\![\tau_i]\!](\rho) = \bot$ is kept. This difference allows us to construct a trace of the intended path π. By appropriately choosing ρ, one can ensure that the generated trace is infinite; hence we have constructed an infinite trace that is not left ν- nor right μ-trace. The proof is completed by the following lemma, which is technical but often used in the analysis of HFL (cf. [4, Lemma 6 & 7], [10, Lemma 26, Appendix E.2] and [15, Lemma 14]):

▶ **Lemma 22.** *Every infinite trace $(\tau_i)_{i\leq 0}$ is either a μ-trace or a ν-trace but not both.*

▶ Remark 23. We are not sure if the proof can be extended to sequents with higher-order free variables. The problem is the construction of ρ. In the current setting, for each free variable f of type $\mathbf{N}^k \to \mathbf{\Omega}$, the valuation ρ is defined for follows: for each k-tuple $\vec{m}$ of natural numbers, (i) if $f\,\vec{m}$ appears on the left side of a sequent in the path, then $\rho(f)(\vec{m}) := \top$; (ii) if $f\,\vec{m}$ appears on the right side of a sequent in the path, then $\rho(f)(\vec{m}) := \bot$; and (iii) otherwise the value of $\rho(f)(\vec{m})$ is arbitrary. The point is that the values of arguments of each fully-applied occurrence of f is canonically determined. This construction of ρ is no longer possible in the presence of a higher-order free variable, say $g : \mathbf{\Omega} \to \mathbf{\Omega}$. When $g\,(f\,\vec{m})$ appears on the left side, there are two assignments that make this formula true, namely $\rho_1(g)(\bot) = \top$ with $\rho_1(f)(\vec{m}) = \bot$ and $\rho_2(g)(\top) = \top$ with $\rho_2(f)(\vec{m}) = \top$. ⌟

7 Related Work

7.1 Cyclic Proof Systems

The idea of cyclic proof can at least be traced back to the work of Sprenger and Dam [13] on a cyclic proof system called S_{glob}, where proofs are explicitly annotated with ordinals. The ordinal annotations are not convenient for automated theorem proving.

Brotherston and Simpson [3] proposed a cyclic proof system called CLKID for a first-order predicate logic with inductive definitions, which does not require ordinal annotations. They have proved soundness of CLKID, and also shown that a proof system with explicit induction rules (called LKID) can be embedded into CLKID; we have shown analogous results in Theorem 20 and Proposition 14 for $\mathrm{HFL}_{\mathbb{N}}$. Doumane [7] formalized a cyclic proof system for the linear-time (propositional) μ-calculus, which features alternating fixed-points. Our global trace condition has been inspired by her trace condition.

Besides cyclic proof systems for ordinary first-order logics, cyclic proof systems have also been proposed for separation logics [1, 2, 14], and automated tools have been developed based on those cyclic proof systems.

7.2 Proof Systems and Games for HFL

HFL was originally proposed by Viswanathan and Viswanathan [16], and its extensions with arithmetic (such as $\mathrm{HFL}_{\mathbb{N}}$) have recently been drawing attention in the context of higher-order program verification [10, 17].

For pure HFL (without natural numbers), Kobayashi et al. [8] proposed a type-based inference system for proving the validity of HFL formulas.[5] It is left for future work to study the relationship between their type system and our cyclic proof system; it would be interesting to see whether their type system can be embedded in our cyclic proof system.

Tsukada [15] gave a game-based characterization of $\mathrm{HFL}_{\mathbb{N}}$. His game may be considered a special case of the infinitary version of our proof system, where formulas are restricted to those whose free variables have only natural number types. Burn et al. [5] proposed a refinement type system for HoCHC, which is closely related to the ν-only fragment of $\mathrm{HFL}_{\mathbb{N}}$. Their proof system is incomplete, and does not support fixed-point alternations.

8 Conclusion

We have proposed a cyclic proof system for $\mathrm{HFL}_{\mathbb{N}}$, a higher-order logic with natural numbers and alternating fixed-points, which we expect to be useful for higher-order program verification. Our proof system has been inspired by previous cyclic proof systems for first-order logics [3, 7]. We have shown the soundness of the proof system and the decidability of the global trace condition. We have also shown a restricted form of standard completeness for the infinitary version of our proof system.

Constructing a (semi-)automated tool based on our cyclic proof system is left for future work. On the theoretical side, we plan to study whether Henkin completeness and the cut elimination property hold for (possibly a variation of) our cyclic proof system.

References

1 James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. Automated cyclic entailment proofs in separation logic. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2011. doi:10.1007/978-3-642-22438-6_12.

2 James Brotherston, Nikos Gorogiannis, and Rasmus Lerchedahl Petersen. A generic cyclic theorem prover. In Ranjit Jhala and Atsushi Igarashi, editors, *Programming Languages and Systems*, pages 350–367, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

3 James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, December 2011.

4 Florian Bruse. Alternating parity Krivine automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2014. doi:10.1007/978-3-662-44522-8_10.

5 Toby Cathcart Burn, C.-H. Luke Ong, and Steven J. Ramsay. Higher-order constrained Horn clauses for verification. *PACMPL*, 2(POPL):11:1–11:28, 2018. doi:10.1145/3158099.

[5] Actually, they formalized a type system for model checking, which can also be used as a proof system for proving validity of $\mathrm{HFL}_{\mathbb{N}}$ formulas without natural numbers.

6 Patrick Cousot and Radhia Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific J. Math.*, 82(1):43–57, 1979. URL: `https://projecteuclid.org:443/euclid.pjm/1102785059`.

7 Amina Doumane. *On the infinitary proof theory of logics with fixed points.* Theses, Université Sorbonne Paris Cité, June 2017.

8 Naoki Kobayashi, Étienne Lozes, and Florian Bruse. On the relationship between higher-order recursion schemes and higher-order fixpoint logic. *ACM SIGPLAN Notices*, 52(1):246–259, 2017.

9 Naoki Kobayashi, Takeshi Nishikawa, Atsushi Igarashi, and Hiroshi Unno. Temporal verification of programs via first-order fixpoint logic. In *International Static Analysis Symposium*, pages 413–436. Springer, 2019.

10 Naoki Kobayashi, Takeshi Tsukada, and Keiichi Watanabe. Higher-order program verification via HFL model checking. In *European Symposium on Programming*, pages 711–738. Springer, 2018.

11 Mayuko Kori, Takeshi Tsukada, and Naoki Kobayashi. A cyclic proof system for HFL$_{\mathbf{N}}$. *CoRR*, 2020. A longer version. `arXiv:2010.14891`.

12 Étienne Lozes. A type-directed negation elimination. In Ralph Matthes and Matteo Mio, editors, *Proceedings Tenth International Workshop on Fixed Points in Computer Science, FICS 2015, Berlin, Germany, September 11-12, 2015*, volume 191 of *EPTCS*, pages 132–142, 2015. `doi:10.4204/EPTCS.191.12`.

13 Christoph Sprenger and Mads Dam. On the structure of inductive reasoning: Circular and tree-shaped proofs in the μ-calculus. In *Foundations of Software Science and Computation Structures*, pages 425–440. Springer Berlin Heidelberg, 2003.

14 Gadi Tellez and James Brotherston. Automatically verifying temporal properties of pointer programs with cyclic proof. *J. Autom. Reasoning*, 64(3):555–578, 2020. `doi:10.1007/s10817-019-09532-0`.

15 Takeshi Tsukada. On computability of logical approaches to branching-time property verification of programs. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, page 886–899, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3373718.3394766`.

16 Mahesh Viswanathan and Ramesh Viswanathan. A higher order modal fixed point logic. In *International Conference on Concurrency Theory*, pages 512–528. Springer, 2004.

17 Keiichi Watanabe, Takeshi Tsukada, Hiroki Oshikawa, and Naoki Kobayashi. Reduction from branching-time property verification of higher-order programs to HFL validity checking. In *Proceedings of the 2019 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, PEPM 2019, page 22–34, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3294032.3294077`.

A Definition of relevant occurrences

Here we give a detailed definition of the notion of *relevant occurrences* introduced after Definition 7. We have already defined relevant occurrences for the rules ($\wedge L$), ($\vee L$), and (Mono). We give the definition for the other rules. Below, n_i refers to the conclusion of each rule, and n_{i+1} refers to one of the premises. In all the rules, each formula in Γ or Δ in n_{i+1} is a relevant occurrence of the corresponding formula in Γ or Δ in n_i.

- In (Cut), φ in n_{i+1} is not relevant to any formula in n_i.
- In (Ctr L) and (Ctl R), both occurrences of φ are relevant to φ in n_i.
- In (Ex L) and (Ex R), ψ (φ, resp.) in n_{i+1} is a relevant occurrence of ψ (φ, resp.) in n_i.
- In (Subst), each formula ψ in Γ (Δ, resp.) of n_{i+1} is relevant to $\psi[\varphi/x]$ in $\Gamma[\varphi/x]$ ($\Delta[\varphi/x]$, resp.) of n_i.

- In (=L), each formula $\psi[t/x, s/y]$ in $\Gamma[t/x, s/y]$ ($\Delta[t/x, s/y]$, resp.) of n_{i+1} is relevant to $\psi[s/x, t/y]$ in $\Gamma[s/x, t/y]$ ($\Delta[s/x, t/y]$, resp.) of n_i.
- In ($\vee$R), φ and ψ in n_{i+1} are relevant to $\varphi \vee \psi$ in n_i.
- In ($\wedge$R), φ and ψ in n_{i+1} are relevant to $\varphi \wedge \psi$ in n_i.
- In (λL) and (λR), $\varphi[\psi/x]\,\vec{\psi}$ in n_{i+1} is relevant to $(\lambda x.\varphi)\,\psi\,\vec{\psi}$ in n_i.
- In (σL) and (σR), $\varphi[\sigma x.\varphi/x]\,\vec{\psi}$ in n_{i+1} is relevant to $(\sigma x.\varphi)\,\vec{\psi}$ in n_i.
- In (Nat), $N\,x$ in n_{i+1} is not relevant to any formula in n_i.
- In (P2), $s = t$ in n_{i+1} is relevant to $\mathbf{S}s = \mathbf{S}t$ in n_i.

B Proof of Theorem 21

Let f be a function on a complete lattice. For each ordinal α, we define $f^{\alpha}(\bot)$ by $f^0(\bot) = \bot$ and $f^{\alpha}(\bot) = \bigsqcup_{\beta<\alpha} f(f^{\beta}(\bot))$. Similarly $f^{\alpha}(\top)$ is defined by $f^0(\top) = \top$ and $f^{\alpha}(\top) = \bigsqcap_{\beta<\alpha} f(f^{\beta}(\top))$.

▶ **Lemma 24** (Cousot-Cousot [6]). *Let $(C, \leq)$ be a complete lattice, f be a monotone function and γ be an ordinal number greater than the cardinality of C. Then $f^{\gamma}(\bot)$ is the least fixed-point of f and $f^{\gamma}(\top)$ is the greatest fixed-point of f.*

We extend the definition of formulas by allowing μ and ν to have ordinal numbers like $\mu^{\alpha}, \nu^{\alpha}$. The definition of $[\![.]\!]$ for $\mu^{\alpha}, \nu^{\alpha}$ is as follows.

$$[\![\mathcal{H} \vdash (\mu^{\alpha} x^T.\varphi)\,\vec{\psi}]\!](\rho) = ((\lambda v.[\![\mathcal{H}, x : T \vdash \varphi]\!](\rho[x \mapsto v]))^{\alpha}\,(\bot_n))\,\rho(\vec{\psi})$$
$$[\![\mathcal{H} \vdash (\nu^{\alpha} x^T.\varphi)\,\vec{\psi}]\!](\rho) = ((\lambda v.[\![\mathcal{H}, x : T \vdash \varphi]\!](\rho[x \mapsto v]))^{\alpha}\,(\top_n))\,\rho(\vec{\psi})$$

▶ **Lemma 25.** *Let $\Gamma \vdash \Delta$ be a sequent and $x^{\mathbf{N}} \in FV(\Gamma, \Delta)$. If $(\Gamma[\mathbf{S}^n\mathbf{Z}/x] \vdash \Delta[\mathbf{S}^n\mathbf{Z}/x])$ is cut-free provable for all $n \in \mathbb{N}$ then $\Gamma \vdash \Delta$ is cut-free provable.*

Proof. Assume $(\Gamma[\mathbf{S}^n\mathbf{Z}/x] \vdash \Delta[\mathbf{S}^n\mathbf{Z}/x])$ is cut-free provable for all $n \in \mathbb{N}$, and let Π_n be the proof. We define $(\Pi^i)_{i\geq 0}$ recursively whose root node is $(N\,x, \Gamma[\mathbf{S}^i x/x] \vdash \Delta[\mathbf{S}^i x/x])$ as follows:

$$\Pi^i := \dfrac{\dfrac{\dfrac{\Pi_i}{\Gamma[\mathbf{S}^i\mathbf{Z}/x] \vdash \Delta[\mathbf{S}^i\mathbf{Z}/x]}}{x = \mathbf{Z}, \Gamma[\mathbf{S}^i x/x] \vdash \Delta[\mathbf{S}^i x/x]}\,(=\mathrm{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\Pi^{i+1}}{N\,x, \Gamma[\mathbf{S}^{i+1}x/x] \vdash \Delta[\mathbf{S}^{i+1}x/x]}}{N\,x', \Gamma[\mathbf{S}^i\mathbf{S}x'/x] \vdash \Delta[\mathbf{S}^i\mathbf{S}x'/x]}\,(\mathrm{Subst})}{x = \mathbf{S}x', N\,x', \Gamma[\mathbf{S}^i x/x] \vdash \Delta[\mathbf{S}^i x/x]}\,(=\mathrm{L})}{\exists x'.x = \mathbf{S}x' \wedge N\,x', \Gamma[\mathbf{S}^i x/x] \vdash \Delta[\mathbf{S}^i x/x]}\,(\exists\mathrm{L}, \wedge\mathrm{L})}{N\,x, \Gamma[\mathbf{S}^i x/x] \vdash \Delta[\mathbf{S}^i x/x]}\,(\mu\mathrm{L})$$

Then $\Pi := \dfrac{\dfrac{\Pi^0}{N\,x, \Gamma \vdash \Delta}}{\Gamma \vdash \Delta}\,(\mathrm{Nat})$ is a pre-proof of $\Gamma \vdash \Delta$. For all infinite paths π in the pre-proof Π, if π goes through some Π_i then there exists left μ-trace or right ν-trace because Π_i is a proof, and otherwise, we can trace the left μ in N. Therefore, Π is a proof of $\Gamma \vdash \Delta$. ◀

▶ **Corollary 26.** *Let $\Gamma \vdash \Delta$ be a sequent and $\vec{x}$ are all natural number free variables in $\Gamma \cup \Delta$. If $(\Gamma[\vec{n}/\vec{x}] \vdash \Delta[\vec{n}/\vec{x}])$ is cut-free provable for all $\vec{n} \in \vec{\mathbb{N}}$ then $\Gamma \vdash \Delta$ is cut-free provable.*

Thanks to this corollary, it suffices to prove completeness of valid sequents whose free variables have type $\mathbf{N}^k \to \mathbf{\Omega}$ for some $k \geq 0$. In other words, we can assume without loss of generality that the sequent of interest has no free variable of type $\mathbf{N}$.

The next two definitions construct a tree T_ω of $\Gamma \vdash \Delta$ without natural number free variables. The first definition is needed to deal with all formulas in a fair manner.

▶ **Definition 27** (Schedule). A schedule element *is a formula of the form* $\varphi \vee \psi, \varphi \wedge \psi, (\lambda x.\varphi)\, \psi\, \vec{\psi}, (\sigma x.\varphi)\, \vec{\psi}$. *We call* $(E_i)_{i \geq 0}$ a schedule *if* E_i *is a schedule element for all* i *and every schedule element appears infinitely often in* $(E_i)_{i\geq 0}$.

There exists a schedule and we fix one.

▶ **Definition 28** (T_ω). *Let* $\Gamma \vdash \Delta$ *be a valid sequent that does not have natural number or higher-order free variables. That is, the type of each free variable in* $\Gamma \vdash \Delta$ *is* $\mathbf{N}^n \to \mathbf{\Omega}$ *for some* $n \in \mathbb{N}$.

Then we will define trees $(T_i)_{i\geq 0}$ *whose roots are* $\Gamma \vdash \Delta$ *inductively by using a schedule* $(E_i)_{i\geq 0}$. *First,* T_0 *is defined by* $T_0 := \Gamma \vdash \Delta$.

Assume T_i *is already defined. Then we define* T_{i+1} *in* T_i *by replacing each open leaf* $\Gamma' \vdash \Delta'$ *with the following tree:*

- *If there exists a formula* $\varphi \in \Gamma' \cap \Delta'$*:*
$$\dfrac{\dfrac{}{\varphi \vdash \varphi}\,(\textit{Axiom})}{\Gamma' \vdash \Delta'}\,(\textit{Wk})$$
- *If* $(\mathbf{S}^n\mathbf{Z} = \mathbf{S}^m\mathbf{Z}) \in \Gamma'$ *for some different natural numbers* n, m*:*
$$\dfrac{\dfrac{\dfrac{}{\mathbf{Z} = \mathbf{S}^{|n-m|}\mathbf{Z} \vdash}\,(\textit{P1})}{\mathbf{S}^n\mathbf{Z} = \mathbf{S}^m\mathbf{Z} \vdash}\,(\textit{P2})}{\Gamma' \vdash \Delta'}\,(\textit{Wk})$$
- *If* $(\mathbf{S}^n\mathbf{Z} = \mathbf{S}^n\mathbf{Z}) \in \Delta'$ *for some* n*:*
$$\dfrac{\dfrac{}{\vdash \mathbf{S}^n\mathbf{Z} = \mathbf{S}^n\mathbf{Z}}\,(\textit{=R})}{\Gamma' \vdash \Delta'}\,(\textit{Wk})$$
- *Otherwise: This replacement is performed in such a way that each formula is chosen as the target of expansion in a fair manner, so that every formula is expanded at some point.*
  - *Case* ($E_i \equiv \varphi \vee \psi$)*:*
    - *if* $E_i \in \Gamma'$*:*
$$\dfrac{\dfrac{\Gamma', \varphi \vdash \Delta' \qquad \Gamma', \psi \vdash \Delta'}{\Gamma', \varphi \vee \psi \vdash \Delta'}\,(\vee L)}{\Gamma' \vdash \Delta'}\,(\textit{Ctr})$$
    - *if* $E_i \in \Delta'$*:*
$$\dfrac{\dfrac{\Gamma' \vdash \varphi, \psi, \Delta'}{\Gamma' \vdash \varphi \vee \psi, \Delta'}\,(\vee R)}{\Gamma' \vdash \Delta'}\,(\textit{Ctr})$$
  - *Case* ($E_i \equiv \varphi \wedge \psi$)*: The tree is defined in the similar way to the case* $E_i \equiv \varphi \vee \psi$.
  - *Case* ($E_i \equiv (\lambda x.\varphi)\, \psi\, \vec{\psi}$)*:*
    - *if* $E_i \in \Gamma'$*:*
$$\dfrac{\dfrac{\Gamma', \varphi[\psi/x]\, \vec{\psi} \vdash \Delta'}{\Gamma', (\lambda x.\varphi)\, \psi\, \vec{\psi} \vdash \Delta'}\,(\lambda L)}{\Gamma' \vdash \Delta'}\,(\textit{Ctr})$$
    - *if* $E_i \in \Delta'$*:*
$$\dfrac{\dfrac{\Gamma' \vdash \varphi[\psi/x]\, \vec{\psi}, \Delta'}{\Gamma' \vdash (\lambda x.\varphi)\, \psi\, \vec{\psi}, \Delta'}\,(\lambda R)}{\Gamma' \vdash \Delta'}\,(\textit{Ctr})$$
  - *Case* ($E_i \equiv (\sigma x.\varphi)\, \vec{\psi}$)*:*

- *if $E_i \in \Gamma'$:*

$$\dfrac{\dfrac{\Gamma', \varphi[\sigma x.\varphi/x]\,\vec{\psi} \vdash \Delta'}{\Gamma', (\sigma x.\varphi)\,\vec{\psi} \vdash \Delta'}\,(\sigma L)}{\Gamma' \vdash \Delta'}\,(\mathit{Ctr})$$

- *if $E_i \in \Delta'$:*

$$\dfrac{\dfrac{\Gamma' \vdash \varphi[\sigma x.\varphi/x]\,\vec{\psi}, \Delta'}{\Gamma' \vdash (\sigma x.\varphi)\,\vec{\psi}, \Delta'}\,(\sigma R)}{\Gamma' \vdash \Delta'}\,(\mathit{Ctr})$$

Note that for all $i \geq 0$, $T_i \subseteq T_{i+1}$ and each sequent of an open leaf in T_{i+1} includes the sequent of the corresponding leaf in T_i. We define T_ω to be $\lim_{i\to\infty} T_i$.

We aim to prove that T_ω is a proof of $\Gamma \vdash \Delta$.

▶ **Definition 29** ($\Gamma_\omega \vdash_{n/\pi} \Delta_\omega, \rho_\omega$)**.** *For all open leaves n of T_ω, we define $\Gamma_\omega \vdash_n \Delta_\omega$ as the leaf sequent. For all infinite paths $\pi = (\pi_i)_{i\geq 0}$ in T_ω, we define $\Gamma_\omega \vdash_\pi \Delta_\omega$ as $\lim_{i\to\infty}(\Gamma_i \vdash \Delta_i)$ where $\Gamma_i \vdash \Delta_i$ is the sequent of π_i.*

A valuation ρ_ω of $\Gamma_\omega \vdash_{n/\pi} \Delta_\omega$ is defined as below:

$$\rho_\omega(x^{\mathbf{\Omega}}) := \begin{cases} \top & \textit{if } x \in \Gamma_\omega \\ \bot & \textit{otherwise} \end{cases}$$

$$\rho_\omega(f^{\mathbf{N}^n \to \mathbf{\Omega}}) := \lambda \vec{x}^{\mathbf{N}^n}. \begin{cases} \top & \textit{if } f\,\vec{x} \in \Gamma_\omega \\ \bot & \textit{otherwise} \end{cases}$$

▶ **Lemma 30.** *T_ω is a proof.*

Proof. We aim to show that T_ω is a pre-proof and T_ω satisfies the global trace condition.

Assume T_ω is not a pre-proof. There exists an open leaf n in T_ω, and all formulas of the leaf are of the form $x^{\mathbf{\Omega}}$, $f\,\vec{t}$ or $\mathbf{S}^m\mathbf{Z} = \mathbf{S}^n\mathbf{Z}$. Then the definition of ρ_ω of $\Gamma_\omega \vdash_n \Delta_\omega$ induces $[\![\varphi]\!]_{\rho_\omega} = \top$ for all $\varphi \in \Gamma_\omega$ and $[\![\varphi]\!]_{\rho_\omega} = \bot$ for all $\varphi \in \Delta_\omega$. This contradicts to $\Gamma \models \Delta$ because $\Gamma \subseteq \Gamma_\omega$ and $\Delta \subseteq \Delta_\omega$. Therefore T_ω is a pre-proof.

We next show that T_ω satisfies the global trace condition. For all infinite paths π in T_ω, we define ρ_ω of $\Gamma_\omega \vdash_\pi \Delta_\omega$ as Definition 29. We have $\Gamma \models \Delta$, $\Gamma \subseteq \Gamma_\omega$ and $\Delta \subseteq \Delta_\omega$ by the assumption and the construction of T_ω. Therefore, it follows that there exists either (1) a formula $\varphi \in \Gamma_\omega$ such that $[\![\varphi]\!]_{\rho_\omega} = \bot$ or (2) a formula $\varphi \in \Delta_\omega$ such that $[\![\varphi]\!]_{\rho_\omega} = \top$. We now give the proof only for the case (1). The other case can be also proved by the same method.

Let j be a number such that $\varphi \in \pi_j$. We will show that there exists a left μ-trace $(\tau_i)_{i\geq j}$ in π starting from φ.

We define $(\tau_i)_{i\geq j}$ and $(\tau'_i)_{i\geq j}$ which satisfy the following conditions:

1. $\tau_i \in \Gamma_i$;
2. we can get τ_i by forgetting ordinal numbers of τ'_i;
3. each ν in τ'_i has an ordinal number;
4. $[\![\tau'_i]\!]_{\rho_\omega} = \bot$;
5. each ordinal number in τ'_{i+1} is less than or equal to the corresponding ordinal number in τ'_i.

The definition of $(\tau_i)_{i\geq j}$ and $(\tau'_i)_{i\geq j}$ as below:

- $\tau_j := \varphi$. Because $[\![\varphi]\!]_{\rho_\omega} = \bot$, we can get φ' which satisfies $[\![\varphi']\!]_{\rho_\omega} = \bot$ by assigning ordinal numbers to all ν in φ.
- Assume τ_i and τ'_i are defined. Below, ψ' means the corresponding subformula of τ'_i for each subformula ψ of τ_i.

- If $\tau_i \equiv x$:
  Since $x \in \Gamma_i$, $[\![x]\!]_{\rho_\omega} = \top$ because of the definition of ρ_ω. By the assumption of τ'_i, $[\![(x)']\!]_{\rho_\omega} = [\![x]\!]_{\rho_\omega} = \bot$. This is a contradiction.
- If $\tau_i \equiv (\mathbf{S}^n\mathbf{Z} = \mathbf{S}^m\mathbf{Z})$:
  By the assumption of τ'_i, $[\![(\mathbf{S}^n\mathbf{Z} = \mathbf{S}^m\mathbf{Z})']\!]_{\rho_\omega} = [\![\mathbf{S}^n\mathbf{Z} = \mathbf{S}^m\mathbf{Z}]\!]_{\rho_\omega} = \bot$. Then $n \neq m$ holds and π is not infinite because of the construction of T_ω. This contradicts the fact that π is an infinite path.
- If $\tau_i \equiv f\,\vec{t}$:
  Since $f\,\vec{t} \in \Gamma_i$, $[\![f\,\vec{t}]\!]_{\rho_\omega} = \top$ because of the definition of ρ_ω. By the assumption of τ'_i, $[\![(f\,\vec{t})']\!]_{\rho_\omega} = [\![f\,\vec{t}]\!]_{\rho_\omega} = \bot$. This is a contradiction.
- If $E_i \not\equiv \tau_i$ then $\tau_{i+1} := \tau_i$ and $\tau'_{i+1} := \tau'_i$.
- Otherwise:
  - Case $\tau_i \equiv (\psi \vee \chi)$:
    Since $[\![\psi' \vee \chi']\!]_{\rho_\omega} = \bot$, $[\![\psi']\!]_{\rho_\omega} = [\![\chi']\!]_{\rho_\omega} = \bot$. $\tau_{i+1} := \psi, \tau'_{i+1} := \psi'$ if π goes to the left leaf, and otherwise $\tau_{i+1} := \chi, \tau'_{i+1} := \chi'$. In each case, $[\![\tau'_{i+1}]\!]_{\rho_\omega} = \bot$ holds.
  - Case $\tau_i \equiv (\psi \wedge \chi)$:
    Since $[\![\psi' \wedge \chi']\!]_{\rho_\omega} = \bot$, either $[\![\psi']\!]_{\rho_\omega}$ or $[\![\chi']\!]_{\rho_\omega}$ is $\bot$.
    $\tau_{i+1} := \psi, \tau'_{i+1} := \psi'$ if $[\![\psi']\!]_{\rho_\omega} = \bot$, and otherwise $\tau_{i+1} := \chi, \tau'_{i+1} := \chi'$.
  - Case $\tau_i \equiv (\lambda x.\psi)\,\chi\,\vec{\theta}$:
    $\tau_{i+1} := \psi[\chi/x]\,\vec{\theta}, \tau'_{i+1} := \psi'[\chi'/x]\,\vec{\theta'}$
    then $[\![\psi'[\chi'/x]\,\vec{\theta'}]\!]_{\rho_\omega} = [\![(\lambda x.\psi')\,\chi'\,\vec{\theta'}]\!]_{\rho_\omega} = \bot$.
  - Case $\tau_i \equiv (\mu x.\psi)\,\vec{\theta}$:
    $\tau_{i+1} := \psi[\mu x.\psi/x]\,\vec{\theta}$, $\tau'_{i+1} := \psi'[\mu x.\psi'/x]\,\vec{\theta'}$
    then $[\![(\psi'[\mu x.\psi'/x])\,\vec{\theta'}]\!]_{\rho_\omega} = [\![(\mu x.\psi')\,\vec{\theta'}]\!]_{\rho_\omega} = \bot$.
  - Case $\tau_i \equiv (\nu x.\psi)\,\vec{\theta}$:
    Let α be the ordinal number assigned to the head ν of $(\nu x.\varphi)\,\vec{\psi}$. If $\alpha = 0$ then $[\![(\nu^0 x^T.\varphi')\,\vec{\psi'}]\!]_{\rho_\omega} = [\![(\top_T)\,\vec{\psi'}]\!]_{\rho_\omega} = \top$ and this contradicts the assumption.

$$
\begin{aligned}
[\![(\nu^\alpha x.\psi')\,\vec{\theta'}]\!]_{\rho_\omega} &= ((\lambda v.[\![\psi']\!]_{\rho_\omega[x\mapsto v]})^\alpha\,(\top_n))\,[\![\vec{\theta'}]\!]_{\rho_\omega} \\
&= (\bigsqcap_{\beta<\alpha} (\lambda v.[\![\psi']\!]_{\rho_\omega[x\mapsto v]})\,((\lambda v.[\![\psi']\!]_{\rho_\omega[x\mapsto v]})^\beta\,(\top_n)))\,[\![\vec{\theta'}]\!]_{\rho_\omega} \\
&= \bot
\end{aligned}
$$

There exists $\beta < \alpha$ such that $((\lambda v.[\![\psi']\!]_{\rho_\omega[x\mapsto v]})\,((\lambda v.[\![\psi']\!]_{\rho_\omega[x\mapsto v]})^\beta\,(\top_n)))\,[\![\vec{\theta'}]\!]_{\rho_\omega} = \bot$.
$\tau_{i+1} := \psi[\nu x.\psi/x]\,\vec{\theta}$, $\tau'_{i+1} := \psi'[\nu^\beta x.\psi'/x]\,\vec{\theta'}$ then

$$
\begin{aligned}
[\![(\psi'[\nu^\beta x.\psi'/x])\,\vec{\theta'}]\!]_{\rho_\omega} &= ((\lambda v.[\![\psi']\!]_{\rho_\omega[x\mapsto v]})\,([\![\nu^\beta x.\psi']\!]_{\rho_\omega}))\,[\![\vec{\theta'}]\!]_{\rho_\omega} \\
&= ((\lambda v.[\![\psi']\!]_{\rho_\omega[x\mapsto v]})\,((\lambda v.[\![\psi']\!]_{\rho_\omega[x\mapsto v]})^\beta\,(\top_n)))\,[\![\vec{\theta'}]\!]_{\rho_\omega} \\
&= \bot
\end{aligned}
$$

If this $(\tau_i)_{i\geq j}$ is a ν-trace, there is an infinite sequence p of the trace by the definition of ν-trace. For all $n \geq 1$, there is a $\nu_{p[0:n]}$ in some τ_i and we denote by α_n the ordinal number assigned to this ν in τ'_i. By the definition of $(\tau'_i)_{i\geq j}$, all ordinal numbers assigned to $\nu_{p[0:n]}$ is equal to α_n. Then for all $n \geq 1$, there exists i such that $(\tau_i \equiv (\nu_{p[0:n]}x.\psi)\,\vec{\theta}) \to (\tau_{i+1} \equiv (\psi[\nu_{p[0:n+1]}x.\psi/x])\,\vec{\theta})$. Hence $\alpha_n > \alpha_{n+1}$ holds for all $n \geq 1$ so $(\alpha_n)_{n\geq 1}$ is a decreasing sequence of ordinal numbers. However, such a sequence does not exist, hence a contradiction. By Lemma 22, it follows that $(\tau_i)_{i\geq j}$ is a left μ-trace of π.

Therefore T_ω satisfies the global trace condition. ◀

Proof (Theorem 21). Let $\vec{x}$ be all natural number free variables of $\Gamma \vdash \Delta$. By Corollary 26, it suffices to show that $\Gamma[\vec{n}/\vec{x}] \vdash \Delta[\vec{n}/\vec{x}]$ is cut-free provable for all $\vec{n} \in \vec{\mathbb{N}}$. For each $\vec{n} \in \vec{\mathbb{N}}$, we construct T_ω for $\Gamma[\vec{n}/\vec{x}] \vdash \Delta[\vec{n}/\vec{x}]$ as Definition 28, then Lemma 30 concludes that T_ω is a proof. Besides, T_ω is cut-free by the construction of T_ω. ◀

Compositional Modelling of Network Games

Elena Di Lavore
Department of Software Science, Tallinn University of Technology, Estonia
elena.di@taltech.ee

Jules Hedges
Department of Computer and Information Sciences, University of Strathclyde, Glasgow, UK
jules.hedges@strath.ac.uk

Paweł Sobociński
Department of Software Science, Tallinn University of Technology, Estonia
pawel.sobocinski@taltech.ee

Abstract

The analysis of games played on graph-like structures is of increasing importance due to the prevalence of social networks, both virtual and physical, in our daily life. As well as being relevant in computer science, mathematical analysis and computer simulations of such distributed games are vital methodologies in economics, politics and epidemiology, amongst other fields. Our contribution is to give compositional semantics of a family of such games as a well-behaved mapping, a strict monoidal functor, from a category of open graphs (syntax) to a category of open games (semantics). As well as introducing the theoretical framework, we identify some applications of compositionality.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases game theory, category theory, network games, open games, open graphs, compositionality

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.30

Funding *Elena Di Lavore* and *Paweł Sobociński* are supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001).

1 Introduction

Compositionality concerns finding homomorphic mappings

$$\mathbf{Syntax} \to \mathbf{Semantics}. \tag{1}$$

This important concept originated in formal logic [20, 21], and is at the centre of formal semantics of programming languages [23]. In recent years, there have been several *2-dimensional* examples [6, 4, 1], where both **Syntax** and **Semantics** are symmetric monoidal categories. Usually **Syntax** is freely generated from a (monoidal) signature, possibly modulo equations. This opens up the possibility of recursive definitions and proofs by structural induction, familiar from our experience with ordinary, 1-dimensional syntax.

In this paper, we consider an instance of (1) that is—at first sight—quite different from the usual concerns of programming and logic: network games [5], also known as graphical games. Network games involve agents that play concurrently, and share information based on an underlying, ambient network topology. Indeed, the utility of each player typically depends on the structure of the network. An interesting application is social networks [10], but they also feature in economics [11], politics [22] and epidemiology [16], amongst other fields. (These games should not be confused with classes of dynamic games played on graphs, such as parity games and pursuit games, which are not within the scope of this paper.)

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 30; pp. 30:1–30:24
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In formal accounts of network games, graphs represent network topologies. Players are identified with graph vertices, and their utility is influenced only by their choices and those of their immediate neighbours. Network games are thus "games on graphs". An example is the *majority game*: players "win" when they agree with the majority of their neighbours.

In what way do such games fit into the conceptual framework of (1)? Our main contribution is the framing of certain network games as monoidal functors from a suitable category of *open graphs* **Grph** [7], our **Syntax**, to the category of open games **Game** [13], our **Semantics**. Given a network game $\mathcal{N}$ (e.g. the majority game), such games are functors

$$F_{\mathcal{N}} \colon \mathbf{Grph} \to \mathbf{Game} \tag{2}$$

that, for any *closed* graph $\Gamma \in \mathbf{Grph}$, yield the game $F_{\mathcal{N}}(\Gamma)$, which is the game $\mathcal{N}$ played on Γ. However, compositionality means that such games are actually "glued together" from simpler, *open* games. In fact, $F_{\mathcal{N}}$ maps each vertex of Γ to an open game called the *utility-maximising player*, and the connectivity of Γ is mapped, following the rules of $\mathcal{N}$, to structure in **Game**.

Our contribution thus makes the intuitively obvious idea that the data of network games is dependent on their network topology precise. Concrete descriptions of network games, given a fixed topology, are often quite involved: our approach means that they can be derived in a principled way from basic building blocks. In some cases, the compositional description can also help in the mathematical analysis of games. For example, in the case of the majority game, the right decomposition of a network topology Γ as an expression in **Grph** can yield a recipe for the Nash equilibrium of $F_{\mathcal{N}}(\Gamma)$ in **Game** in terms of the equilibria of the open games obtained via $F_{\mathcal{N}}$ from the open graphs in the decomposition. As it happens when solving optimization problems, a compositional analysis of the equilibria is possible only when the game has optimal substructure, which is the case for the majority game (but is not the case in general). Nevertheless, compositional modelling is valuable for the understanding of the structure of the system. It allows, for example, to modify a part of a system while keeping the analysis done for the rest of the system, as we show in Example 31.

Technically, we proceed as follows. We introduce *monoid network games* (Definition 7) that make common structure of all of our motivating examples explicit, and that we believe cover the majority of network games studied in the literature. Roughly speaking, monoid network games are parametrised wrt *(i)* a monoid that aggregates information from neighbours and *(ii)* functions that govern how that information is propagated in the network. While we are able to model all network games, the structure of monoid network games allows us to characterise them as functors in a generic fashion.

Our category of open graphs **Grph** (Definition 18) is an extension of the approach of [7], from undirected graphs to undirected *multi*graphs. Multigraphs allow us to model games on networks where some links are stronger than others, cf. Example 30. Our **Grph** is different from other notions of "open graph" in the literature, e.g. via cospans [9], in that it is centred on the use of *adjacency matrices*, which are commonly used in graph theory to encode connectivity. Adjacency matrices give an explicit presentation of the graphs that allows an explicit description of the games played on them. Moreover, the emphasis on the matrix algebra means that **Grph** has the structure of commutative bialgebra—equivalent to the algebra of ordinary $\mathbb{N}$ matrices [15, 24]—but also additional structure that captures the algebraic content of adjacency matrices. Given that **Grph** has a presentation in terms of generators and equations, to obtain (2) it suffices to define it on the generators and check that **Grph**-equations are respected in **Game**. This is our main result, Theorem 27.

In addition to the presentation of **Grph** in terms of generators and equations, we characterise it as another category (Theorem 23) that makes clear its status as a category of "open graphs". The result can be understood as a kind of normal form for the morphisms of **Grph**, useful to describe concrete instantiations of $F_{\mathcal{N}}$ for arbitrary open graphs (Theorem 29).

Our work is a first step towards a more principled way of defining games parametrised by graphs. We would like to remark that the methodology that we present to define games on networks is more general than the particular instance worked out in this paper. Indeed, future work will extend both the notions of graphs (e.g. by considering directed graphs), as well as the kinds of games played on them (e.g. stochastic games, repeated games). While we do identify some applications, we believe that compositional reasoning is severely under-rated in traditional game theory, and that its adoption will lead to both more flexible modelling frameworks, as well as more scalable mathematical analyses.

Structure of the paper

We introduce our running examples in §2 and unify them under the umbrella of monoid network games. Next, we recall the basics of open games in §3 and identify the building blocks needed for (2). In §4 we introduce the category **Grph** of open, undirected multigraphs, and give a combinatorial characterisation, which is useful in applications. The construction of $F_{\mathcal{N}}$ is in §5, and several applications of our compositional framework are given in §6.

2 Network games

In this section we introduce motivating examples for our compositional framework and introduce a notion of game called the *monoid network game* that unifies them.

Network games [5, 14] are parametric wrt a network topology, usually represented by a graph. Players are the vertices, and the possible connections between the players are represented by the edges. Moreover, each player's payoff is affected only by the choices of its immediate *neighbours* on the graph. We use undirected multigraphs to model network topologies.

▶ **Definition 1.** *An undirected multigraph is $G = (V_G, E_G)$, where V_G is the set of vertices and E_G is a sym. multi-relation on V_G: a function $E_G \colon V_G \times V_G \to \mathbb{N}$ st $E_G(v_i, v_j) = E_G(v_j, v_i)$.*

A common way of capturing the connectivity of a graph is via *adjacency matrices*, which play an important role in graph theory. They are also crucial for our compositional account.

Assuming an ordering on the set of vertices of a graph, square matrices A with entries from $\mathbb{N}$ can record connections between vertex i and j in A_{ij}: a 0-entry signifies no edge, and non-zero entries count the connections. Ordinary matrices are too concrete to uniquely represent connectivity since edges between i and j can be recorded in the (i, j)th entry or the (j, i)th entry. One could use symmetric matrices or triangular matrices. For us, it is better to equate matrices that encode the same connectivity: $A \sim A'$ iff $A + A^T = A' + A'^T$.

▶ **Definition 2.** *An* adjacency matrix *is an equivalence class $[A]$ of matrices with entries in the natural numbers. The equivalence relation is given by*

$$A \sim A' \iff A + A^T = A' + A'^T.$$

A finite multigraph can also be defined as $(k_G, [A])$ where $k_G \in \mathbb{N}$ and $[A]$ a $k_G \times k_G$ adjacency matrix. Let $\mathbf{G}(n)$ be the set of multigraphs with n vertices, enumerated as $v_1, \ldots, v_n$.

▶ **Definition 3** (Network game). *An n-player* network game $\mathcal{N}$ *consists of, for each player $1 \leq i \leq n$, a set of choices X_i and a payoff $u_i : \mathbf{G}(n) \times \prod_{j=1}^{n} X_j \to \mathbb{R}$, such that each player's payoff is affected only by its own and its neighbours' choices: for each $G \in \mathbf{G}(n)$, each player i, each $j \neq i$ such that $(v_i, v_j) \notin E_G$, each $\underline{x}_{-j} \in \prod_{k \neq j}^{n} X_k$, and each $x_j, x'_j \in X_j$*

$$u_i(G, x_j, \underline{x}_{-j}) = u_i(G, x'_j, \underline{x}_{-j})$$

(The notation x_{-j}, standard in game theory, means a tuple with the jth element missing.)

The set of strategies *is $\prod_{i=1}^{n} X_i$ and its elements $\underline{x} \in \prod_{i=1}^{n} X_i$ are* strategy profiles.

The best response, *for a graph $G \in \mathbf{G}(n)$, is a relation $\mathbb{B}_{\mathcal{N}}$ on the set of strategies, defined by*

$$(\underline{x}, \underline{x}') \in \mathbb{B}_{\mathcal{N}} \Leftrightarrow \forall 1 \leq i \leq n.\ \forall y_i \in X_i.\ u_i(G, \underline{x}[i \mapsto x'_i]) \geq u_i(G, \underline{x}[i \mapsto y_i])$$

A Nash equilibrium, *for $G \in \mathbf{G}(n)$, is a strategy profile $\underline{x}$ s.t. for each player $1 \leq i \leq n$, $u_i(G, \underline{x}) \geq u_i(G, \underline{x}[i \mapsto x'_i])$ for each $x'_i \in X_i$. It is a fix-point of the best response relation.*

We now recall three important examples of network games.

▶ **Example 4** (Majority game). Each player has two choices, $X_i = \{Y, N\}$. A player receives a utility of 1 if its choice is the majority choice of its neighbours, and 0 otherwise, i.e.

$$u_i(G, \underline{x}) = \begin{cases} 1 & \text{if } |\{v_j \mid (v_i, v_j) \in E_G \text{ and } x_i = x_j\}| \geq |\{v_j \mid (v_i, v_j) \in E_G \text{ and } x_i \neq x_j\}| \\ 0 & \text{otherwise.} \end{cases}$$

Nash equilibria are strategy profiles where players take the majority choice of their neighbours.

▶ **Example 5** (Best-shot public goods game). Each player has two choices, $X_i = \{Y, N\}$, interpreted as investing or not investing in a public good. The investor bears a cost $0 < c < 1$, and gives a utility of 1 to themselves and every neighbour. The players are already partially satisfied with the current situation and assign a utility of $1 - c + \epsilon$, with $0 < \epsilon < c$, to the situation where neither the player nor its neighbours invest. The utility functions thus are:

$$u_i(G, \underline{x}) = \begin{cases} 1 - c & \text{if } x_i = Y \\ 1 & \text{if } x_i = N \text{ and } x_j = Y \text{ for some } (v_i, v_j) \in E_G \\ 1 - c + \epsilon & \text{otherwise.} \end{cases}$$

The Nash equilibrium is when no player invests, an example of a 'tragedy of the commons'.

▶ **Example 6** (Weakest-link public goods game). Each player's choice is an investment, valued in $\mathbb{R}_+$. The cost to the player given by an increasing cost function $c : \mathbb{R}_+ \to \mathbb{R}_+$ where $c(0) = 0$, and utility is the *minimum* level of investment of the player and all neighbours:

$$u_i(G, \underline{x}) = \min_{j=i \text{ or } (v_i, v_j) \in E_G} x_j - c(x_i).$$

A necessary condition for Nash equilibrium is that no player invests more than its neighbours.

In Examples 4, 5 and 6 every player has the same set of choices, and the utility depends in a uniform way on neighbours' choices. We collect these, and other examples in the literature, under the umbrella of *monoid network games.* Most examples in the literature can be collected in two classes [5, ch. 5], namely games on networks with constrained continuous actions or with binary actions. Provided that weights are natural numbers, the latter can be expressed as monoid network games. To express the former as monoid network games, we need to additionally ask that the parameters appearing in the utility functions of the players be constant. However, we can still express games of this class with different parameters for different players by composing different monoid network games. This is shown in Example 32.

Network games that do not fall into this category can be nevertheless expressed in a compositional way as illustrated in Fig. 1. If a game can be described in the form of a a *monoid* network game, we can say more: such games are a monoidal functor from the category of syntax to the category of semantics. The details are in Section 5.

To the best of our knowledge, the following has not previously appeared in the literature.

▶ **Definition 7** (Monoid network game). *A* monoid network game *is* $\mathcal{N} = (X, M, f, g)$ *where:*

- X *is the set of choices for each player*
- $M = (M, \oplus, e)$ *is a commutative monoid*
- $f : X \to M$ *and* $g : X \times M \to \mathbb{R}$ *are functions such that each utility function has the form*

$$u_i(G, \underline{x}) = g\left(x_i, \bigoplus_{(v_i,v_j)\in E_G} f(x_j)\right).$$

Examples 4, 5, 6 are indeed examples of monoid network games:

- The majority game (Example 4) has the monoid $(\mathbb{N}, +, 0) \times (\mathbb{N}, +, 0)$, counting the Y and N 'votes'. Define $f : \{Y, N\} \to \mathbb{N}^2$ by $f(Y) = (1, 0)$ and $f(N) = (0, 1)$, and $g : \{Y, N\} \times \mathbb{N}^2 \to \mathbb{R}$ is:

$$g(x, (n_1, n_2)) = \begin{cases} 1 & \text{if } x = Y \text{ and } n_1 \geq n_2 \\ 1 & \text{if } x = N \text{ and } n_1 \leq n_2 \\ 0 & \text{otherwise.} \end{cases}$$

- The best-shot public goods game (Example 5) is a monoid network game with the monoid $\mathbf{Bool} = (\{Y, N\}, \vee, N)$, where $\vee$ is logical or, $f : \mathbf{Bool} \to \mathbf{Bool}$ is the identity, and $g : \mathbf{Bool} \times \mathbf{Bool} \to \mathbb{R}$:

$$g(x, y) = \begin{cases} 1 - c & \text{if } x = Y \\ 1 & \text{if } x = N \text{ and } y = Y \\ 1 - c + \epsilon & \text{if } x = N \text{ and } y = N \end{cases}$$

- The weakest-link public goods game (Example 6) has the monoid $\mathbb{R}_+^\infty = (\{\mathbb{R}_+ \cup \{\infty\}, \min, \infty)$, f the embedding $\mathbb{R}_+ \hookrightarrow \mathbb{R}_+^\infty$, and $g : \mathbb{R}_+ \times \mathbb{R}_+^\infty \to \mathbb{R}$ is $g(x, y) = \min(x, y) - c(x)$.

3 Open games

Open games were introduced in [13] as a compositional approach to game theory.

▶ **Definition 8** (Open game). *Let* X, Y, R, S, Σ *be sets. An open game* $\mathcal{G}\colon \binom{X}{S} \overset{\Sigma}{\nrightarrow} \binom{Y}{R}$ *has:*

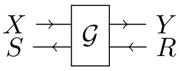
(i) $\mathbb{P}_{\mathcal{G}} \colon \Sigma \times X \to Y$*, called play function*

(ii) $\mathbb{C}_{\mathcal{G}} \colon \Sigma \times X \times R \to S$*, called coplay function*

(iii) $\mathbb{B}_{\mathcal{G}} \colon X \times (Y \to R) \to \mathcal{P}(\Sigma^2)$*, called best response function.*

Roughly speaking, an open game is a process that *(i)* given a *strategy* and *observation*, decides a *move*, and *(ii)* given a strategy, observation, and a *utility*, returns a *coutility* to the environment. Coutility is not a concept of classical game theory, but it enables compositionality by incorporating the fact that players reason about the future consequences of their actions. Finally, *(iii)*, the best response function, which, given a context for the game returns a relation on the set of strategies. A strategy σ is related to another strategy σ' if the latter is a best response to the former.

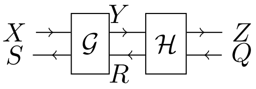
An open game is a process that receives observations (X) *"from the past"*, and the utility (R) *"from the future"*. It outputs moves (Y) covariantly and coutility (S) contravariantly.

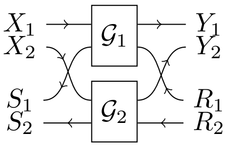


Open games are morphisms in a symmetric monoidal category **Game**. In order to formally define composition and monoidal product of games, it is useful to rephrase the definition in terms of lenses [19]. The detailed definitions are given in [13].

▶ **Definition 9 (Game).** **Game** *is the symmetric monoidal category with pairs of sets* $\left(\begin{smallmatrix} X \\ S \end{smallmatrix}\right)$ *as objects and (equivalence classes of) open games* $\mathcal{G}\colon \left(\begin{smallmatrix} X \\ S \end{smallmatrix}\right) \overset{\Sigma}{\nrightarrow} \left(\begin{smallmatrix} Y \\ R \end{smallmatrix}\right)$ *as morphisms.*

We give some intuitions. Composition, shown below left, is sequential play: $\mathcal{H} \cdot \mathcal{G}$ is thought of as $\mathcal{H}$ happening after $\mathcal{G}$, observing the moves of $\mathcal{G}$ and feeding back its coutility as $\mathcal{G}$'s utility. The monoidal product of open games represents two games played independently. The games are placed side by side with no connections, as shown below right.





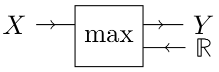
Classical games are *scalars* in **Game**, i.e. open games $\left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right) \nrightarrow \left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$. The fix-points of the best response functions of scalars in **Game** are the Nash equilibria of the games they represent.

Next we define specific open games used in our compositional account of network games. The first is the Utility Maximising Player, modelling typical players of classical game theory.

▶ **Definition 10** (Utility Maximising Player). *Let X and Y be sets and* $\mathsf{argmax}\colon \mathbb{R}^Y \to \mathcal{P}(Y)$ *take a function $\kappa : Y \to \mathbb{R}$ to the subset of Y where κ is maximised. Define $\mathcal{D}$ to be:*

$$\mathcal{D}\colon\ \left(\begin{smallmatrix} X \\ 1 \end{smallmatrix}\right) \overset{Y^X}{\nrightarrow} \left(\begin{smallmatrix} Y \\ \mathbb{R} \end{smallmatrix}\right)$$

$$\begin{cases} \mathbb{P}_{\mathcal{D}}(f,x) = f(x) \\ \mathbb{C}_{\mathcal{D}}(f,x,r) = * \\ \mathbb{B}_{\mathcal{D}}(x,\kappa) = \{(y,y') \in Y^X \times Y^X : y'(x) \in \mathsf{argmax}(\kappa)\} \end{cases}$$

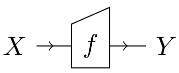


The category of sets and functions **Set** embeds into **Game** in two ways. In our account of network games, these embeddings encode how neighbours influence each other's utilities.

▶ **Definition 11.** *Let X, Y be sets and $f\colon X \to Y$ a function. Its covariant lifting is defined:*

$$f^*\colon\ \left(\begin{smallmatrix} X \\ 1 \end{smallmatrix}\right) \overset{1}{\nrightarrow} \left(\begin{smallmatrix} Y \\ 1 \end{smallmatrix}\right)$$

$$\begin{cases} \mathbb{P}_{f^*}(*,x) = f(x) \\ \mathbb{C}_{f^*}(*,x,*) = * \\ \mathbb{B}_{f^*}(x,*) = \{(*,*)\} \end{cases}$$



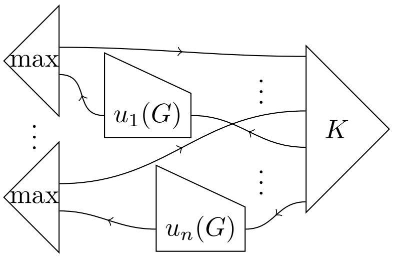
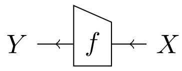


Figure 1 Open game representing a network game $\mathcal{N}$ played on a multigraph G

Similarly, its contravariant lifting is the following:

$$f_*\colon \left(\begin{smallmatrix}1\\Y\end{smallmatrix}\right) \overset{1}{\nrightarrow} \left(\begin{smallmatrix}1\\X\end{smallmatrix}\right)$$

$$\begin{cases}\mathbb{P}_{f_*}(*,*) = * \\ \mathbb{C}_{f_*}(*,*,x) = f(x) \\ \mathbb{B}_{f_*}(*,x) = \{(*,*)\}\end{cases}$$



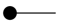
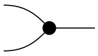
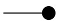
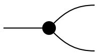
To obtain Examples 4, 5 and 6 as scalars in **Game**, players are taken to be utility-maximising players. The connectivity of the multigraph G determines their utility functions as contravariant liftings $u_i(G)$, while the context K sends back the choices of all players:

$$K\colon \left(\begin{smallmatrix}X^n\\X^n\times\cdots\times X^n\end{smallmatrix}\right) \overset{1}{\nrightarrow} \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$$
$$\mathbb{C}_K(\underline{x}) = (\underline{x},\ldots,\underline{x}).$$

The respective games are then obtained as the composition illustrated in Fig. 1. In this way, we obtain a compositional description of any network game. If a game can be described in the form of a a *monoid* network game, we can say more: such games are a monoidal functor from **Grph**, defined in the next section, to **Game**. The details are in Section 5.

4 Open graphs

We extend the compositional approach to graph theory of [7] from simple graphs to undirected multigraphs, identifying a "syntax" of network games as the arrows of a prop[1] **Grph**, generated from a monoidal signature and equations. We also provide a characterisation of **Grph** that explains its arrows as "open graphs". Differently from other approaches [3, 9], **Grph** uses adjacency matrices (Definition 2). Indeed, the presentation includes generators

 $: 0 \to 1,$  $: 2 \to 1,$  $: 1 \to 0,$  $: 1 \to 2$ (BIALG)

and the equations of Fig. 2. The prop **B** generated by this data is isomorphic [15, 24] to the prop of matrices with entries from $\mathbb{N}$, with composition being matrix multiplication. To convert between the two, think of the matrix as recording the numbers of paths: indeed, the (i, j)th entry in the matrix is the number of paths from the ith left port to the jth right port.

[1] A prop [17, 15] is a symmetric strict monoidal category where the objects are $\mathbb{N}$, and $m \otimes n := m + n$.

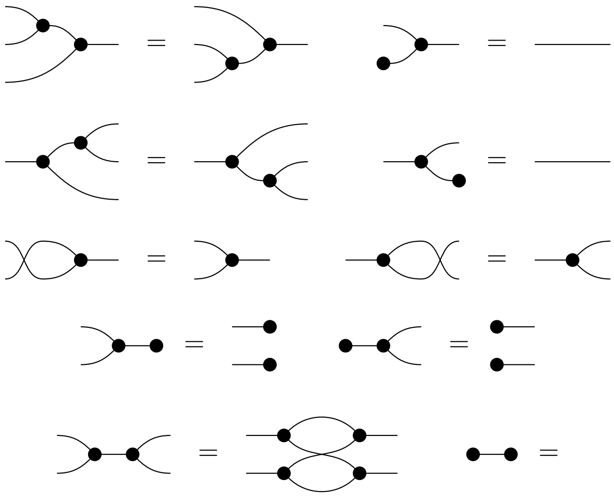


Figure 2 Commutative bialgebra equations, yielding prop **B**.

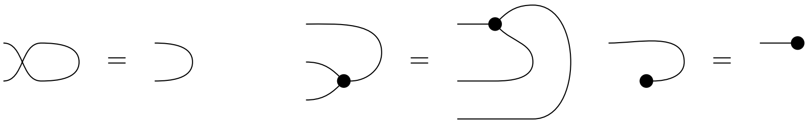
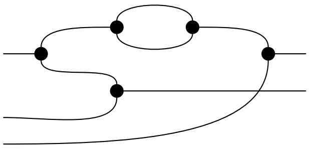
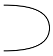


Figure 3 Equations of ∪, which together with the equations of Fig. 2 yield prop **BU**.

▶ **Example 12.** The following string diagram in **B** corresponds to the 3 × 2 matrix $\begin{pmatrix} 2 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$.



Next, we add a "cup" generator denoted

 $: 2 \to 0$ (U)

with its equations given in Fig. 3.

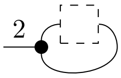
▶ **Definition 13.** *Let* **BU** *be the prop obtained from* (BIALG) *and* (U)*, quotiented by equations in Figs. 2 and 3, where the empty diagram is the identity on the monoidal unit.*

Just as **B** captures ordinary matrices, **BU** captures adjacency matrices:

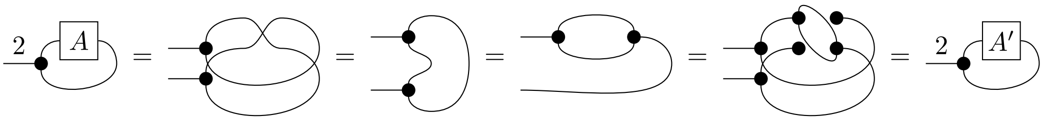
▶ **Proposition 14.** *For* $n \in \mathbb{N}$*, the hom-set* $[n, 0]$ *of* **BU** *is in bijection with* $n \times n$ *adjacency matrices, in the sense of Definition 2.*

We have seen that the relationship between matrices and diagrams in **B** is that the former encode the path information from the latter. Thus an $m \times n$ matrix is a diagram from m to n. Adding the cup and the additional equations means that, in general, a diagram from n to 0 in **BU** "encapsulates" an $n \times n$ matrix that expresses connectivity information in a similar way to adjacency matrices. We now give a concrete derivation to demonstrate this.

▶ **Example 15.** The equivalence relation of adjacency matrices is captured by the equations of Fig. 3. Consider matrices $A = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right) \sim \left(\begin{smallmatrix} 0 & 2 \\ 0 & 0 \end{smallmatrix}\right) = A'$. The morphism in **BU** is obtained by constructing their diagram in **B** as in Example 12 and "plugging" them in the following.

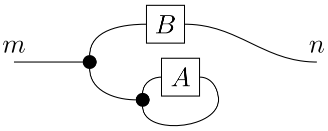


As shown below, the two diagrams obtained are equated by the axioms of **BU**.



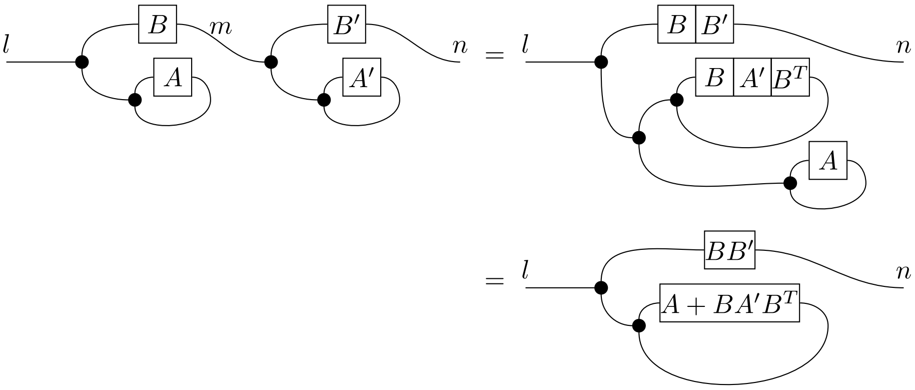
The prop **BU** can be given a straightforward combinatorial characterisation as the prop **Adj**.

▶ **Definition 16 (Adj).** *A morphism $\alpha\colon m \to n$ in the prop* **Adj** *[7] is a pair $(B, [A])$, where $B \in \mathbf{Mat}_{\mathbb{N}}(m, n)$ is a matrix, while $[A]$, with $A \in \mathbf{Mat}_{\mathbb{N}}(m, m)$, is an adjacency matrix. The components of* **Adj** *morphisms can be read off a "normal form" for* **BU** *arrows, as follows.*



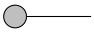
Composition in **Adj** becomes intuitive when visualised with string diagrams.

$(B, [A]) \circ (B', [A']) = (BB', [A + BA'B^T])$



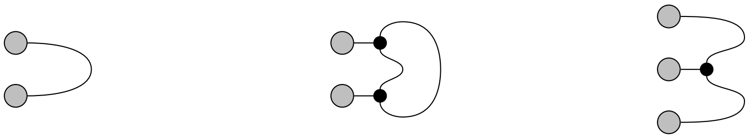
▶ **Proposition 17.** **BU** *is isomorphic to the prop* **Adj**. ◀

The proof is similar to the case for $\mathbb{Z}_2$ [7]. An extension of **BU** with just one additional generator and no additional equations yields the prop **Grph** of central interest for us.

▶ **Definition 18.** *The prop* **Grph** *is obtained from the generators in* (BIALG) *and* (U) *together with a generator*  $: 0 \to 1$. *The equations are those of Figs. 2 and 3.*

As we shall see, arrows $0 \to 0$ in **Grph** are precisely finite undirected multigraphs taken up to isomorphism: the additional generator plays the role of a graph vertex.

▶ **Example 19.** For example, the first of the following represents a multigraph with two vertices, connected by a single edge. The second one, two vertices connected by two edges. The third one, is a multigraph with three vertices and two edges between them.



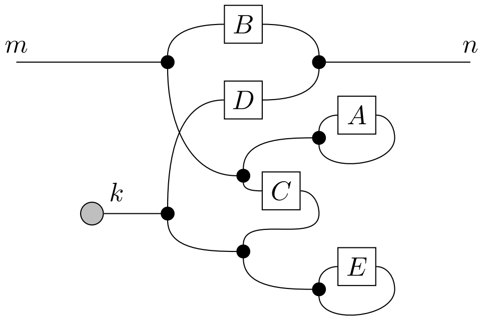
While the arrows $[0,0]$ are (iso classes of) multigraphs, general arrows can be understood as open graphs. Roughly speaking, they are graphs together with interfaces, and data that specifies the connectivity of the graph to its interfaces. We make this explicit below. Indeed, we shall see (Theorem 23) that the prop $\mathcal{A}$, defined below, is isomorphic to **Grph** – for this reason we use **Grph** string diagrams to illustrate its structure.

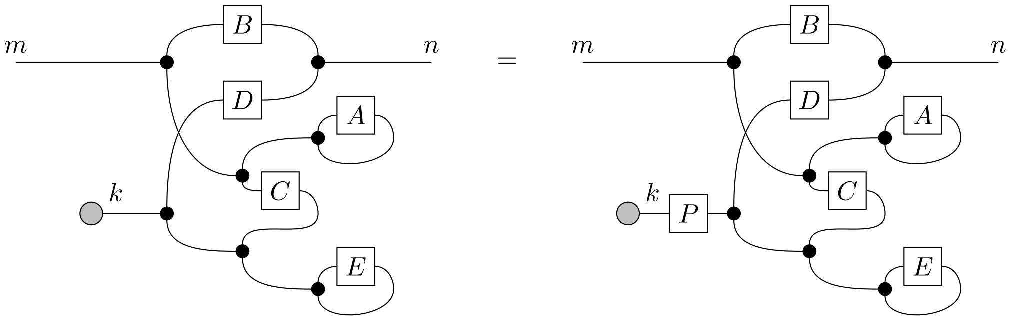
▶ **Definition 20** (The prop $\mathcal{A}$)**.** *A morphism* $\Gamma \colon m \to n$ *in the prop* $\mathcal{A}$ *is defined by*

$$\Gamma = (k, [A], B, C, D, [E]) \tag{3}$$

where $k \in \mathbb{N}$*,* $A \in \mathbf{Mat}_{\mathbb{N}}(m,m)$*,* $B \in \mathbf{Mat}_{\mathbb{N}}(m,n)$*,* $C \in \mathbf{Mat}_{\mathbb{N}}(m,k)$*,* $D \in \mathbf{Mat}_{\mathbb{N}}(k,n)$ *and* $E \in \mathbf{Mat}_{\mathbb{N}}(k,k)$*. Similarly to* **Adj** *(Definition 16), the components of* (3) *can be read off a "normal form" for arrows of* **Grph***, as visualised below right.*

Tuples (3) *are taken up to an equivalence relation that captures the fact that the order of the vertices is immaterial. Let* $\Gamma \sim \Gamma'$ *iff they are morphisms of the same type,* $\Gamma, \Gamma' \colon m \to n$ *with* k *vertices, and there is a permutation matrix* $P \in \mathbf{Mat}(k,k)$ *such that* $\Gamma' = (k, [A], B, CP^T, PD, [PEP^T])$*. The justification for this equivalence is the equality of the following two string diagrams in* **Grph***, below (for the details, see Appendix A on page 18).*





It is worthwhile to give some intuition for the components of (3). The idea is that an arrow Γ specifies a multigraph $G = (k, [E])$, and:

- B specifies connections between the two boundaries, bypassing G
- C specifies connections between the left boundary and G
- D specifies connections between G and the right boundary
- A specifies connections between the interfaces on the left boundary. This allows Γ to introduce connections between the vertices of an "earlier" open graph Δ. See Example 21.

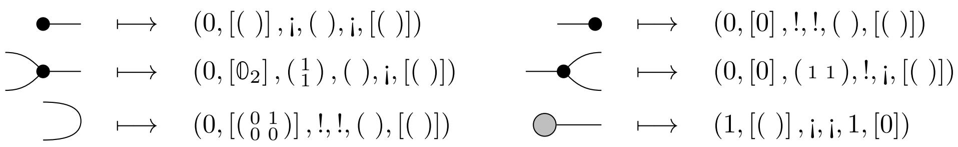
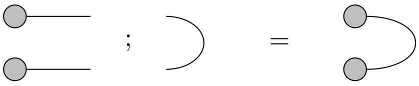

$\mapsto$ $(0, [(\,)], ¡, (\,), ¡, [(\,)])$
$\mapsto$ $(0, [\mathbb{0}_2], \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right), (\,), ¡, [(\,)])$
$\mapsto$ $(0, [\left(\begin{smallmatrix}0&1\\0&0\end{smallmatrix}\right)], !, !, (\,), [(\,)])$
$\mapsto$ $(0, [0], !, !, (\,), [(\,)])$
$\mapsto$ $(0, [0], \left(\begin{smallmatrix}1&1\end{smallmatrix}\right), !, ¡, [(\,)])$
$\mapsto$ $(1, [(\,)], ¡, ¡, 1, [0])$

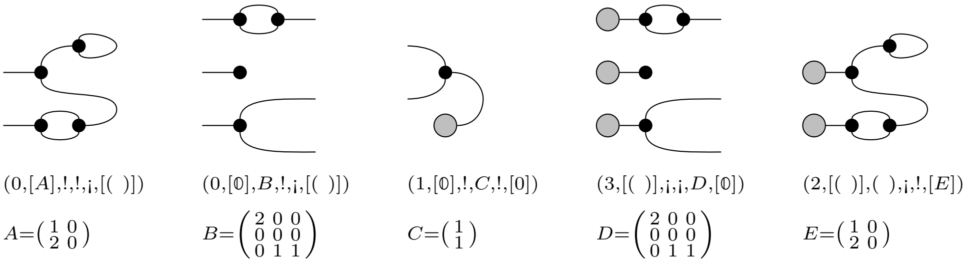
Figure 4 Image of θ on the generators

Defining composition in $\mathcal{A}$ is straightforward, given the above intuitions, but the details are rather tedious: see Lemma 33 in Appendix A.

▶ **Example 21.** In a composite $\Delta\ ;\Gamma$, Γ may introduce edges between the vertices of Δ. Indeed, the first diagram in Example 19 can be decomposed:



▶ **Example 22.** The following show the role of $\mathcal{A}$-morphism components, when isolated. The leftmost open graph has only left-side ports. It introduces a self-loop and two connections. The second has only connections between the left and right interfaces; the first left port is connected twice to the first right port, the second port is disconnected, and the third left port is connected to the second and third right ports. The third open graph has one vertex connected to the two left ports. The fourth has three vertices connected to the right ports, following the specification in the second. The rightmost (closed) multigraph has its vertices connected according to the specification of the leftmost vertex-less open graph. We write ! for matrices without columns, ¡ for matrices without rows and ( ) for the empty matrix.


$(0,[A],!,¡,[(\,)])$
$A=\left(\begin{smallmatrix}1&0\\2&0\end{smallmatrix}\right)$
$(0,[\mathbb{0}],B,!,¡,[(\,)])$
$B=\begin{pmatrix}2&0&0\\0&0&0\\0&1&1\end{pmatrix}$
$(1,[\mathbb{0}],!,C,!,[0])$
$C=\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$
$(3,[(\,)],¡,¡,D,[\mathbb{0}])$
$D=\begin{pmatrix}2&0&0\\0&0&0\\0&1&1\end{pmatrix}$
$(2,[(\,)],(\,),¡,!,[E])$
$E=\left(\begin{smallmatrix}1&0\\2&0\end{smallmatrix}\right)$

The main result in this section is the following.

▶ **Theorem 23.** *There is an isomorphism of props* $\theta\colon \mathbf{Grph} \to \mathcal{A}$.

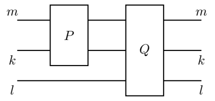
The remainder of this section builds a proof of the above, summarised in the diagram below.

$$\begin{tikzcd}
\mathbf{BU} \arrow[r] \arrow[d, "\cong\ (\text{Proposition 17})"'] & \mathbf{Grph} \cong \mathbf{BU} + \{\bigcirc\!\!-\} \arrow[d, dotted, "\theta"] & \{\bigcirc\!\!-\} \arrow[l] \arrow[d, "\cong\ (\text{Lemma 25})"'] \\
\mathbf{Adj} \arrow[r] & \mathcal{A} \cong \mathbf{Adj} + \mathbf{b}\mathbb{P}\ (\text{Proposition 26}) & \mathbf{b}\mathbb{P} \arrow[l]
\end{tikzcd}$$

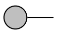
First, note that $\mathbf{Grph}$ is the coproduct $\mathbf{BU} + \{\bigcirc\!\!-\}$ in the category of props, where $\{\bigcirc\!\!-\}$ is the free prop on a single generator $0 \to 1$. Next, we characterise $\{\bigcirc\!\!-\}$ as $\mathbf{b}\mathbb{P}$, defined below, in Lemma 25. Given that $\mathbf{BU} \cong \mathbf{Adj}$, as shown in Proposition 17, to show the existence of θ it suffices to show that $\mathcal{A}$ satisfies the universal property of the coproduct $\mathbf{Adj} + \mathbf{b}\mathbb{P}$, which is Proposition 26. The action of θ on the generators of **Grph** is in Fig. 4.

▶ **Definition 24** ($\mathbf{b}\mathbb{P}$). *The prop of* bound permutations $\mathbf{b}\mathbb{P}$ *has as morphisms* $m \to m + k$ *pairs* $[(k, P)]$ *where* $k \in \mathbb{N}$ *and* $P \in \mathbf{Mat}_{\mathbb{N}}(m + k, m + k)$ *is a permutation matrix. Such pairs are identified to ensure that the order of the lower* k *rows of* P *is immaterial. Roughly speaking, considering* P *as a permutation of* $m+k$ *inputs to* $m+k$ *outputs, in* $[(k, P)]$ *the final* k *inputs are "bound". Explicitly,* $(k, P) \sim (k, P')$ *iff there is a permutation* $\sigma \in \mathbf{Mat}_{\mathbb{N}}(k, k)$ *st* $P = \left(\begin{smallmatrix} \mathbb{1}_m & \mathbb{0} \\ \mathbb{0} & \sigma \end{smallmatrix}\right) P'$. *Composition is defined:*

$$(l, Q) \circ (k, P) = (k + l, \left(\begin{smallmatrix} P & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_l \end{smallmatrix}\right) Q)$$



Identities are identity matrices $id_n = (0, \mathbb{1}_n)$. *The fact that* $\mathbf{b}\mathbb{P}$ *is a prop is Lemma 34 in Appendix A.*

▶ **Lemma 25.** $\mathbf{b}\mathbb{P}$ *is isomorphic to* $\Big\{$  $\Big\}$.

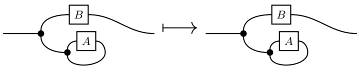
Proof. Let us call $\phi = (0, (1)) \colon 0 \to 1$, which is a morphism in $\mathbf{b}\mathbb{P}$. We show directly that, for any other prop $\mathbb{P}$ that contains a morphism $v \colon 0 \to 1$, there is a unique prop homomorphism $\alpha^{\#} \colon \mathbf{b}\mathbb{P} \to \mathbb{P}$ such that $\alpha^{\#}(\phi) = v$. The details are given as Lemma 35 in Appendix A. ◀

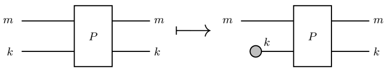
Given the results of Proposition 17 and Lemma 25, we obtain the isomorphism $\theta \colon \mathbf{Grph} \to \mathcal{A}$, thereby completing the proof of Theorem 23, by showing that:

▶ **Proposition 26.** $\mathcal{A}$ *satisfies the universal property of the coproduct* $\mathbf{Adj} + \mathbf{b}\mathbb{P}$.

Proof. In order to show that $\mathcal{A}$ is a coproduct $\mathbf{Adj} + \mathbf{b}\mathbb{P}$, we define the two inclusions.

$$\begin{aligned} i_1 \colon \mathbf{Adj} &\longrightarrow \mathcal{A} \\ n &\longmapsto n \\ (B, [A]) &\longmapsto (0, [A], B, !, ¡, [(\,)]) \end{aligned} \qquad \begin{aligned} i_2 \colon \mathbf{b}\mathbb{P} &\longrightarrow \mathcal{A} \\ n &\longmapsto n \\ (k, P) &\longmapsto (k, [\mathbb{0}_n], P_*^{[1,n]}, \mathbb{0}_{nk}, P_*^{[n+1,n+k]}, [\mathbb{0}_k]) \end{aligned}$$





We indicate with $P_*^{[1,n]}$ the first n rows of the matrix P and, similarly, with $P_*^{[n+1,n+k]}$ the rows between the $n+1$-th and the $n+k$-th. It is not difficult to show that these are indeed homomorphism, the details are given as Claim 36 in Appendix A.

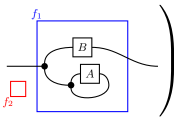
Now, we show that, for any other prop $\mathcal{C}$ with prop homomorphisms $\mathbf{Adj} \xrightarrow{f_1} \mathcal{C} \xleftarrow{f_2} \mathbf{b}\mathbb{P}$, there exists a unique prop homomorphism $H \colon \mathcal{A} \to \mathcal{C}$ such that $H \circ i_1 = f_1$ and $H \circ i_2 = f_2$.

$$\begin{aligned} H \colon \mathcal{A} &\longrightarrow \mathcal{C} \\ n &\longmapsto n \\ (k, [A], B, C, D, [E]) &\longmapsto f_1\left(\left(\begin{smallmatrix} B \\ D \end{smallmatrix}\right), \left[\left(\begin{smallmatrix} A & C \\ \mathbb{0} & E \end{smallmatrix}\right)\right]\right) \circ (\mathbb{1}_m \otimes f_2(k, \mathbb{1}_k)) \end{aligned}$$

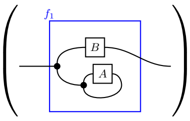
We verify that H is a homomorphism in Lemma 37 in Appendix A. Next, we confirm that $H \circ i_1 = f_1$ and $H \circ i_2 = f_2$, where two functor boxes [8] for f_1 and f_2 are coloured:

$$H \circ i_1(B,[A]) = H(0,[A], B, !, ¡, [(\ )])$$

$$= f_1(B,[A]) \circ (\mathbb{1}_m \otimes f_2(0,\mathbb{1}_0)) \Bigg($$



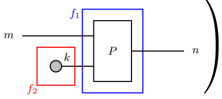
$$\Bigg) = f_1(B,[A]) \Bigg($$



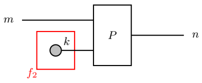
$$\Bigg)$$

$$H \circ i_2(k,P) = H(k,[\mathbb{0}_n], P_*^{[1,n]}, \mathbb{0}_{nk}, P_*^{[n+1,n+k]}, [\mathbb{0}_k])$$

$$= f_1(P,[\mathbb{0}_{n+k}]) \circ (\mathbb{1}_n \otimes f_2(k,\mathbb{1}_k)) \Bigg($$

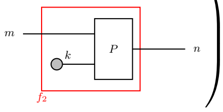


$$\Bigg) = P \circ f_2(k,\mathbb{1}_{n+k}) \Bigg($$



$$\Bigg)$$

$$= f_2(k,P) \Bigg($$



$$\Bigg)$$

Moreover, H is the unique prop homomorphism with these properties. In fact, suppose there is $H' \colon \mathcal{A} \to C$ such that $H' \circ i_1 = f_1$ and $H' \circ i_2 = f_2$. Then:

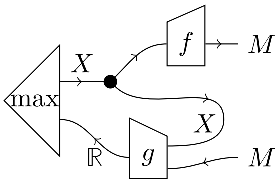
$$\begin{aligned} H'(k,[A], B, C, D, [E]) =& H'(i_1((\begin{smallmatrix} B \\ D \end{smallmatrix}), [(\begin{smallmatrix} A & C \\ \mathbb{0} & E \end{smallmatrix})]) \circ (\mathbb{1}_m \otimes i_2(k,\mathbb{1}_k))) \\ =& H'i_1((\begin{smallmatrix} B \\ D \end{smallmatrix}), [(\begin{smallmatrix} A & C \\ \mathbb{0} & E \end{smallmatrix})]) \circ (H'(\mathbb{1}_m) \otimes H'i_2(k,\mathbb{1}_k)) \\ =& f_1((\begin{smallmatrix} B \\ D \end{smallmatrix}), [(\begin{smallmatrix} A & C \\ \mathbb{0} & E \end{smallmatrix})]) \circ (\mathbb{1}_m \otimes f_2(k,\mathbb{1}_k)) = H(k,[A], B, C, D, [E]). \blacktriangleleft \end{aligned}$$

5 Games on graphs via functorial semantics

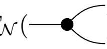
Here we show that monoid network games $\mathcal{N}$ define monoidal functors $F_{\mathcal{N}} \colon \mathbf{Grph} \to \mathbf{Game}$, which is our main contribution. To every open graph Γ, $F_{\mathcal{N}}$ associates an open game, where $\mathcal{N}$ is played on Γ. We give an explicit account of the $F_{\mathcal{N}}$-image of open graphs Γ, using Theorem 23. We also explain how $F_{\mathcal{N}}$ acts on closed graphs, giving classical games.

Since **Grph** is given by generators and equations, it suffices to define $F_{\mathcal{N}}$ on the generators and show that the equations are respected. Fix a monoid network game $\mathcal{N} = (X, M, f, g)$.

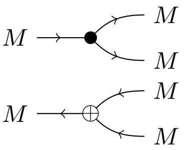
- On objects, $F_{\mathcal{N}}(1) = (\begin{smallmatrix} M \\ M \end{smallmatrix})$. Thus, for $n \in \mathbf{Grph}$, we have $F_{\mathcal{N}}(n) = (\begin{smallmatrix} M^n \\ M^n \end{smallmatrix})$
- The vertex ○—— $: 0 \to 1$ is mapped to the open game $F_{\mathcal{N}}($○——$) \colon (\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}) \xrightarrow{X} (\begin{smallmatrix} M \\ M \end{smallmatrix})$ defined
  - $\Sigma_{F_{\mathcal{N}}(\circ\!-)} = X$
  - $\mathbb{P}_{F_{\mathcal{N}}(\circ\!-)}(x_i, *) = f(x_i)$
  - $\mathbb{C}_{F_{\mathcal{N}}(\circ\!-)}(x_i, *, m) = *$
  - $(x_i, x_i') \in \mathbb{B}_{F_{\mathcal{N}}(\circ\!-)}(*, \kappa : M \to M)$
    iff $x_i' \in \arg\max\limits_{x_i'':X} g(x_i'', \kappa(f(x_i'')))$

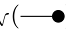


- The generators (BIALG) are mapped to the bialgebra structure on (M, M) induced by the monoid action of M. Specifically, they are:

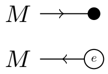
$F_{\mathcal{N}}($$)\colon \binom{M}{M} \overset{1}{\nrightarrow} \binom{M^2}{M^2}$

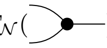
$$\begin{cases}\mathbb{P}(*,m)=(m,m)\\ \mathbb{C}(*,m_1,m_2,m_3)=m_2\oplus m_3\end{cases}$$



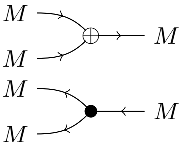
$F_{\mathcal{N}}($$)\colon \binom{M}{M} \overset{1}{\nrightarrow} \binom{1}{1}$

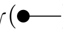
$$\begin{cases}\mathbb{P}(*,m)=*\\ \mathbb{C}(*,m,*)=e\end{cases}$$



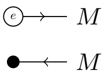
$F_{\mathcal{N}}($$)\colon \binom{M^2}{M^2} \overset{1}{\nrightarrow} \binom{M}{M}$

$$\begin{cases}\mathbb{P}(*,m_1,m_2)=m_1\oplus m_2\\ \mathbb{C}(*,m_1,m_2,m_3)=(m_1,m_1)\end{cases}$$



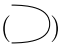
$F_{\mathcal{N}}($$)\colon \binom{1}{1} \overset{1}{\nrightarrow} \binom{M}{M}$

$$\begin{cases}\mathbb{P}(*,*)=e\\ \mathbb{C}(*,*,m)=*\end{cases}$$

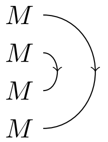


where each of these open games is built from lifted functions (Definition 11).

- $: 2 \to 0$ is mapped to the following open game (see [13])

$F_{\mathcal{N}}($$)\colon \binom{M^2}{M^2} \overset{1}{\nrightarrow} \binom{1}{1}$

$$\begin{cases}\mathbb{P}(*,m_1,m_2)=*\\ \mathbb{C}(*,m_1,m_2,*)=(m_2,m_1)\end{cases}$$



To prove that $F_{\mathcal{N}}$ is a symmetric monoidal functor it suffices to show that the equations of **Grph** are respected; this is a straightforward but somewhat lengthy computation.

▶ **Theorem 27.** *$F_{\mathcal{N}}$ defines a symmetric monoidal functor* **Grph** $\to$ **Game***.*

Proof. See Appendix B, on page 23. ◀

Note that $F_{\mathcal{N}}$ does *not* respect axioms (C1) or (C2) of [7], so it does not define a functor **ABUV** → **Game** in the terminology of *loc. cit.* This, together with the increased expressivity of multigraphs over simple graphs, motivates our extension from **ABUV** to **Grph**.

Theorem 23 gives a convenient "normal form" for the arrows of **Grph**, which we use to give an explicit description of the image of any (open) graph Γ under $F_{\mathcal{N}}$. First, we specialise to closed graphs that yield ordinary network games. This result—a sanity check for our compositional framework—is a corollary of the more general Theorem 29, proved subsequently.

▶ **Corollary 28.** *Let $\mathcal{N}=(X,M,f,g)$ be a monoid network game, and consider $\Gamma: 0\to 0$ in* **Grph***, an undirected multigraph with k vertices. Then the game $F_{\mathcal{N}}(\Gamma)\colon \binom{1}{1} \overset{X^k}{\nrightarrow} \binom{1}{1}$ has:*

- *$\Sigma_{F_{\mathcal{N}}(\Gamma)} = X^k$ as its strategy profiles,*
- *$\mathbb{B}_{F_{\mathcal{N}}(\Gamma)}(*,*) \subseteq X^k \times X^k$ is the best response relation of $\mathcal{N}$ played on Γ.*

Note that while the expressions in the statement of Theorem 29 below may seem involved, they are actually derived in an entirely principled, compositional manner from the generators of **Grph**. Indeed, the proof is by structural induction on the morphisms of **Grph**.

▶ **Theorem 29.** *Let $\mathcal{N}=(X,M,f,g)$ be a monoid network game. Let $\Gamma: i\to j$ be a morphism in* **Grph** *with k vertices st $\theta(\Gamma)=(k,[A],B,C,D,[E])$, where $A: i\times i$, $B: i\times j$, $C: i\times k$, $D: k\times j$ and $E: k\times k$. Then the open game $F_{\mathcal{N}}(\Gamma)\colon \binom{M^i}{M^i} \overset{X^k}{\nrightarrow} \binom{M^j}{M^j}$ has:*

- *set of strategy profiles $\Sigma(F_{\mathcal{N}}(\Gamma)) = X^k$*
- *play function $\mathbb{P}_{F_{\mathcal{N}}(\Gamma)}: X^k\times M^i \to M^j$ given by $\mathbb{P}_{F_{\mathcal{N}}(\Gamma)}(\underline{\sigma},\underline{x}) = B^T\underline{x}\oplus D^T f(\underline{\sigma})$*
- *coplay function $\mathbb{C}_{F_{\mathcal{N}}(\Gamma)}: X^k\times M^i\times M^j \to M^i$ is $\mathbb{C}_{F_{\mathcal{N}}(\Gamma)}(\underline{\sigma},\underline{x},\underline{r}) = (A+A^T)\underline{x}\oplus B\underline{r}\oplus Cf(\underline{\sigma})$*

- *best response relation* $\mathbb{B}_{F_{\mathcal{N}}(\Gamma)} : M^i \times (M^j \to M^j) \to \mathcal{P}(X^k \times X^k)$ *is* $(\underline{\sigma}, \underline{\sigma}') \in \mathbb{B}_{F_{\mathcal{N}}(\Gamma)}(\underline{x}, \kappa)$ *iff, for all* k*,*

$$\sigma'_k \in \operatorname*{argmax}_{s \in X} g\left(s, (C^T)^k_* \underline{x} \oplus D^k_* \kappa \left(B^T \underline{x} \oplus D^T f(\underline{\sigma}\,[k \mapsto s])\right) \oplus (E + E^T)^k_* f(\underline{\sigma}\,[k \mapsto s])\right)$$

Proof. See Appendix B on page 23. ◀

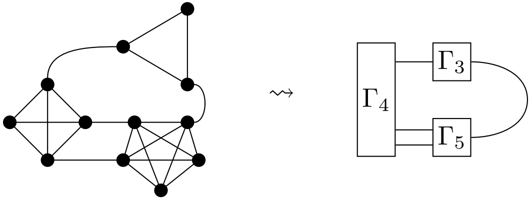
6 Examples

We return to examples: the majority (Example 4), the best-shot public goods (Example 5) and the weakest-link public goods (Example 6) games, and demonstrate various applications of our framework. We first show that to compute the Nash equilibrium of the majority game played on interconnected cliques is to calculate equilibria of its clique subgames.

▶ **Example 30** (Majority game). In the majority game the best response can be decomposed into the best responses of its components. Let $\mathcal{N}$ be the monoid network game for the majority game, defined on pg. 5, and consider a graph composed of N cliques, as follows:

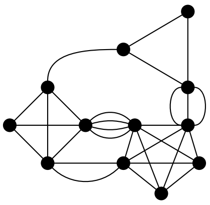
- each vertex of each clique can be connected to at most one vertex of another clique,
- in each clique there is at least one vertex not connected to any vertex outside its clique.

Such graphs decompose as N open graphs, each a clique with some boundary connections. We omit the details and give, instead, an illustrative example: below left is a picture of three connected cliques, while the schematic on the right is the corresponding expression in **Grph**.



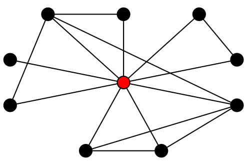
It is easy to show that the choice of each clique does not depend on the choices of other cliques. Indeed, the Nash equilibria of the majority game played on connected cliques in our sense are those strategy profiles where, in every clique, all players make the same choice. In particular, there are 2^N Nash equilibria.

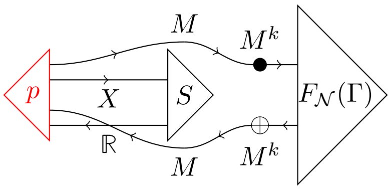
In some cases, players can take into account the choice of another player with a different intensity. This can be modelled by changing the number of edges between the vertices. Let us consider the above example with some of the vertices connected multiple times. This modification of the network—illustrated below—reflects in a modification of the equilibria, which are now strategy profiles in which every player takes the same choice.



In the best-shot public goods game (Example 5), the Nash equilibrium is when no player invests. In Example 31, we show how the compositional description is useful to adapt the model to a slightly different situation. We can imagine that one of the players now has access to incentives to invest in the public good. This scenario is represented by modifying the game and allowing one player to interact with the environment, which is the source of the incentives for this player. This modification "opens" the game to one of type $\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right) \to \left(\begin{smallmatrix}X\\\mathbb{R}\end{smallmatrix}\right)$: as a result, the Nash equilibrium changes. This is a simple model of a common economic situation, 'solving' a social dilemma by external intervention, for example by regulation [12].

▶ **Example 31** (Best-shot public goods game). Consider the best-shot public goods game played on a graph that contains a vertex connected to all other vertices. Removing the central vertex from this graph leaves an open graph that we will call Γ.





Here, $F_{\mathcal{N}}(\Gamma)$ is the best-shot public goods game played on the open graph Γ, p is the central player that has been substituted, and S is the external open game that influences p. The utility function of player p and the coplay function of S are as follows.

$$u_p(\Gamma, x) = \begin{cases} 1 - c + \delta & \text{if } x_p = 1 \\ 1 - \epsilon & \text{if } x_p = 0 \wedge \exists (p,j) \in E_\Gamma \; x_j = 1 \\ 1 - c & \text{if } \forall j \; x_j = 0 \end{cases}$$

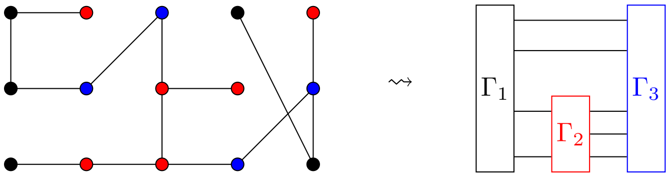
$$S\colon \left(\begin{smallmatrix} X \\ \mathbb{R} \end{smallmatrix}\right) \xrightarrow{1} \left(\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\right)$$

$$\mathbb{C}(*, x, *) = \begin{cases} \delta & \text{if } x = 1 \\ -\epsilon & \text{if } x = 0 \end{cases}$$

The addition of the open game S and the modification of player p modifies the Nash equilibrium to be the strategy profile where only the central player invests. The idea is that the “external” agent S incentivises the central player p to invest.

Our last example illustrates a common situation where the compositional description of a game does *not* allow a compositional analysis of the best response. However, in this case, compositionality can be used to obtain a variant of the weakest-link public goods game (Example 6) where different cost functions are used in different parts of the graph G. The desired game is obtained by composing such open games according to the structure of G.

▶ **Example 32** (Weakest-link public goods game). Consider the weakest-link public goods game played on a connected graph G. Suppose that players have different cost functions. We partition them according to their cost functions, and use this partition to decompose the G into an expression in **Grph**, as illustrated for a particular example below:



While the definition above uses our compositional techniques, the Nash equilibrium is calculated on the resulting closed game, and is a strategy profile where every player invests equally, with utility depending on individual cost functions. While it may be unsatisfying, this failure of Nash equilibria to be compositional can be seen as an inherent feature of game theory. In particular it is already present in the theory of open games; the passage from graphs to games is nevertheless fully compositional.

7 Conclusions

Our contribution is a compositional account of network games via strict monoidal functors. This adds a class of network games to the games that have been expressed in compositional game theory [13, 2]. Of independent interest is our work on the category **Grph**, extending [7]. This is an approach to “open graphs” that, as we have seen, is compatible with the structure of open games, and in future work we will identify other uses of this category.

We also intend to extend the class of open graphs to *directed* open graphs. The motivation for this is that, in some network games, interactions between players are *not* bidirectional. Consider, for example, a variant of the majority game where there is an "influencer": a player whose choice affects the choices of other players, but is not in turn conversely affected.

We will also extend the menagerie of games that can be played on a graph. We plan to study games with more generic utility functions, incomplete information, and repeated games. It could also prove interesting to study natural transformations between the functors that define games, and explore the game theoretical relevance of such transformations.

References

1 John C Baez and Brendan Fong. A compositional framework for passive linear networks, 2015. arXiv preprint: `https://arxiv.org/abs/1504.05625`.

2 Joe Bolt, Jules Hedges, and Philipp Zahn. Bayesian open games, 2019. arXiv preprint: `https://arxiv.org/abs/1910.03656`.

3 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 710–719. ACM, 2016. `doi:10.1145/2933575.2935316`.

4 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. The calculus of signal flow diagrams I: linear relations on streams. *Inf. Comput.*, 252:2–29, 2017. `doi:10.1016/j.ic.2016.03.002`.

5 Yann Bramoullé, Andrea Galeotti, and Brian Rogers. *The Oxford handbook of the economics of networks*. Oxford University Press, 2016.

6 Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari. A connector algebra for P/T nets interactions. In *Concurrency Theory (CONCUR '11)*, volume 6901 of *LNCS*, pages 312–326. Springer, 2011. `doi:10.1007/978-3-642-23217-6_21`.

7 Apiwat Chantawibul and Paweł Sobociński. Towards compositional graph theory. In *Proceedings of MFPS'15*, volume 319 of *ENTCS*, pages 121–136, 2015. `doi:10.1016/j.entcs.2015.12.009`.

8 J. R. B. Cockett and R. A. G. Seely. Linearly distributive functors. *Journal of Pure and Applied Algebra*, 143(1-3):155–203, 1999.

9 Brendan Fong. Decorated cospans. *Theory and Applications of Categories*, 30(33):1096–1120, 2015.

10 James H Fowler and Nicholas A Christakis. Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the framingham heart study. *BMJ*, 337, 2008. `doi:10.1136/bmj.a2338`.

11 Andrea Galeotti. Talking, searching and pricing. *International Economic Review*, 51(4):1159–1174, 2010. `doi:10.1111/j.1468-2354.2010.00614.x`.

12 Andrea Galeotti, Benjamin Golub, and Sanjeev Goyal. Targeting interventions in networks. Forthcoming in *Econometrica*, 2019.

13 Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. In *Proceedings of Logic in Computer Science (LiCS) 2018*. ACM, 2018. `doi:10.1145/3209108.3209165`.

14 Matthew Jackson and Yves Zenou. Games on networks. In *Handbook of game theory with economic applications*, volume 4, chapter 3, pages 95–163. Elsevier, 2015. `doi:10.1016/B978-0-444-53766-9.00003-3`.

15 Stephen Lack. Composing PROPs. *Theor. App. Categories*, 13(9):147–163, 2004.

16 Qiu Li, MingChu Li, Lin Lv, Cheng Guo, and Kun Lu. A new prediction model of infectious diseases with vaccination strategies based on evolutionary game theory. *Chaos, Solitons & Fractals*, 104:51–60, 2017. `doi:10.1016/j.chaos.2017.07.022`.

17 Saunders Mac Lane. Categorical algebra. *Bull. Amer. Math. Soc.*, 71:40–106, 1965.

18 Saunders Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag New York, 1978.

19 Frank Joseph Oles. *A Category-theoretic Approach to the Semantics of Programming Languages.* PhD thesis, Syracuse University, Syracuse, NY, USA, 1982. AAI8301650.

20 Alfred Tarski. The concept of truth in the languages of the deductive sciences. *Prace Towarzystwa Naukowego Warszawskiego, Wydzial III Nauk Matematyczno-Fizycznych*, 34(13-172):198, 1933.

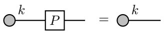
21 Alfred Tarski and Robert L Vaught. Arithmetical extensions of relational systems. *Compositio mathematica*, 13:81–102, 1957.

22 Giorgio Topa. Social Interactions, Local Spillovers and Unemployment. *The Review of Economic Studies*, 68(2):261–295, April 2001. doi:10.1111/1467-937X.00169.

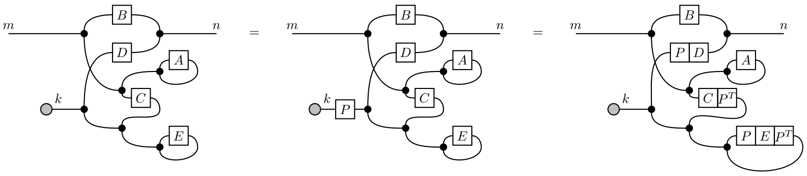
23 Glynn Winskel. *The formal semantics of programming languages: an introduction.* MIT press, 1993.

24 Fabio Zanasi. *Interacting Hopf Algebras: the theory of linear systems.* PhD thesis, École Normale Supérieure, 2015.

A Proofs for Section 4

Details for definition 20. By naturality of the symmetries, the vertex generators commute with any permutation matrix P:  .

Thus, we can show that $\Gamma = (k, [A], B, C, D, [E])$ and $\Gamma' = (k, [A], B, CP^T, PD, [PEP^T])$ represent the same open graph.

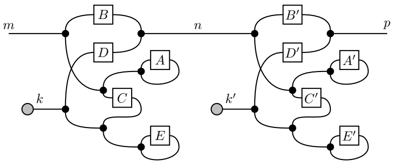


◀

▶ **Lemma 33.** *$\mathcal{A}$ is a prop.*

Proof. We start by proving that $\mathcal{A}$ is a category. The diagram below can be rewritten, using the axioms of **B**, as a diagram of the form shown in Definition 20. The components of the normal form obtained in this way give the algebraic definition of the composition.

$$\Gamma'\circ\Gamma = \left(k + k', \left[A + BA'B^T\right], BB', (C + B(A' + A'^T)D^T|BC'), \left(\begin{smallmatrix} DB' \\ D' \end{smallmatrix}\right), \left[\left(\begin{smallmatrix} E+DA'D^T & DC' \\ \mathbb{0} & E' \end{smallmatrix}\right)\right]\right)$$



Identities are defined in the obvious way: $\mathbb{1}_n = (0, [\mathbb{0}_n], \mathbb{1}_n, !, ¡, [(\ )])$.

The definition of composition is coherent with the equivalence classes because, whenever $\Gamma \sim \Gamma_0$ with matrix P and $\Gamma' \sim \Gamma'_0$ with matrix P', $\Gamma' \circ \Gamma \sim \Gamma'_0 \circ \Gamma_0$ with matrix $\left(\begin{smallmatrix} P & \mathbb{0} \\ \mathbb{0} & P' \end{smallmatrix}\right)$. Composition is associative because the matrices relative to the vertices are [ ]-equivalent. Clearly, composition is unital and we proved that $\mathcal{A}$ is a category. Now we prove that it is monoidal.

Lead by the interpretation of the matrices that define a morphism, we define monoidal product as follows.

$$\Gamma \otimes \Gamma' = \left(k+k', \left[\left(\begin{smallmatrix} A & \mathbb{0} \\ \mathbb{0} & A' \end{smallmatrix}\right)\right], \left(\begin{smallmatrix} B & \mathbb{0} \\ \mathbb{0} & B' \end{smallmatrix}\right), \left(\begin{smallmatrix} C & \mathbb{0} \\ \mathbb{0} & C' \end{smallmatrix}\right), \left(\begin{smallmatrix} D & \mathbb{0} \\ \mathbb{0} & D' \end{smallmatrix}\right), \left[\left(\begin{smallmatrix} E & \mathbb{0} \\ \mathbb{0} & E' \end{smallmatrix}\right)\right]\right)$$

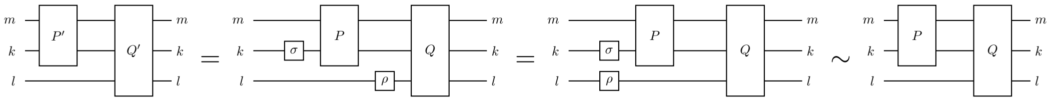
The monoidal unit is the empty diagram: $\mathbb{I} = (0, [(\ )], (\ ), (\ ), (\ ), [(\ )])$

The monoidal product is well-defined on equivalence classes because, whenever $\Gamma \sim \Gamma_0$ with matrix P and $\Gamma' \sim \Gamma'_0$ with matrix P', $\Gamma \otimes \Gamma' \sim \Gamma_0 \otimes \Gamma'_0$ with matrix $\left(\begin{smallmatrix} P & \mathbb{0} \\ \mathbb{0} & P' \end{smallmatrix}\right)$. Clearly, monoidal product is strictly associative and unital. Therefore, the pentagon and the triangle equations [18] hold trivially. The monoidal product is a bifunctor because $(\Gamma_0 \circ \Gamma) \otimes (\Gamma'_0 \circ \Gamma_0) \sim (\Gamma_0 \otimes \Gamma'_0) \circ (\Gamma \otimes \Gamma')$ with permutation matrix $P = \left(\begin{smallmatrix} \mathbb{1} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1} & \mathbb{0} \\ \mathbb{0} & \mathbb{1} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{1} \end{smallmatrix}\right)$.
Thus, $\mathcal{A}$ is a monoidal category. Finally, we prove that it is symmetric. Let $\sigma_{m,n}$ indicate the symmetry: $\sigma_{m,n} = (0, [\mathbb{0}], \left(\begin{smallmatrix} \mathbb{0} & \mathbb{1}_m \\ \mathbb{1}_n & \mathbb{0} \end{smallmatrix}\right), !, ¡, [(\ )])$

Clearly, the symmetry is its own inverse: $\sigma_{m,n} \circ \sigma_{n,m} = \mathbb{1}_{m+n}$.
Moreover, σ is natural as $\sigma_{n,n'} \circ (\Gamma \otimes \Gamma') \sim (\Gamma' \otimes \Gamma) \circ \sigma_{m,m'}$ with permutation matrix $P = \left(\begin{smallmatrix} \mathbb{0} & \mathbb{1} \\ \mathbb{1} & \mathbb{0} \end{smallmatrix}\right)$.
Lastly, the symmetry satisfies the hexagon equations. Thus, $\mathcal{A}$ is a symmetric monoidal category whose objects are natural numbers. In other words, it is a prop. ◀

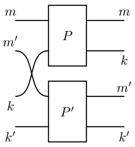
▶ **Lemma 34.** b$\mathbb{P}$ *is a prop.*

Proof. The proof proceeds exactly as the previous one. We will use diagrammatic calculus of **Mat** for the permutation matrix of the morphisms in order to make the proofs more readable. We start by proving that **b**$\mathbb{P}$ is a category. Composition is well-defined on equivalence classes by the monoidal structure of **Mat**. Let $(k, P) \sim (k, \left(\begin{smallmatrix} \mathbb{1}_m & \mathbb{0} \\ \mathbb{0} & \sigma \end{smallmatrix}\right) P) = (k, P')$ and $(l, Q) \sim (l, \left(\begin{smallmatrix} \mathbb{1}_{m+k} & \mathbb{0} \\ \mathbb{0} & \rho \end{smallmatrix}\right) Q) = (l, Q')$.

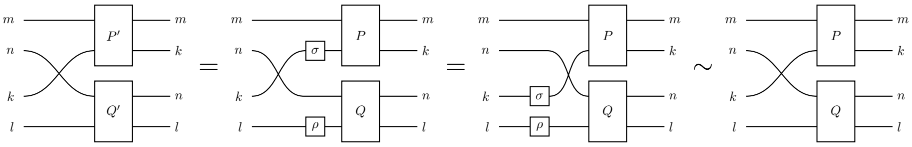


with permutation matrix $\left(\begin{smallmatrix} \mathbb{1}_m & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \sigma & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \rho \end{smallmatrix}\right)$. Composition is clearly associative and unital because it is associative and unital in **Mat**. The monoidal product is defined with a symmetry on the left because we need to keep track of which of the inputs are bound.

$$(k, P) \otimes (k', P') = (k+k', \begin{pmatrix} \mathbb{1}_m & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1}_{m'} & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_k & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{1}_{k'} \end{pmatrix} \left(\begin{smallmatrix} P & \mathbb{0} \\ \mathbb{0} & P' \end{smallmatrix}\right))$$

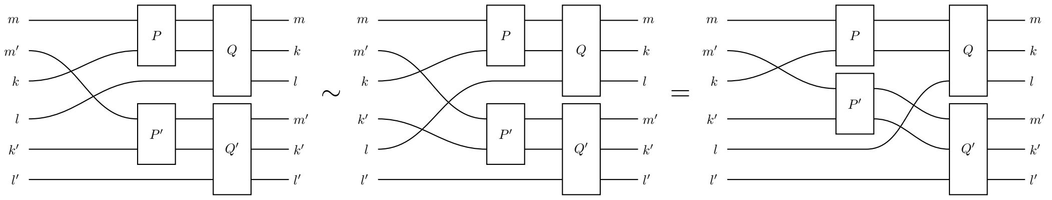


The monoidal unit is the empty diagram: $\mathbb{I} = (0, (\ ))$. The monoidal product is well-defined on equivalence classes by naturality of the symmetries in **Mat**. Let $(k, P) \sim (k, \left(\begin{smallmatrix} \mathbb{1}_m & \mathbb{0} \\ \mathbb{0} & \sigma \end{smallmatrix}\right) P) = (k, P')$ and $(l, Q) \sim (l, \left(\begin{smallmatrix} \mathbb{1}_n & \mathbb{0} \\ \mathbb{0} & \rho \end{smallmatrix}\right) Q) = (l, Q')$.



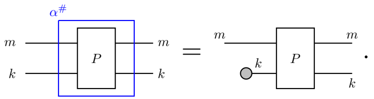
with permutation matrix $\left(\begin{smallmatrix} \mathbb{1}_{m+n} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \sigma & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \rho \end{smallmatrix}\right)$. The monoidal product is a functor because we can change the order in which we enumerate the vertices and because symmetries are natural in

Mat.

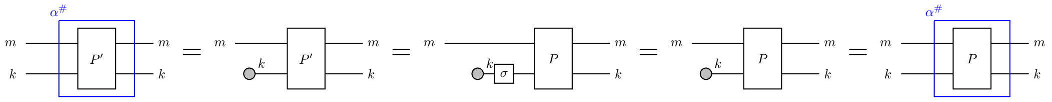


with matrix $\begin{pmatrix} \mathbb{1}_{m+m'+k} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1}_{k'} & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_l & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{1}_{l'} \end{pmatrix}$. The monoidal product is clearly unital. The symmetry is lifted from **Mat**: $\sigma_{m,n} = (0, \left(\begin{smallmatrix} \mathbb{0} & \mathbb{1}_m \\ \mathbb{1}_n & \mathbb{0} \end{smallmatrix}\right))$. The symmetry is its own inverse and it satisfies the hexagon equations because it does so in **Mat**.
Therefore, $\mathbf{b}\mathbb{P}$ is a prop. ◀

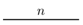
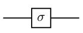
▶ **Lemma 35.** $\mathbf{b}\mathbb{P}$ *is isomorphic to the free prop on one generator* $0 \to 1$.

Proof. Define $\alpha^\# \colon \mathbf{b}\mathbb{P} \longrightarrow \mathbb{P}$ to be $\alpha^\#(k, P) = P \circ (\mathbb{1}_m \otimes \bigotimes_k v)$, where $P \in \mathbb{P}$ is the product of the symmetries that form P in $\mathbf{b}\mathbb{P}$. Diagrammatically, .

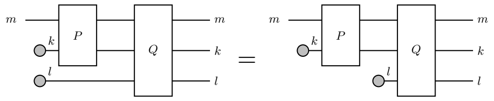
We prove that $\alpha^\#$ is well-defined on equivalence classes. Let $(k, P) \sim (k, \left(\begin{smallmatrix} \mathbb{1}_m & \mathbb{0} \\ \mathbb{0} & \sigma \end{smallmatrix}\right) P) = (k, P')$.



We show graphically that $\alpha^\#$ is a prop homomorphism

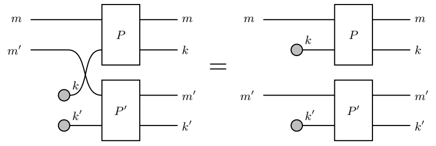
$\alpha^\#(\mathbb{0}) = \quad = \mathbb{0}$ $\alpha^\#(\mathbb{1}_n) =$  $= \mathbb{1}_n$ $\alpha^\#(0, \sigma) =$  $= \sigma$

$\alpha^\#((l, Q) \circ (k, P)) =$



$= \alpha^\#(l, Q) \circ \alpha^\#(k, P)$

$\alpha^\#((k, P) \otimes (k', P')) =$



$= \alpha^\#(k, P) \otimes \alpha^\#(k', P')$

and, by its definition,

$$\alpha^\#(\phi) = v$$

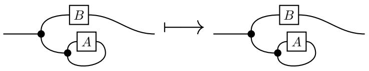
Moreover, $\alpha^\#$ is the unique morphism $\mathbf{b}\mathbb{P} \to \mathbb{P}$ with this property. In fact, suppose there is $\beta \colon \mathbf{b}\mathbb{P} \to \mathbb{P}$ such that $\beta(\phi) = v$. Then,

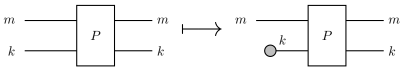
$$\begin{aligned} \beta(k, P) =& \beta((0, P) \circ ((0, \mathbb{1}_n) \otimes (k, \mathbb{1}_k))) = \beta(0, P) \circ (\beta(0, \mathbb{1}_n) \otimes \beta(k, \mathbb{1}_k)) \\ =& P \circ (\mathbb{1}_n \otimes \bigotimes_k v) = a^\#(k, P) \end{aligned}$$

Then $\mathbf{b}\mathbb{P}$ is isomorphic to the free prop over one generator $0 \to 1$. ◀

▷ Claim 36. The following are prop homomorphisms.

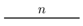
$$\begin{aligned} i_1\colon \mathbf{Adj} &\longrightarrow \mathcal{A} \\ n &\longmapsto n \\ (B,[A]) &\longmapsto (0,[A],B,!,¡,[(\ )]) \end{aligned} \qquad\qquad \begin{aligned} i_2\colon \mathbf{b}\mathbb{P} &\longrightarrow \mathcal{A} \\ n &\longmapsto n \\ (k,P) &\longmapsto (k,[\mathbb{0}_n],P_*^{[1,n]},\mathbb{0}_{nk},P_*^{[n+1,n+k]},[\mathbb{0}_k]) \end{aligned}$$



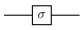


Proof. We prove graphically that they are prop homomorphisms.

$i_1(\mathbb{0}) = \quad = \mathbb{0}$ $\quad i_1(\mathbb{1}_n) =$

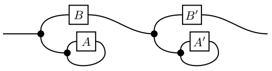


$= \mathbb{1}_n \quad i_1(\sigma,[(\ )]) =$



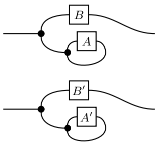
$= \sigma$

$i_1((B',[A'])\circ(B,[A])) =$



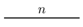
$= i_1(B',[A'])\circ i_1(B,[A])$

$i_1((B,[A])\otimes(B',[A'])) =$

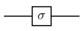


$= i_1(B,[A])\otimes i_1(B',[A'])$

$i_2(\mathbb{0}) = \quad = \mathbb{0}$ $\quad i_2(\mathbb{1}_n) =$

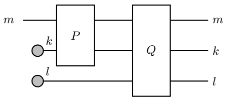


$= \mathbb{1}_n \quad i_2(0,\sigma) =$

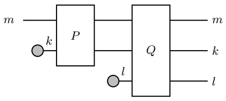


$= \sigma$

$i_2((l,Q)\circ(k,P)) =$

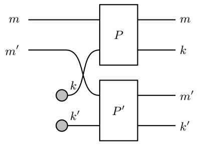


$=$

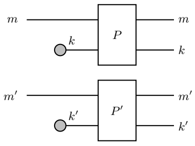


$= i_2(l,Q)\circ i_1(k,P)$

$i_2((k,P)\otimes(k',P')) =$



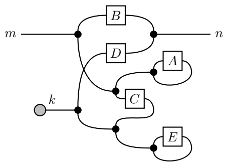
$=$



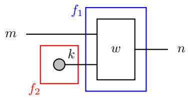
$= i_2(k,P)\otimes i_2(k',P')$ ◀

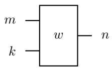
▶ **Lemma 37.** *H, defined on page 12, is a prop homomorphism.*

Proof. Recall that $H\colon \mathcal{A}\to\mathcal{C}$ is identity on objects and $H(k,[A],B,C,D,[E]) = f_1\left(\left(\begin{smallmatrix}B\\D\end{smallmatrix}\right),\left[\left(\begin{smallmatrix}A & C\\ \mathbb{0} & E\end{smallmatrix}\right)\right]\right)\circ(\mathbb{1}_m\otimes f_2(k,\mathbb{1}_k))$. By calling $w = \left(\left(\begin{smallmatrix}B\\D\end{smallmatrix}\right),\left[\left(\begin{smallmatrix}A & C\\ \mathbb{0} & E\end{smallmatrix}\right)\right]\right)$, which is a morphism in **Adj**, we can depict the image of H diagrammatically.

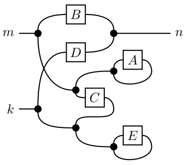


$\longmapsto$



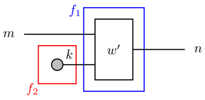


$=$

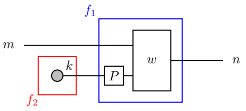


We need to prove that H is well-defined on equivalence classes. Let $\Gamma = (k,[A],B,C,D,[E]) \sim (k,[A],B,CP^T,PD,\left[PEP^T\right]) = \Gamma'$.

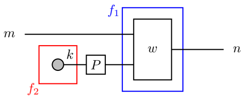
$H(\Gamma') =$



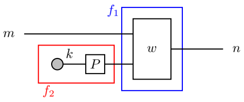
$=$



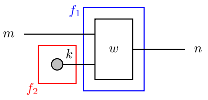
$=$



$=$



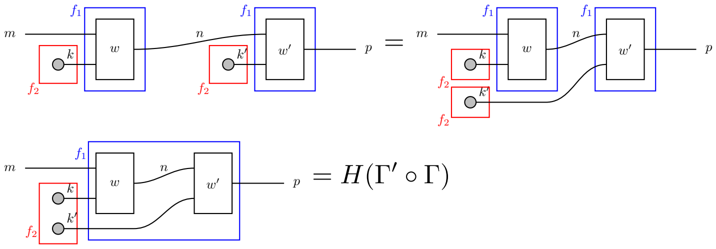
$=$



$= H(\Gamma)$

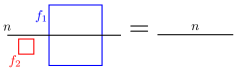
We prove that H is a prop homomorphism. Clearly, H is identity on objects. Moreover, it preserves composition, as it is shown by the diagrams.

$H(\Gamma') \circ H(\Gamma) =$



$= H(\Gamma' \circ \Gamma)$

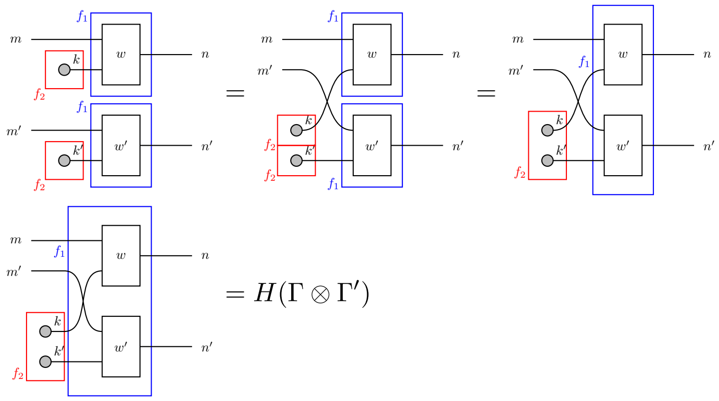
H preserves identities: $H(\mathbb{1}_n) =$



$= \mathbb{1}_n.$

H preserves monoidal product. This is also more clearly seen with string diagrams.

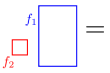
$H(\Gamma) \otimes H(\Gamma') =$



$= H(\Gamma \otimes \Gamma')$

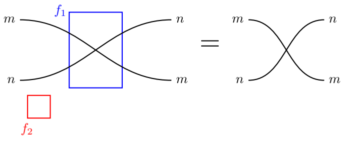
It is easy to show that H preserves monoidal unit and symmetries.

$H(\mathbb{I}) =$



$= \mathbb{I}$

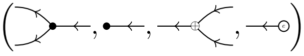
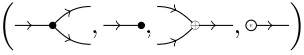
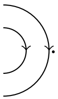
$H(\sigma_{m,n}) =$



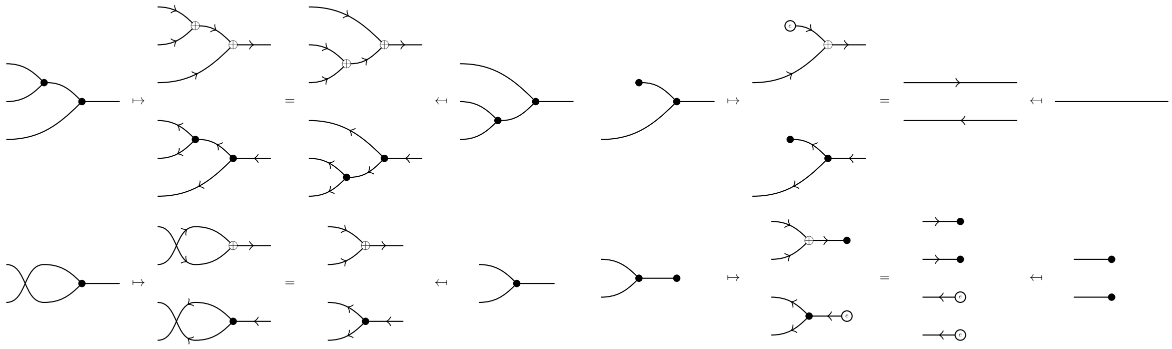
$= \sigma_{m,n}$

◀

B Proofs for Sections 5

Proof of Theorem 27. $F_{\mathcal{N}}$ respects the equations of **Grph** because both the tuples  and  satisfy the commutative bialgebra axioms in figure 2, and they both interact as in figure 3 with the cup .

We explain in detail that the functor $F_{\mathcal{N}}$ preserves associativity of the black monoid, the rest of the equations are written with the same convention. We write on the left-most and right-most sides morphisms in **Grph** that, by associativity, they must be equal. In the centre, we write the morphisms in **Game** to which they are mapped (indicated with ↦) by the functor $F_{\mathcal{N}}$. These morphisms are equal in **Game** by associativity of the monoid operation $\oplus$ on M and coassociativity of copying. Thus, we can say that $F_{\mathcal{N}}$ preserves associativity of the black monoid.



and similarly for their transposed versions. ◀

Proof of Theorem 29. The proof proceeds by structural induction on Γ.
It is straightforward to check from the definition of $F_{\mathcal{N}}$ that the generators of **Grph** are sent to open games of the required form.
We need to check that composition is of the form as in the statement. We compute explicitly its play, coplay and best response functions.

$$\mathbb{P}_{F_{\mathcal{N}}(\Gamma'\circ\Gamma)}((\underline{\sigma},\underline{\sigma}'),\underline{x}) = \mathbb{P}_{F_{\mathcal{N}}(\Gamma')}(\underline{\sigma}', B^T\underline{x}\oplus D^T f(\underline{\sigma})) = (BB')^T\underline{x}\oplus \left(\begin{smallmatrix} DB' \\ D' \end{smallmatrix}\right)^T f(\left(\begin{smallmatrix} \underline{\sigma} \\ \underline{\sigma}' \end{smallmatrix}\right))$$

$$\mathbb{C}_{F_{\mathcal{N}}(\Gamma'\circ\Gamma)}((\underline{\sigma},\underline{\sigma}'),\underline{x},\underline{q}) = \mathbb{C}_{F_{\mathcal{N}}(\Gamma)}(\underline{\sigma},\underline{x},\mathbb{C}_{F_{\mathcal{N}}(\Gamma')}(\underline{\sigma}',\mathbb{P}_{F_{\mathcal{N}}(\Gamma)}(\underline{\sigma},\underline{x}),\underline{q}))$$
$$= ((A+BA'B^T)+(A+BA'B^T)^T)\underline{x}\oplus BB'\underline{q}\oplus (C+B(A'+A'^T)D^T|BC')f(\left(\begin{smallmatrix} \underline{\sigma} \\ \underline{\sigma}' \end{smallmatrix}\right))$$

$$(\underline{\rho},\underline{\rho}') \in \mathbb{B}_{F_{\mathcal{N}}(\Gamma'\circ\Gamma)}(\underline{x},\kappa)$$
$$\Leftrightarrow (\underline{\sigma},\underline{\sigma}') \in \mathbb{B}_{F_{\mathcal{N}}(\Gamma)}(\underline{x},\kappa\circ F_{\mathcal{N}}(\Gamma')_{\underline{\tau}}) \wedge (\underline{\tau},\underline{\tau}') \in \mathbb{B}_{F_{\mathcal{N}}(\Gamma')}(F_{\mathcal{N}}(\Gamma)_{\underline{\sigma}}\circ\underline{x},\kappa)$$
$$\Leftrightarrow \forall a = 1,\ldots,k+k'$$
$$s \in \operatorname*{argmax}_{s\in X} g\Big(s, \begin{pmatrix} C^T+D(A'+A'^T)B^T \\ C'^T B^T \end{pmatrix}^a_* \underline{x}$$
$$\oplus \left(\begin{smallmatrix} DB' \\ D' \end{smallmatrix}\right)^a_* \kappa((BB')^T\underline{x}\oplus (B'^T D^T|D'^T) f(\underline{\rho}[a\mapsto \rho'_a]))$$
$$\oplus \begin{pmatrix} E+E^T+D(A'+A'^T)D^T & DC' \\ C'^T D^T & E'+E'^T \end{pmatrix}^a_* f(\underline{\rho}[a\mapsto\rho'_a])\Big)$$

Similarly, we show that monoidal product has the desired form.

$$\mathbb{P}_{F_{\mathcal{N}}(\Gamma\otimes\Gamma')}((\underline{\sigma},\underline{\sigma}'),(\underline{x},\underline{x}')) = \left(\begin{smallmatrix} B & \mathbb{0} \\ \mathbb{0} & B' \end{smallmatrix}\right)^T \begin{pmatrix} \underline{x} \\ \underline{x}' \end{pmatrix} \oplus \left(\begin{smallmatrix} D & \mathbb{0} \\ \mathbb{0} & D' \end{smallmatrix}\right)^T f(\begin{pmatrix} \underline{\sigma} \\ \underline{\sigma}' \end{pmatrix})$$

$$\begin{aligned}&\mathbb{C}_{F_{\mathcal{N}}(\Gamma\otimes\Gamma')}((\underline{\sigma},\underline{\sigma}'),(\underline{x},\underline{x}'),(\underline{r},\underline{r}'))\\ &= \left(\left(\begin{smallmatrix} A & \mathbb{0} \\ \mathbb{0} & A' \end{smallmatrix}\right) + \left(\begin{smallmatrix} A & \mathbb{0} \\ \mathbb{0} & A' \end{smallmatrix}\right)^T\right) \begin{pmatrix} \underline{x} \\ \underline{x}' \end{pmatrix} \oplus \left(\begin{smallmatrix} B & \mathbb{0} \\ \mathbb{0} & B' \end{smallmatrix}\right) \begin{pmatrix} \underline{r} \\ \underline{r}' \end{pmatrix} \oplus \left(\begin{smallmatrix} B & \mathbb{0} \\ \mathbb{0} & B' \end{smallmatrix}\right) f(\begin{pmatrix} \underline{\sigma} \\ \underline{\sigma}' \end{pmatrix})\end{aligned}$$

$$\begin{aligned}&(\underline{\rho},\underline{\rho}') \in \mathbb{B}_{F_{\mathcal{N}}(\Gamma'\otimes\Gamma)}((\underline{x},\underline{x}'),\langle\kappa,\kappa'\rangle)\\ \Leftrightarrow&\forall a = \ ...,k+k' \ \rho'_a \in \operatorname*{argmax}_{s\in X} g\Big(s, \Big(\left(\begin{smallmatrix} C & \mathbb{0} \\ \mathbb{0} & C' \end{smallmatrix}\right)^T\Big)^a_* \begin{pmatrix} \underline{x} \\ \underline{x}' \end{pmatrix} \oplus \Big(\left(\begin{smallmatrix} D & \mathbb{0} \\ \mathbb{0} & D' \end{smallmatrix}\right)^T\Big)^a_* \langle\kappa,\kappa'\rangle(\left(\begin{smallmatrix} B & \mathbb{0} \\ \mathbb{0} & B' \end{smallmatrix}\right)^T \begin{pmatrix} \underline{x} \\ \underline{x}' \end{pmatrix}\\ &\oplus \left(\begin{smallmatrix} D & \mathbb{0} \\ \mathbb{0} & D' \end{smallmatrix}\right)^T f(\underline{\rho}[a\mapsto s])) \oplus \Big(\left(\begin{smallmatrix} E & \mathbb{0} \\ \mathbb{0} & E' \end{smallmatrix}\right) + \left(\begin{smallmatrix} E & \mathbb{0} \\ \mathbb{0} & E' \end{smallmatrix}\right)^T\Big)^a_* f(\underline{\rho}[a\mapsto s])\Big) \qquad \blacktriangleleft\end{aligned}$$

Canonization for Bounded and Dihedral Color Classes in Choiceless Polynomial Time

Moritz Lichter
TU Kaiserslautern, Germany
lichter@cs.uni-kl.de

Pascal Schweitzer
TU Kaiserslautern, Germany
schweitzer@cs.uni-kl.de

Abstract

In the quest for a logic capturing PTIME the next natural classes of structures to consider are those with bounded color class size. We present a canonization procedure for graphs with dihedral color classes of bounded size in the logic of Choiceless Polynomial Time (CPT), which then captures PTIME on this class of structures. This is the first result of this form for non-abelian color classes.

The first step proposes a normal form which comprises a "rigid assemblage". This roughly means that the local automorphism groups form 2-injective 3-factor subdirect products. Structures with color classes of bounded size can be reduced canonization preservingly to normal form in CPT.

In the second step, we show that for graphs in normal form with dihedral color classes of bounded size, the canonization problem can be solved in CPT. We also show the same statement for general ternary structures in normal form if the dihedral groups are defined over odd domains.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Complexity theory and logic

Keywords and phrases Choiceless polynomial time, canonization, relational structures, bounded color class size, dihedral groups

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.31

Related Version A full version [21] is available at `https://arxiv.org/abs/2010.12182`.

Funding Received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EngageS: agreement No. 820148).

1 Introduction

One of the central open questions in the field of descriptive complexity theory asks about the existence of a logic that captures polynomial time (PTIME) [11]. This question goes back to Chandra and Harel [5]. They ask whether there is a logic within which we can define exactly the polynomial-time computable properties of relational structures. For the complexity class NP such a logic is known, namely existential second order logic. This was shown by Fagin in his famous theorem [7]. However, for the class PTIME, the question has been open now for more than 35 years. A fundamental difficulty at its heart is a mismatch between logics and Turing machines. An input has to be written onto a tape to provide it to a Turing machine. So all inputs are necessarily ordered by the position of each character on the tape. This is the case even when there is no natural order to begin with, which for example happens with the vertices of a graph that is encoded. In contrast to this, such an order is typically not given for a logic. In fact, if an order is given a priori then there is a logic capturing PTIME, for example on totally ordered structures. Indeed, IFP (first order logic enriched with a fixed-point operator) is such a logic as shown by the Immerman-Vardi Theorem [18].

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 31; pp. 31:1–31:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the ongoing search for a logic for unordered structures, one of the most promising candidates is the logic Choiceless Polynomial Time (CPT). It manages to capture an important aspect demanded from a "reasonable logic" in the sense of Gurevich [17], namely that such a logic cannot make arbitrary choices. Whenever there are multiple indistinguishable elements, a logic can either process all or none of them. It is impossible to pick one element arbitrarily and process just that. This is common for many algorithms executed on Turing machines exploiting the order given by the tape. In the form originally defined by Blass, Gurevich, and Shelah [3], CPT has a pseudocode-like syntax for processing hereditarily finite sets. Most importantly there is a construct to process all elements of a set in parallel, because we cannot choose one to process first. Subsequently, there were definitions by Rossman [27] and Grädel and Grohe [9] in a more "logical" way using iteration terms or fixed points.

The question of whether a logic capturing PTIME on a class of structures exists is closely linked to the problem of canonization. Suppose it is possible to canonize input structures from a particular class in a logic (i.e., to define an isomorphic copy enriched by a total order). Then the logic (extended by IFP) captures PTIME on this class by the already mentioned Immerman-Vardi Theorem. This yields a general approach to show that some logic captures PTIME on a class of structures: proving that canonization of the structures is definable in this logic. This approach has been the method of choice for numerous results in descriptive complexity theory. To this end, it was shown that canonization is IFP+C (IFP with counting) definable on interval graphs [20], graphs with excluded minors [10, 12, 13, 14], and graphs with bounded rank width [15]. Thus IFP+C captures PTIME on these classes. Regarding CPT, all CPT-definable properties and transformations (e.g., canonization) are in particular polynomial-time computable. If canonization is CPT-definable for a graph class then CPT captures PTIME on this class because CPT subsumes IFP+C.

Closely linked to the problem of canonization is the problem of isomorphism testing. A polynomial-time canonization algorithm implies a polynomial-time isomorphism test. While we do not know of a formal reduction the other way around, we usually have efficient canonization algorithms for all classes for which an efficient isomorphism test is known (see [28] for an overview). This statement can even be proven unconditionally for classes of vertex-colored structures with a CPT-definable isomorphism problem [16]. Accepting for the moment that canonization and isomorphism testing are algorithmically very related, we arrive at the following observation: if isomorphism testing is polynomial-time solvable on a class of structures then, to capture PTIME, we must "solve" the isomorphism problem in the logic anyway. If we do so in CPT we (almost) immediately obtain a logic capturing PTIME. In summary, it appears the question of a logic for PTIME boils down to isomorphism testing within a logic.

There is a notable class for which we have polynomial-time isomorphism testing and canonization algorithms, but for which we do not know how to canonize them in CPT. This is the class of structures with bounded color class size. Specifically, for an integer q, a q-bounded structure is a vertex-colored structure, where at most q vertices have the same color and a total order on the colors is given. Structures with bounded color class size can be canonized in polynomial time with group-theoretic techniques (see [8, 1, 2]). The introduction of group-theoretic techniques marks an important step for the design of canonization algorithms. The use of algorithmic group theory turned out to be very fruitful and subsequently lead to Luks's famous polynomial-time isomorphism test for graphs of bounded degree [22]. It uses a more general and more complicated machinery than needed for bounded color classes.

For the purpose of isomorphism testing, these group-theoretic techniques inherently rely on choosing generating sets, and it is not clear how this can be done in a choiceless logic. A well-known construction of Cai, Fürer, and Immerman [4] shows that IFP+C does not provide us with a Ptime logic for 2-bounded structures. Finding a natural, alternative logic for q-bounded structures is still an open problem. Studying structures with bounded color class size is a reasonable a next step, because the canonization algorithm for them makes use of comparatively easy group theory but we still do not know how to transfer these techniques into logics.

A first result towards the canonization of structures with bounded color class size in CPT was the canonization of structures with abelian colors, that is the automorphism group of every color class is abelian, due to Abu Zaid, Grädel, Grohe, and Pakusa [29]. They use a certain class of linear equation systems to encode the group-theoretic structure of abelian color classes and solve these systems in CPT. In particular, they show that CPT captures Ptime on 2-bounded structures. Considering dihedral groups is a next natural step because dihedral groups are extensions of abelian groups by abelian groups.

Contribution. This paper presents a canonization procedure in CPT for finite q-bounded structures with dihedral colors. A color class is dihedral (resp. cyclic) if it induces a substructure whose automorphism group is dihedral (resp. cyclic). A dihedral group is the automorphism group of a regular n-gon consisting of rotations and reflections and we call it odd if n is odd. Dihedral groups are non-abelian for $n > 2$. We thereby provide the first canonization procedure for a class of q-bounded structures with non-abelian color classes and in particular show that CPT captures Ptime on it. Overall, we prove the following theorem:

▶ **Theorem 1.** *The following structures can be canonized in CPT:*

1. *q-bounded relational structures of arity at most 3 with odd dihedral or cyclic colors*
2. *q-bounded graphs with dihedral or cyclic colors.*

Our approach consists of two steps. As a first step, we propose a normal form for arbitrary finite q-bounded structures. Then, in a second step, we use group-theoretic arguments to canonize structures with dihedral colors given in the aforementioned normal form.

Concretely, the first step is a reduction transforming the input structure into a normal form, which ensures that a color class and its adjacent color classes form a “rigid assemblage”. That is, locally the automorphism groups form 2-injective 3-factor subdirect products or are quotient groups of other color classes. In the the case of 2-injective 3-factor subdirect products, the automorphisms of three adjacent color classes are not independent of each other. This means that every nontrivial automorphism of the substructure induced by these three color classes is never constant on two of them. More precisely, we prove the following theorem (formal definitions and proofs are given later in the paper).

▶ **Theorem 2.** *For every q and signature τ there is a q' and another signature τ', such that relational q-bounded τ-structures of arity r can be reduced canonization preservingly in CPT to q'-bounded 2-injective quotient τ'-structures.*

It was not necessary to consider 2-injective groups for abelian colors yet, but it is for non-abelian colors. Towards a reduction step, a purely group-theoretic analysis of 2-injective groups is given in [23]. The main insight is basically that such groups decompose naturally into structurally simpler parts which are related via a common abelian normal subgroup. We extend the techniques to canonize abelian color classes and show how they can be combined with the analysis of 2-injective groups to obtain a canonization procedure for said structures with dihedral colors in CPT. That is, we provide new methods to integrate group-theoretic reasoning, which is at the core of canonizing q-bounded structures algorithmically, into logics.

Our Technique. The strategy of our canonization procedure is to reduce the dihedral groups in some way to abelian groups and then exploit the canonization procedure of [29].

Since the automorphism groups of the color classes are restricted to be dihedral or cyclic, we can characterize all occurring 2-injective 3-factor subdirect products. Using this characterization we show that we can partition the input structure into parts we call reflection components. These reflection components have the property that automorphisms either simultaneously reflect the points in all color classes of the component or in none. In the latter case they rotate the points in all color classes. We use this property to force all groups in a reflection component to become abelian: we prohibit reflections in one color class of the reflection component and this automatically prohibits reflections in all other classes of the component, too. Once all reflections are removed, the remaining groups are abelian. Then we apply the canonization procedure for structures with bounded abelian color classes from [29] to the entire reflection component.

Not limiting ourselves to dihedral groups but also allowing cyclic groups has the benefit that the class of occurring groups is closed under quotients and subgroups. Quotient and subgroups of the input color classes occur naturally in our reduction process to the normal form. For dihedral groups it turns out that odd dihedral groups are easier to handle than the other ones. In fact, reflection components are not really independent but can have "global" dependencies. We show that for "odd" dihedral groups, reflection components can only be related via color classes with abelian automorphism groups, that is their global dependencies are abelian. For "even" dihedral groups, there is a single non-abelian exception that can connect reflection components, which complicates matters. For even dihedral color classes this restricts us to the treatment of graphs (see Theorem 1 above).

Towards generalization, it unfortunately becomes cumbersome to exploit the group structure theory in CPT, which is heavily required to execute the approach. Extending the treatment of linear equation systems, which is a subroutine in [29], to dihedral groups requires significant work already. We still follow the strategy of [29] and use a certain class of equation systems to encode the global dependencies. However, we need to generalize the equation systems. Consequently, we have to adapt all operations used on these equations systems to work in the more general setting (e.g. the check for consistency). This becomes technically even more involved than the techniques of [29] already are.

Related Work. There already exist various results for CPT regarding structures with bounded color class size in addition to the ones mentioned above: Cai, Fürer, and Immerman introduced the so-called CFI graphs. From every base graph, a pair of two non-isomorphic CFI graphs is derived. The isomorphism problem on these pairs of graphs is used to separate IFP+C from Ptime [4]. Dawar, Richerby, and Rossman showed in [6] that the isomorphism problem for the CFI graphs can be solved in CPT for base graphs of color class size 1.

This result was strengthened by Pakusa, Schalthöfer, and Selman to base graphs with logarithmic color class size [26]. The techniques of [6] and [26] are used in [29] to solve the mentioned equation systems.

The logic IFP+C has a strong connection to the higher dimensional Weisfeiler-Leman algorithm and to an Ehrenfeucht–Fraïssé-like game, the so-called bijective pebble game. They are often used to show that IFP+C identifies graphs in a given graph class, i.e., that for any two non-isomorphic graphs of the class there is an IFP+C formula distinguishing them. It was shown by Otto [24] that if IFP+C captures Ptime on a graph class then IFP+C also identifies all graphs in this class. The converse direction is open [12]. The capabilities of IFP+C to detect graph decompositions were recently investigated in [19].

Structure of this Paper. We begin with the characterization of 2-injective 3-factor subdirect products of dihedral and cyclic groups in Section 3. Then we turn to structures and to permutation groups. We begin with the reduction to the above mentioned normal form in Section 4 and then preprocess structures with dihedral colors in Section 5. In Section 6 we introduce tree-like cyclic linear equation systems (TCES) and show that a certain subclass of them can be solved in CPT. Finally, we define and analyze reflection components in Section 7 and give the CPT-definable canonization procedure for dihedral colors. Full formal proofs can be found in [21].

2 Preliminaries

Bounded Relational Structures. A (relational) signature $\tau = \{R_1, \dots, R_k\}$ is a set of relation symbols with associated arities $r_i \in \mathbb{N}$ for all $i \in [k] := \{1, \dots, k\}$. We consider signatures containing a binary relation symbol $\preceq$. A τ-structure $\mathcal{H}$ is a tuple $\mathcal{H} = (H, R_1^{\mathcal{H}}, \dots, R_k^{\mathcal{H}}, \preceq)$ where $R_i^{\mathcal{H}} \subseteq H^{r_i}$ for all $i \in [k]$ and $\preceq \subseteq H^2$ is a total preorder. Unless said otherwise, all structures considered in this paper will be finite. The preorder $\preceq$ partitions H into equivalence classes, which we call *color classes*, and induces a total order on them. We denote the set of $\mathcal{H}$-color classes by $\mathbb{C}_{\mathcal{H}}$. For a set $I \subseteq H$ we denote with $\mathcal{H}[I]$ the substructure induced by I. If $I = C$ is a color class, we just write $\mathcal{C}$ for $\mathcal{H}[C]$, if the structure $\mathcal{H}$ is clear from the context. If $I \subseteq \mathbb{C}_{\mathcal{H}}$ we also write $\mathcal{H}[I]$ for $\mathcal{H}[\bigcup I]$. Two colors classes $C, C' \in \mathbb{C}_{\mathcal{H}}$ are *related*, if there is some tuple containing a vertex from C and C' in some $R_i^{\mathcal{H}}$.

A relation $R_i^{\mathcal{H}}$ is *homogeneous* if $R_i^{\mathcal{H}} \subseteq C^k$ for some $C \in \mathbb{C}_{\mathcal{H}}$ and $k \in \mathbb{N}$, otherwise it is *heterogeneous*. A structure $\mathcal{H}$ is of *arity* r if the largest arity of a heterogeneous relation is r. A structure is *q-bounded*, if $|C| \leq q$ for all $C \in \mathbb{C}_{\mathcal{H}}$ and the arity of every homogeneous relation is bounded by q. We write $\mathrm{Aut}(\mathcal{H})$ for the automorphism group of $\mathcal{H}$. For two structures $\mathcal{H}_1$ and $\mathcal{H}_2$ we write $\mathrm{Iso}(\mathcal{H}_1, \mathcal{H}_2)$ for the set of isomorphisms between $\mathcal{H}_1$ and $\mathcal{H}_2$.

An ordered copy of $\mathcal{H}$ is a pair $(\mathcal{H}', <)$, such that $\mathcal{H}' = (H', R_1^{\mathcal{H}'}, \dots, R_k^{\mathcal{H}'}, \preceq')$, $\mathcal{H}' \cong \mathcal{H}$, and $<$ is a total order that refines $\preceq'$. A canonical copy $\mathsf{can}(\mathcal{H})$ is an ordered copy of $\mathcal{H}$ obtained in a canonical way, i.e., defined in CPT in the following. For a canonical copy $\mathsf{can}(\mathcal{H})$ we call the set $\mathrm{Iso}(\mathcal{H}, \mathsf{can}(\mathcal{H}))$ the *canonical labellings.*

Choiceless Polynomial Time. To give a concise definition of CPT, we follow the definition of [9] and use the same idea of e.g. [25] to enforce polynomial bounds.

For a set of atoms A we denote with $\mathsf{HF}(A)$ the *hereditarily finite sets over* A. This is the inclusion-wise smallest set with $A \subseteq \mathsf{HF}(A)$ and $a \in \mathsf{HF}(A)$ for every $a \subseteq \mathsf{HF}(A)$. A set $a \in \mathsf{HF}(A)$ is called transitive, if $c \in a$ whenever there is some b with $c \in b \in a$. We denote with $\mathsf{TC}(a)$ the *transitive closure* of a, that is the least transitive set b with $a \subseteq b$.

Let τ be a signature. We extend τ by adding set-theoretic function symbols to obtain $\tau^{\mathsf{HF}} := \tau \uplus \{\emptyset, \mathsf{Atoms}, \mathsf{Pair}, \mathsf{Union}, \mathsf{Unique}, \mathsf{Card}\}$. For a τ-structure $\mathcal{H}$, the *hereditarily finite expansion* $\mathsf{HF}(\mathcal{H})$ is a τ^{HF}-structure over the universe $\mathsf{HF}(H)$ defined as follows: all relations in τ are interpreted as in $\mathcal{H}$. The other function symbols have the expected interpretation:

- $\emptyset^{\mathsf{HF}(\mathcal{H})} = \emptyset$ and $\mathsf{Atoms}^{\mathsf{HF}(\mathcal{H})} = H$,
- $\mathsf{Pair}^{\mathsf{HF}(\mathcal{H})}(a, b) = \{a, b\}$ and $\mathsf{Union}^{\mathsf{HF}(\mathcal{H})}(a) = \{b \mid \exists c \in a.\ b \in c\}$,
- $\mathsf{Unique}^{\mathsf{HF}(\mathcal{H})}(a) = \begin{cases} b & \text{if } a = \{b\} \\ \emptyset & \text{otherwise} \end{cases}$ and $\mathsf{Card}^{\mathsf{HF}(\mathcal{H})}(a) = \begin{cases} |a| & \text{if } a \notin H \\ \emptyset & \text{otherwise} \end{cases}$,

  where the number $|a|$ is encoded as a von Neuman ordinal.

A *BGS term* is composed as usual from variables and function symbols from τ^{HF}. There are two additional constructs: if $s(\bar{x}, y)$ and $t(\bar{x})$ are terms and $\varphi(\bar{x}, y)$ is a formula then

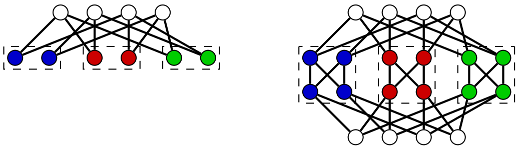


Figure 1 Two vertex-colored graphs whose automorphism groups are the CFI group (left) and the double CFI group (right).

$r(\bar{x}) = \{s(\bar{x}, y) \mid y \in t(\bar{x}), \varphi(\bar{x}, y)\}$ is a *comprehension term*. If $s(x)$ is a term with a single free variable x, then s^* is an *iteration term*. *BGS formulas* are composed of terms $t_1, \dots, t_k$ as $R(t_1, \dots, t_k)$ (for $R \in \tau$ of arity k), $t_1 = t_2$, and the usual boolean connectives.

Let $\mathcal{H}$ be a τ structure. BGS terms and formulas are interpreted over $\mathsf{HF}(H)$. We define the denotation $[\![t]\!]^{\mathcal{H}} \colon \mathsf{HF}(H)^k \to \mathsf{HF}(\mathcal{H})$ that for a term $t(\bar{x})$ with free variables $\bar{x} = (x_1, \dots, x_k)$ maps $\bar{a} = (a_1, \dots, a_k) \in \mathsf{HF}(\mathcal{H})^k$ to the value of t if we replace x_i with a_i. For a formula $\varphi(\bar{x})$ we define $[\![\varphi]\!]^{\mathcal{H}}$ to be the set of all $\bar{a} = (a_1, \dots, a_k) \in \mathsf{HF}(\mathcal{H})^k$ satisfying φ.

For the comprehension term r as above, the denotation is defined as follows: $[\![r]\!]^{\mathcal{H}}(\bar{a}) = \big\{[\![s]\!]^{\mathcal{H}}(\bar{a}b) \mid b \in [\![t]\!]^{\mathcal{H}}(\bar{a}), (\bar{a}b) \in [\![\varphi]\!]^{\mathcal{H}}\big\}$, where $\bar{a}b = (a_1, \dots, a_k, b)$. For an iteration term s^* we define a sequence of sets via $a_0 := \emptyset$ and $a_{i+1} := [\![s]\!]^{\mathcal{H}}(a_i)$. Let $\ell := \ell(s^*, \mathcal{H})$ be the least number with $a_{i+1} = a_i$. We set $[\![s^*]\!]^{\mathcal{H}} := a_i$ if such an ℓ exists and $[\![s^*]\!]^{\mathcal{H}} := \emptyset$ otherwise.

A CPT term (or formula respectively) is a tuple (t, p) (or (φ, p) respectively) of a BGS term (or formula) and a polynomial $p(n)$. CPT has the same semantics as BGS by replacing t with (t, p) everywhere (or φ with (φ, p)) with an exception for iteration terms: We set $[\![(s^*, p)]\!]^{\mathcal{H}} := [\![s^*]\!]^{\mathcal{H}}$ if $\ell(s^*, \mathcal{H}) \leq p(|H|)$ and $|\mathsf{TC}(a_i)| \leq p(|H|)$ for all i, where the a_i are defined as above. Otherwise, we set $[\![(s^*, p)]\!]^{\mathcal{H}} := \emptyset$. We use $|\mathsf{TC}(a_i)|$ as a measure of the size of a_i, because by transitivity of $\mathsf{TC}(a_i)$, whenever there is a set $b_k \in \dots \in b_1 \in a_i$, then also $b_k \in \mathsf{TC}(a_i)$ and thus $\mathsf{TC}(a_i)$ counts all sets occurring somewhere in the structure of a_i.

3 Classification of 2-Injective Subdirect Products of Dihedral Groups

We begin with the classification of 2-injective subdirect products of dihedral groups. A group $\Gamma \leq G_1 \times G_2 \times G_3$ is called a *(3-factor) subdirect product* if the projection to each factor is surjective. It is called 2-*injective* if $\ker(\overline{\pi}_i(\Gamma)) = \{1\}$ for all $i \in [3]$, where $\overline{\pi}_i$ is the projection on the two factors apart the i-th. Another way of looking at this is that two components of an element of Γ determine the third one uniquely.

For $n \geq 3$, the *dihedral group* D_n of order $2n$ is the automorphism group of a regular n-gon in the plane. It consists of n rotations and n reflections and acts naturally on the set of n vertices of the polygon. We regard the identity 1 as rotation and write $\mathrm{Rot}(\mathsf{D}_n)$ for the rotation subgroup consisting only of rotations. It is isomorphic to the cyclic group C_n of order n. Only in the degenerate cases D_1 and D_2 the dihedral group is abelian. It holds that $\mathsf{D}_1 \cong \mathsf{C}_2$ and $\mathsf{D}_2 \cong \mathsf{C}_2^2$. Elements in the direct product $\mathsf{D}_{n_1} \times \dots \times \mathsf{D}_{n_k}$ are rotations (resp. reflections) if all components are rotations (resp. reflections). For $n_i \geq 2$ it contains mixed elements that are neither a reflection nor a rotation. Subgroups of such a group may or may not contain mixed elements: A group $\Gamma \leq \mathsf{D}_{n_1} \times \dots \times \mathsf{D}_{n_k}$ with $n_i > 2$ for all $i \in [k]$ is called a *rotate-or-reflect* group if every $g \in \Gamma$ is a rotation or a reflection. Our classification involved many technical proofs and we only state its result here (proofs can be found in [21]). Roughly speaking, almost all 2-injective subdirect products of cyclic and dihedral groups are abelian or rotate-or-reflect groups. There are precisely two exceptions involving the double CFI group:

▶ **Definition 3** (CFI Groups). *We call the group* $\Gamma_{\mathsf{CFI}} := \{(g_1, g_2, g_3) \in \mathsf{D}_1^3 \mid g_1 g_2 g_3 = 1\} < \mathsf{D}_1^3$ *the* CFI group *and the wreath product* $\Gamma_{\mathsf{2CFI}} := \Gamma_{\mathsf{CFI}} \wr \mathsf{C}_2$ *the* double CFI group *(cf. Figure 1).*

▶ **Theorem 4.** *Let* $\Gamma \leq \mathsf{D}_{n_1} \times \mathsf{D}_{n_2} \times \mathsf{D}_{n_3}$ *be a 2-injective subdirect product. Then exactly one of following holds:*

1. $n_i > 2$ *for all* $i \in [3]$ *and* Γ *is a rotate-or-reflect group.*
2. $n_i = 4$ *for all* $i \in [3]$ *and* Γ *is isomorphic to the double CFI group* Γ_{2CFI}*.*
3. $n_i \leq 2$, $n_j = n_k > 2$ *for* $\{i, j, k\} = [3]$*, and* $\overline{\pi}_i(\Gamma)$ *is a rotate-or-reflect group.*
4. $n_i \leq 2$ *for all* $i \in [3]$ *and* Γ *is abelian.*

▶ **Theorem 5.** *Let* $\Gamma \leq \mathsf{C}_{n_1} \times \mathsf{D}_{n_2} \times \mathsf{D}_{n_3}$ *be a 2-injective subdirect product. Then exactly one of the following holds:*

1. $n_1 \leq 2$, $n_2, n_3 > 2$, *and* $\overline{\pi}_1(\Gamma)$ *is a rotate-or-reflect group.*
2. $n_1 = n_2 = n_3 = 4$ *and* $\Gamma \cong \Gamma_{\mathsf{2CFI}} \cap (\mathit{Rot}(\mathsf{D}_4) \times \mathsf{D}_4 \times \mathsf{D}_4)$*.*
3. $n_1, n_2, n_3 \leq 2$ *and* Γ *is abelian.*

Furthermore, there are no 2-injective subdirect products of $\mathsf{D}_n \times G_2 \times G_3$ *for* $n > 2$ *if* G_2 *and* G_3 *are abelian groups.*

The classification is later used in the canonization of structures with bounded dihedral colors to analyze how color classes can be connected to others. But first, we make the local automorphism groups, which form 2-injective 3-factor subdirect products, explicit.

4 Normal Forms for Structures

In this section we describe a normal form for relational structures. We sketch how it can be obtained in CPT. It is also important that we have means within CPT to translate a canonical form of the normal form back into a canonical form of the original structure. A structure $\mathcal{H}$ can be reduced to another structure $\mathcal{H}'$ *canonization preservingly* in CPT, if we can define the reduction in CPT from $\mathcal{H}$ to $\mathcal{H}'$ and if we can define a canonical copy of $\mathcal{H}$ whenever we are given $\mathcal{H}$ and a canonical copy of $\mathcal{H}'$. Formally, the reduction is between classes of structures:

▶ **Definition 6.** *A* canonization-preserving CPT-reduction *from a class of structures* $\mathcal{A}$ *to a class of structures* $\mathcal{B}$ *is a pair of CPT-interpretations* (Φ, Ψ) *with the following properties:*

- Φ *is a CPT-interpretation from* $\mathcal{A}$*-structures to* $\mathcal{B}$*-structures.*
- Ψ *is a CPT-interpretation from pairs of an* $\mathcal{A}$*-structure and an ordered* $\mathcal{B}$*-structure to ordered* $\mathcal{A}$*-structures.*
- *Given a CPT-interpretation* Θ *from* $\mathcal{B}$*-structures to ordered* $\mathcal{B}$ *structures, i.e., a CPT-definable canonization procedure, then* $\Psi((\mathcal{H}, \Theta(\Phi(\mathcal{H}))))$ *is an ordered copy of* $\mathcal{H}$ *for every* $\mathcal{A}$*-structure* $\mathcal{H}$*.*

We also say that $\mathcal{A}$ *can be reduced canonization preservingly in CPT to* $\mathcal{B}$ *if there is a canonization-preserving CPT-reduction from* $\mathcal{A}$ *to* $\mathcal{B}$*.*

As a first step, we are interested in structures those color classes cannot be refined by “local” properties:

- We call a relational structure $\mathcal{H}$ *transitive on s color classes* if for every $I \subseteq \mathbb{C}_{\mathcal{H}}$ satisfying $|I| \leq s$ the group $\mathrm{Aut}(\mathcal{H}[I])|_C$ is transitive for every $C \in I$.
- Let $\mathcal{H} = (H, R_1^{\mathcal{H}}, \ldots, R_k^{\mathcal{H}}, \preceq)$ be a structure of arity r. We call $(C_1, \ldots, C_\ell)$ the *type* of $R_i^{\mathcal{H}}$ if $R_i^{\mathcal{H}} \subseteq C_1 \times \cdots \times C_\ell$ and $C_i \neq C_j$ for all $i \neq j$. We denote with $\mathbb{T}_{\mathcal{H}}$ the set of types of all relations that have a type. We say that $\mathcal{H}$ *has typed relations* if every relation is either homogeneous or has a type.

We can define a CPT reduction from q-bounded structures to q-bounded typed structures transitive on s color classes, because the additional properties can be checked on substructures of constant size. If they are violated, we split the affected color classes and relations. Additionally, we ensure that all color classes are *regular*, i.e., their automorphism groups are regular (a color class C is regular if $|\mathrm{Aut}(\mathcal{C})| = |C|$ and $\mathrm{Aut}(\mathcal{C})$ is transitive, so has only one C-orbit). This is archived by replacing a color class C by a certain ℓ-orbit of $\mathrm{Aut}(\mathcal{C})$ (for a sufficiently large ℓ). Then we modify the relations to preserve the automorphism groups of the color classes and the connections between them. The said properties simplify the following constructions designed to gain more control on local automorphism groups.

We want to reduce certain local automorphism groups to 2-injective subdirect products. Recall from Section 3 the condition $\ker(\overline{\pi}_i(\Gamma)) = \{1\}$ for 2-injective products: If the condition is violated, we want to factor out a normal subgroup $N \lhd \mathrm{Aut}(\mathcal{C})$ (the kernel above) of a color class C. By factoring N out of C, we obtain a quotient color class:

▶ **Definition 7** (Quotient Color Class). *Let $\mathcal{H} = (H, R_1^{\mathcal{H}}, \dots, R_k^{\mathcal{H}}, \preceq)$ be a structure and the automorphism group of $C \in \mathbb{C}_{\mathcal{H}}$ be regular. Let $N \lhd Aut(\mathcal{C})$. We say that another color class C' is an* N-quotient of C *if $Aut(\mathcal{C}') \cong Aut(\mathcal{C})/N$ and there is a function $R_j^{\mathcal{H}} \subseteq C \times C'$ determining the orbit partition of N acting on C, i.e., a vertex in C' corresponds to an N orbit and the vertices of C are adjacent to its orbit vertices via $R_j^{\mathcal{H}}$. The relation $R_j^{\mathcal{H}}$ is called the* orbit-map *(of C).*

Quotient color classes can always be defined: the N-orbits on C become the vertices of the quotient color class and the orbit-map is given by containment of a C-vertex in an N-orbit.

Now we turn to structures with 2-injective subdirect products as local automorphism groups. The structures we are aiming for consist of two different kinds of color classes. The group color classes form 2-injective subdirect products. They are quotient color classes of extension color classes, which connect different group color classes via the orbit-maps (cf. Figure 4). Formally:

▶ **Definition 8.** *Let $\mathcal{H} = (H_{\mathsf{gr}} \uplus H_{\mathsf{ex}}, R_1^{\mathcal{H}}, \dots, R_k^{\mathcal{H}}, \preceq)$ be a structure, where H_{gr} and H_{ex} are unions of color classes. We call a color class $C \subseteq H_{\mathsf{gr}}$ (respectively $C \subseteq H_{\mathsf{ex}}$) a* group color class *(respectively an* extension color class*). We define the* group types *$\mathbb{T}_{\mathsf{gr}}^{\mathcal{H}} \subseteq \mathbb{T}^{\mathcal{H}}$ to be the set of all types only consisting of group color classes. Let $T = (C_1, \dots, C_j) \in \mathbb{T}_{\mathsf{gr}}^{\mathcal{H}}$. We set $\Gamma_T^{\mathcal{H}} := Aut(\mathcal{H}[\bigcup_{i\in[j]} C_i]) \leq \bigotimes_{i\in[j]} Aut(\mathcal{C}_i)$. Finally, we call $\mathcal{H}$ an* $(r-1)$-injective quotient structure *if it satisfies the following (cf. Figure 4):*

a) *$\mathcal{H}$ is of arity r, has typed relations, and all color classes are regular.*

b) *Every group color class $C \subseteq H_{\mathsf{gr}}$ is an N-quotient of exactly one extension color class $C' \subseteq H_{\mathsf{ex}}$, where $N \lhd Aut(\mathcal{C}')$, and not related to any other extension color class apart from C'. Moreover, C is only related by the orbit-map to C'.*

c) *All relations only between group color classes are of arity exactly r and every group color class occurs in only one group type.*

d) *For every $T \in \mathbb{T}_{\mathsf{gr}}^{\mathcal{H}}$ the group $\Gamma_T^{\mathcal{H}}$ is an $(r-1)$-injective r-factor subdirect product (for a straightforward generalization of 2-injective 3-factor subdirect products).*

Proof sketch of Theorem 2. We now consider a structure $\mathcal{H}$ of arity 3. First, apply the previous reductions to obtain typed structures that are transitive on 3 color classes and which have regular color classes. Then consider a type $T = (C_1, C_2, C_3) \in \mathbb{T}_{\mathcal{H}}$. Let $\Gamma_T := \mathrm{Aut}(\mathcal{H}[C_1 \cup C_2 \cup C_3])$ and $N_i^T := \ker(\overline{\pi}_1^{\Gamma_T})$ for all $i \in [3]$. Then the group $\Gamma_T/N_1^T/N_2^T/N_3^T$ is a 2-factor subdirect product. We realize this product in the structure by constructing the N_i^T-quotient color class of C_i and adjusting the relations as required. We perform this operation for all types T. The constructed quotient color classes form the group color classes of the output structure and the original ones the extension color classes (cf. Figure 2). ◀

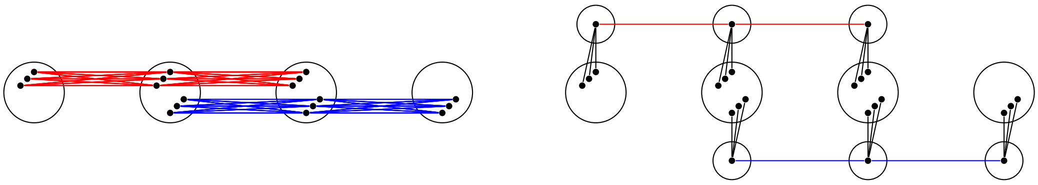


Figure 2 The situation in Theorem 2: On the left the input structure. Each circle represents one color class with drawn tuples of two relations (red and blue) between two $\Gamma_T/N_1^T/N_2^T/N_3^T$-orbits of each color class (where T is different for the red and blue relation). On the right the altered structure: For the types of the red and blue relations there are new group color classes (on the top for red and on the bottom for blue), where the orbits are contracted to a single vertex. The "old" color classes became extension color classes.

For structures of arity $r > 3$, we also need to reduce the arity. For this, we insert color classes dividing a relation into two new relations of lower arity. In particular, if the input structure is of arity 3, the automorphism group of an output color class is a section (a subgroup of a quotient) of the automorphism group of some input color class (cf. Theorem 38 in [21]). We reduce to arity 3, because arity 2 does not simplify the problem further.

5 Structures with Dihedral Colors

A structure $\mathcal{H} = (H, R_1^{\mathcal{H}}, \dots, R_k^{\mathcal{H}}, \preceq)$ *has dihedral colors* if for every $\mathcal{H}$-color class C the group $\mathrm{Aut}(\mathcal{C})$ is a dihedral or cyclic group. We allow cyclic groups to ensure closure under taking subgroups and quotients. We want to make the dihedral groups explicit as follows: For cyclic groups we require two relations only to avoid case distinctions. Regular and dihedral color classes can always be brought in standard form. In fact, it is always possible to pick two 2-orbits of the color class, that serve as the two required relations. The standard form cycles will help us later to prohibit the reflections in a color class.

6 Cyclic Linear Equation Systems in CPT

Before we begin to canonize structures with dihedral colors, we need to discuss a special class of linear equation systems. These systems are later used in the canonization procedure to encode the canonical labellings. Let V be a set of variables and $\preceq$ be a preorder on V. The *variable classes* are the $\preceq$-equivalence classes and are totally ordered by $\preceq$. A *cyclic constraint* on $W \subseteq V$ is a consistent set of linear equations over $\mathbb{Z}_q$ containing for each pair $u, v \in W$ an equation of the form $u - v = d$ for $d \in \mathbb{Z}_q$.

▶ **Definition 9** (TCES). *A* tree-like cyclic linear equation system (TCES) *over $\mathbb{Z}_q$ (q a prime power) is a tuple $(V, S, \preceq)$ with the following properties:*

- *The variable classes form a rooted tree with respect to being a direct successor in $\preceq$.*
- *S is a linear equation system on V containing for every variable class a cyclic constraint.*
- *For every constraint $\sum_i a_i u_i = d$ with $u_i \in V$ and $a_i, d \in \mathbb{Z}_q$ in S every pair of variables u_i, u_j is $\preceq$-comparable.*

In CPT, a system of linear equations S is represented by a set of constraints, which itself are encoded as sets. TCESs generalize cyclic linear equations systems (CES) from [29], where $\preceq$ must be total.

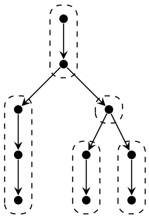


Figure 3 The variable tree (a vertex represents a variable class) of a TCES with all local components. Reordering variable classes inside a local component does not affect the tree structure.

An important operation on CES is the check for consistency. We sketch the check from [29] and its extension to TCES only very roughly, since it requires many technical details (see Section 6 in [21]). In principle, one would like to choose one variable per variable class and eliminate the others using the $u - v = d$ equations. If this was done, we would be left with a totally ordered system (in case of CESs). Of course, choosing the variables is not possible, but the system can be encoded in an equation system of hyperterms, which in some way encode all possible choices of variables and allow arithmetic manipulation. In these hyperterms, the variable classes "became the variables" for which we can apply a variant of Gaussian elimination using a certain kind of hermite normal form. This variant takes care of the issue that we are working over rings and not over fields: it reorders the variable classes based on their coefficients in the equations. Only if they have the same, the variable classes are ordered by the original order.

The translation to hyperterms can be done with TCES exactly as for CES. But the Gaussian elimination does not work anymore: Reordering a tree by the order of the coefficients does not necessarily result in a tree anymore (which causes further difficulties). To overcome this problem, we restrict to TCES, where the reordering does not harm the tree structure but only happens inside so-called local components (cf. Figure 3). Now, we can apply another variant of Gaussian elimination, which does not require a total order on the variable classes. It processes local components from leafs to the root and handles local and global variables differently. We now define local components and these two kinds of variables.

A *local component* is a maximal and (in the variable tree) connected set of variable classes, in which the tree does not branch (see Figure 3). On the local components the preorder $\preceq$ induces a tree in which every local component has degree > 1 or is a leaf. A variable is *local* if in every equation in which it occurs, i.e. it has non-zero coefficient, only variables of the same local component occur, too. Other variables are called *global*. An equation is *local* if it contains at least one local variable and *global* otherwise. For the subsequent canonization the rings $\mathbb{Z}_{2^\ell}$ will be of special interest and thus treated differently.

▶ **Definition 10.** *A TCES $\mathcal{T} = (V, S, \preceq)$ over $\mathbb{Z}_q$ is called* weakly global*, if*

- *q is a power of an odd prime and every equation (equivalently every variable) is local or*
- *$q = 2^\ell$ is a power of 2 and for every global variable $u \in V$ there is an equation $2u = 0 \in S$.*

The definition states that only values of order at most 2 are candidates for solutions of global variables. The adaption of Gaussian elimination strongly depends on this property, in particular this restriction guarantees that reordering is only required inside local components.

▶ **Theorem 11.** *Solvability of weakly global TCESs over $\mathbb{Z}_q$ is CPT-definable.*

The proof follows the same strategy as [25, 29] to solve CESs, but significant adaptations were required throughout the whole procedure.

Finally, we need to form the union of two TCESs. In general, a naive union is not a TCES anymore if the variable structures of the two systems are incompatible. For a working solution, we need the following notion: Let $\mathcal{T} = (V, S, \preceq)$ be a TCES. We say that V' are the *topmost variables* of $\mathcal{T}$ if V' is the set of all variables of the local component containing the root class of the variable tree L_r (formally $V' = \bigcup L_r$). Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two TCESs over $\mathbb{Z}_q$ which are CPT distinguishable. That is, there is a CPT term defining the ordered tuple $(\mathcal{T}_1, \mathcal{T}_2)$ or equivalently an order $\mathcal{T}_1 < \mathcal{T}_2$ (e.g. the two TCESs are defined by different CPT terms or can be ordered by their structure). If their common variables are topmost in both TCESs and whose order is compatible in them, we can define a TCES $\mathcal{T}_1 \uplus \mathcal{T}_2$ by joining them together at the topmost variables (cf. Definition 60 in[21]):

▶ **Lemma 12.** *If $\mathcal{T}_1$ and $\mathcal{T}_2$ (with variables V_i and topmost variables V_i') are compatible, then $\mathcal{T}_1 \uplus \mathcal{T}_2$ is again a TCES with topmost variables $V_1' \cup V_2'$ and it satisfies $L(\mathcal{T}_1 \uplus \mathcal{T}_2) = \bigcap_{i \in [2]} ext_{V_1 \cup V_2}(L(\mathcal{T}_i))$. If both $\mathcal{T}_1$ and $\mathcal{T}_2$ are weakly global then $\mathcal{T}_1 \uplus \mathcal{T}_2$ is weakly global, too.*

Here, $\text{ext}_{V_1 \cup V_2}(L(\mathcal{T}_i))$ denotes the set $N \subseteq \mathbb{Z}_q^{V_1 \cup V_2}$ whose projection to $\mathbb{Z}_q^{V_i}$ is equal to $L(\mathcal{T}_i)$. We write $\mathcal{T} = (\mathcal{T}_{p_1}, \dots, \mathcal{T}_{p_k})$ for a sequence of TCESs over pairwise coprime prime powers p_i and $L(\mathcal{T})$ for the solution space of $\mathcal{T}$. Lemma 12 generalizes to series of TCESs by making the assumptions of the lemma for TCESs $\mathcal{T}_{p_i}, \mathcal{T}'_{q_j}$ with p_i and q_i powers of the same prime.

7 Canonization of Structures with Dihedral Color Classes

Recall that for our canonization problem the reduction to normal forms (Theorem 2) shows that we can assume the input structure to be a dihedral 2-injective quotient structure. Our further strategy is as follows. We want to reduce canonization of dihedral 2-injective quotient structures to that of structures with abelian color classes and then apply the canonization procedure for abelian color classes. The main idea is to artificially prohibit reflections in one color class and then hope that this prohibits reflections in other color classes as well. For this, we want to exploit the classification of 2-injective subdirect products of dihedral groups (Theorems 4 and 5) saying that most 2-injective subdirect products are rotate-or-reflect groups. In particular, if we prohibit reflections in one color class of a rotate-or-reflect group then reflections in the other color classes are prohibited, too. This effect of prohibiting reflections continues through most 2-injective subdirect products and quotient color classes. However, it does not have to reach all color classes since some 2-injective subdirect product are not rotate-or-reflect groups (for example if one factor is abelian). We call the parts of the structure in which reflections are linked in this way and can only occur simultaneously *reflection components*. We analyze how reflection components can depend on each other. It will turn out, that different reflection components can indeed only be connected through abelian color classes. We call these color classes *border color classes*. Overall, we will follow a two-leveled approach: on the top level, we deal with the dependencies between the border (and all other abelian) color classes, and on the second level we consider each reflection component on its own and how it is embedded in its border color classes.

To ensure that the border color classes are indeed all abelian we have to forbid the single exception in Theorem 4, which is not a rotate-or-reflect group, namely the double CFI group.

▶ **Definition 13** (Double-CFI-Free Structure). *We call a 2-injective dihedral quotient structure* double-CFI-free*, if for every $T \in \mathbb{T}_{\mathsf{gr}}$ the group Γ_T is neither isomorphic to the double CFI group Γ_{2CFI} nor to $\Gamma_{\mathsf{2CFI}} \cap (\mathit{Rot}(\mathsf{D}_4) \times \mathsf{D}_4 \times \mathsf{D}_4)$.*

There are two natural classes of structures that are double-CFI-free after applying the preprocessing: graphs with dihedral colors and structures of arity 3 which are *odd dihedral*, that is, for every non-abelian $C \in \mathbb{C}_{\mathcal{H}}$ there is an odd k such that $\mathrm{Aut}(\mathcal{C}) \cong \mathsf{D}_k$.

7.1 Reflection Components

Let $\mathcal{H} = (H_{\mathsf{gr}} \uplus H_{\mathsf{ex}}, R_1^{\mathcal{H}}, \dots, R_k^{\mathcal{H}}, \preceq)$ be an arbitrary dihedral 2-injective double-CFI-free quotient structure. Whenever we construct a CPT term in the following, it does not depend on $\mathcal{H}$ but gets $\mathcal{H}$ as input and in particular satisfies the claimed properties for all dihedral 2-injective double-CFI-free quotient structures. We use the set $\mathbb{O} := \{\uparrow, \downarrow\}$ to denote orientations. For an orientation $o \in \mathbb{O}$ we set $\overline{o} := o'$ as the reverse orientation, so that $\mathbb{O} = \{o, o'\}$.

▶ **Definition 14** (Orientation). *We say that a structure $\mathcal{H}' = (H_{\mathsf{gr}} \uplus H_{\mathsf{ex}}, R_1^{\mathcal{H}}, \dots, R_k^{\mathcal{H}}, \preceq')$ is an* orientation *of $\mathcal{H}$ if $\preceq'$ refines $\preceq$ with the following property: Let $C \in \mathbb{C}_{\mathcal{H}}$ be a color class that is split by $\preceq'$, then Aut($\mathcal{H}[C]$) is a non-abelian dihedral group and C is split into two color classes $C^{\uparrow}$and $C^{\downarrow}$, such that each of the two classes contains one of the two oriented cycles inducing the standard form in C. We say that $\mathcal{H}'$* orients C*.*

By splitting the color class C in the above manner, we precisely prohibit the reflections in C. Because an orientation modifies only the preorder of the structure, defining an orientation of $\mathcal{H}$ is always canonization preserving. For a color class C with dihedral automorphism group we can define in CPT two orientations $\mathcal{H}_C^o$ for $o \in \mathbb{O}$ that only orient C (by the two possible orders $C^o \prec' C^{\overline{o}}$). Of course, we cannot choose one orientation canonically. But the orientation of C can canonically be propagated to other color classes in the following cases:

a) Whenever C is part of a rotate-or-reflect group (because once we cannot reflect in one component, we cannot do so in the others), and

b) whenever C' is a quotient of C (because once we remove reflections from C we can also remove remaining reflections from quotient groups).

To prove Case a) we use the classification of 2-injective subdirect products of dihedral groups (Theorems 4 and 5). We obtain an equivalence relation on the color classes: two classes are equivalent if an orientation of one color class can be propagated to the other. We define *reflection components* as the equivalence classes of said relation, which consists of color classes with dihedral automorphism group (cf. Definitions 65 and 70 in the full version).

Because all color classes of a reflection component D can be oriented by orienting only a single color class, we can speak of the two orientations of a reflection component D. We now analyze how a reflection components can be connected to another one:

▶ **Definition 15** (Color Class in Standard Form). *Let $\mathcal{H}$ be a structure and $C \in \mathbb{C}_{\mathcal{H}}$. We say that* a color class $C \in \mathbb{C}_{\mathcal{H}}$ is in standard form *if the following holds:*

- *If Aut($\mathcal{C}$) $\cong \mathsf{C}_{|C|}$ then there are relations $R_i^{\mathcal{H}}, R_j^{\mathcal{H}} \subseteq C^2$ of arity 2 each forming a directed cycle of length $|C|$ on C.*
- *Otherwise Aut($\mathcal{C}$) $\cong \mathsf{D}_{|C|/2}$ and there are two relations $R_i^{\mathcal{H}}, R_j^{\mathcal{H}} \subseteq \mathcal{C}^2$ such that $R_j^{\mathcal{H}}$ defines two directed and disjoint cycles of length $|C|/2$ and $R_i^{\mathcal{H}}$ connects them by a perfect matching such that the two cycles are directed into opposite directions (cf. Figure 4).*

We say that the relations $R_i^{\mathcal{H}}$ and $R_j^{\mathcal{H}}$ induce the standard form of $\mathcal{C}$*. The* color classes of $\mathcal{H}$ are in standard form*, if every color class is in standard form.*

▶ **Definition 16** (Border Color Class). *Let $D \subseteq \mathbb{C}_{\mathcal{H}}$ be a reflection component. We call a color class $C \in \mathbb{C}_{\mathcal{H}}$* a border color class of D *if $C \notin D$ and C is related to a color class contained in D. We denote with $B(D)$ the set of all border color classes of D.*

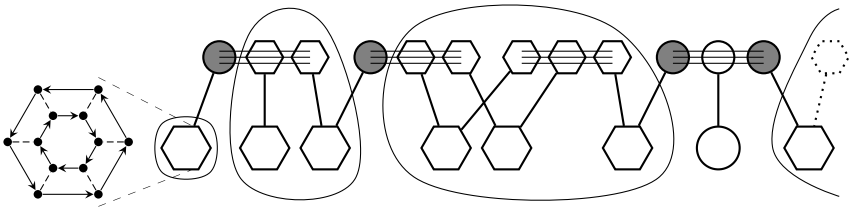


Figure 4 A 2-injective quotient structure with dihedral colors: an abelian color class is drawn as circle, a non-abelian one as hexagon. The group color classes are at the top, the extension classes at the bottom. An edge between a group class C and an extension class C' denotes an orbit-map and C is a quotient of C'. Edges between group color classes indicate relations of arity 3. The reflection components are encircled and border color classes are gray. On the left a dihedral color class in standard form with automorphism group D_6.

▶ **Lemma 17.** *Let $D \subseteq \mathbb{C}_{\mathcal{H}}$ be a reflection component and $C \in B(D)$ a border color class of D. Then $Aut(\mathcal{C})$ is isomorphic to one of $\{\mathsf{C}_1, \mathsf{C}_2, \mathsf{D}_2\}$ and C is a group color class.*

This lemma is also proven using Theorems 4 and 5. So the border color classes of a reflection component D are all abelian group color classes. That is, the reflection components are embedded in a global abelian part of the structure (an example is shown in Figure 4). We define $\mathcal{H}_D := \mathcal{H}[B(D) \cup \bigcup D]$ and denote the two CPT-definable (abelian) orientations of $\mathcal{H}_D$ with $\mathcal{H}_D^o$, $o \in \mathbb{O}$. Let $\mathsf{can}(\mathcal{H}_D^o)$ be canonizations for all $o \in \mathbb{O}$. We denote with $\overline{\mathsf{can}}(\mathcal{H}_D^o)$ the structure obtained from $\mathsf{can}(\mathcal{H}_D^o)$ by undoing the orientation. Then $\mathcal{H}_D \cong \overline{\mathsf{can}}(\mathcal{H}_D^o)$. Let $<$ be the lexicographical order on canonizations. We define the canonization $\mathsf{can}(\mathcal{H}_D)$ to be the $<$-minimal canonization $\overline{\mathsf{can}}(\mathcal{H}_D^o)$ with $o \in \mathbb{O}$. We analyze the canonical labellings of D.

▶ **Lemma 18.** *If $\mathsf{can}(\mathcal{H}_D^o) < \mathsf{can}(\mathcal{H}_D^{\overline{o}})$, then $\mathrm{Iso}(\mathcal{H}_D, \mathsf{can}(\mathcal{H}_D)) = \mathrm{Iso}(\mathcal{H}_D^o, \mathsf{can}(\mathcal{H}_D^o))$.*

▶ **Lemma 19.** *If $\mathsf{can}(\mathcal{H}_D^{\uparrow}) = \mathsf{can}(\mathcal{H}_D^{\downarrow})$, then $\mathrm{Iso}(\mathcal{H}_D, \mathsf{can}(\mathcal{H}_D)) = \bigcup_{o \in \mathbb{O}} \mathrm{Iso}(\mathcal{H}_D^o, \mathsf{can}(\mathcal{H}_D^o))$.*

7.2 Canonizing Abelian Structures

Our canonization procedure strongly depends on the canonization procedure for q-bounded structured with *abelian* color classes. This procedure not only outputs a canonization, but also a CES *encoding* the canonical labellings. The automorphism group of a color class is decomposed into a direct sum of cyclic groups, which are used to define variables and cyclic constraints for this color class. In particular, if the automorphism group of a color class is the direct product of cyclic groups of prime power order q, then all variables for this color class range over $\mathbb{Z}_q$. A formal statement of the canonization procedure and the notion of encoding isomorphisms can be found in [29] and in Theorem 81 in the full version of this paper. It is also possible to start the canonization procedure for abelian color classes with a TCES that encodes an initial set of allowed labellings (Lemma 83 in [21]).

7.3 Canonization Procedure

For dihedral colors we want to maintain an equation system encoding all canonical labellings of all abelian color classes (and hence including all border color classes) that extend to canonical labellings of the input structure. This suffices to encode the dependencies between different reflection components because – as we have seen in the previous section – they can only be connected via abelian color classes. As initialization step, we apply the canonization procedure for abelian colors to all abelian color classes. Then we want to inductively add one

```
1  Compute the set $A \subseteq \mathbb{C}_{\mathcal{H}}$ of abelian color classes;
2  Compute all reflection components $D_1 < \cdots < D_m$ of $\mathcal{H}$;
3  Compute $\mathsf{can}(\mathcal{H}_0) := \mathsf{can}(\mathcal{H}[A])$ and $\Phi_0 := \mathrm{Iso}(\mathcal{H}[A], \mathsf{can}(\mathcal{H}[A]))$ using the
     canonization procedure for abelian colors;
4  for $i \in [m]$ do
5      Set $D := D_i$ and define the two orientations $\mathcal{H}_D^o$;
6      Compute $\mathsf{can}(\mathcal{H}_D^o)$ and $\Phi^o := \mathrm{Iso}(\mathcal{H}_D^o, \mathsf{can}(\mathcal{H}_D^o))$ such that $\Phi_{i-1} \cap \mathrm{ext}_A(\Phi^o) \neq \emptyset$
         with the canonization procedure for abelian colors;
7      if $\mathsf{can}(\mathcal{H}_D^o) < \mathsf{can}(\mathcal{H}_D^{\bar{o}})$ for some $o \in \mathbb{O}$ then
8          $\mathsf{can}(\mathcal{H}_i) := \mathsf{can}(\mathcal{H}_{i-1}) \cup \overline{\mathsf{can}}(\mathcal{H}_D^o)$;
9          $\Phi_i := \Phi_{i-1} \cap \mathrm{ext}_A(\Phi^o|_A)$;
10     else
11         $\mathsf{can}(\mathcal{H}_i) := \mathsf{can}(\mathcal{H}_{i-1}) \cup \overline{\mathsf{can}}(\mathcal{H}_D^{\uparrow}) \cup \overline{\mathsf{can}}(\mathcal{H}_D^{\downarrow})$;
12         $\Phi_i := \Phi_{i-1} \cap \mathrm{ext}_A((\Phi^{\uparrow} \cup \Phi^{\downarrow})|_A)$;
13 $\mathsf{can}(\mathcal{H}) := \mathsf{can}(\mathcal{H}_m)$;
```

Figure 5 Canonizing a 2-injective double-CFI-free structure $\mathcal{H}$ with dihedral colors in CPT.

reflection component in each step (possibly restricting the canonical labellings of the border color classes). To do so, we want to define a canonical copy of the reflection component D by taking the existing partial canonization into account. That is, given an equation system encoding all canonical labellings of the partial canonization computed so far, we want to increase both, the equation system and the canonization, by D in one step.

From now, we assume that the abelian color classes of a structure $\mathcal{H}$ are smaller than the non-abelian ones (according to $\preceq$). The canonization procedure is given in Figure 5, where we use $\mathrm{ext}_A(\Phi)$ as shorthand for $\mathrm{ext}_{\bigcup A}(\Phi)$. We fix the input structure $\mathcal{H} = (H, R_1^{\mathcal{H}}, \dots R_k^{\mathcal{H}}, \preceq)$ in the following (again, our CPT terms will not depend on $\mathcal{H}$). The algorithm maintains canonizations $\mathsf{can}(\mathcal{H}_i)$ of $\mathcal{H}_i := \mathcal{H}[A \cup \bigcup_{j \in [i]} D_j]$ and sets Φ_i of canonical labellings.

▶ **Lemma 20.** *For $i \leq m$ the following holds: $\mathcal{H}_i \cong \mathsf{can}(\mathcal{H}_i)$ and $\Phi_i = \mathrm{Iso}(\mathcal{H}_i, \mathsf{can}(\mathcal{H}_i))|_A$.*

Proof Sketch. The canonization procedure for abelian color classes yields the desired set of canonical labellings. By Lemmas 18 and 19 the canonical labellings of the (unoriented) reflection component are computed correctly. ◀

We cannot compute with the sets Φ_i directly in CPT because they can be exponentially large. So we encode them with sequences of weakly global TCESs $\mathcal{T}_i$. We maintain that the variables V_A of the abelian color classes $A \subseteq \mathbb{C}_{\mathcal{H}}$ are contained in the topmost variables of the $\mathcal{T}_i$ and thus the occurring TCESs will all be compatible. With what we have seen so far, the canonization procedure can be expressed in CPT apart one exception in Line 12: We have to show how to define a TCES encoding $\mathrm{ext}_A((\Phi^{\uparrow} \cup \Phi^{\downarrow})|_A) = \mathrm{ext}_A(\mathrm{Iso}(\mathcal{H}_D, \mathsf{can}(\mathcal{H}_D))|_{B(D)})$.

7.4 Equation Systems for Reflection Components

Let $\mathcal{T} := \mathcal{T}_{i-1}$ for some $1 < i \leq m$ be in the series of weakly global TCESs for the canonization constructed so far, $D := D_i$ be the next reflection component to canonize (cf. Figure 5), and $\mathsf{can}(\mathcal{H}_D^{\uparrow}) = \mathsf{can}(\mathcal{H}_D^{\downarrow})$. Let $\mathcal{S}^o$ be the series of CESs encoding the sets $\Phi^o = \mathrm{Iso}(\mathcal{H}_D^o, \mathsf{can}(\mathcal{H}_D^o))$ and the variables of the (abelian) border color classes of D be $B = B_1 < \cdots < B_k$. These variables are equal for $\mathcal{S}^{\uparrow}$ and $\mathcal{S}^{\downarrow}$ and are contained in the topmost variables V_A of $\mathcal{T}$.

We cannot fix an isomorphism in $\mathrm{Iso}(\mathcal{H}_D^{\uparrow},\mathcal{H}_D^{\downarrow}) = \mathrm{Iso}(\mathcal{H}_D^{\downarrow},\mathcal{H}_D^{\uparrow})$ canonically, but one isomorphism contained in $\mathrm{Iso}(\mathcal{H}_D^{\uparrow},\mathcal{H}_D^{\downarrow})|_{B(D)}$: Note that by Lemma 17 the border color classes have automorphism groups C_2^{ℓ} for $\ell \in \{0,1,2\}$. Hence, all variables for the border color classes range over $\mathbb{Z}_2$. We rename the variables of the two series of CESs, such that both use different variables, but we still can identify a variable of a border color class of $\mathcal{S}^o$ with a variable of a border color class of $\mathcal{S}^{\overline{o}}$. Hence, for two vectors $x^o \in \mathbb{Z}_2^{B_i^o}$ we still can write $x^{\uparrow} = x^{\downarrow}$. We denote with V^o (and B^o respectively) the changed variables for $o \in \mathbb{O}$.

▶ **Lemma 21.** *There is a CPT term defining two vectors $x^o = (x_1^o,\dots,x_k^o) \in \mathbb{Z}_2^{B_1^o} \times \dots \times \mathbb{Z}_2^{B_k^o}$ for both $o \in \mathbb{O}$ such that if $y^o \in L(\mathcal{S}^o)$, then there is a $y^{\overline{o}} \in L(\mathcal{S}^{\overline{o}})$ such that $y^o|_{B^o} + x^o = y^{\overline{o}}|_{B^{\overline{o}}}$.*

Proof Sketch. Assume we have defined x^o for the first i border color classes. We define a TCES that is consistent if and only if there are $y^o \in L(\mathcal{S}^o)$ for both $o \in \mathbb{O}$ that have different values for exactly the B_j ($j \in [i+1]$) with $j = i+1$ or $j \leq i$ and $x^o(u) = 1$ for all $u \in B_j$. If the TCES is consistent, we set all entries for B_{i+1} in x^o to 1 and otherwise to 0. ◀

We now use the vectors x^o to represent the canonical labellings of the border color classes, which additionally extend to canonical labellings of the reflection component, as a TCES.

▶ **Lemma 22.** *There is a CPT term defining a series of weakly global TCESs $\mathcal{T}_D$ with the following properties: B is contained in the topmost variables of $\mathcal{T}_D$, $\mathcal{T}_D$ encodes the set $\mathrm{Iso}(\mathcal{H}_D, \mathsf{can}(\mathcal{H}_D))|_{B(D)}$, and the size of $\mathcal{T}_D$ is polynomial in $|D|$.*

Proof Sketch. Let x^o be the two vectors given by Lemma 21. We define a set of two variables $B_\alpha := \{\alpha^{\uparrow},\alpha^{\downarrow}\}$ (and set $\alpha^o := \mathcal{H}_D^o$), $V_D := B \cup B_\alpha \cup V^{\uparrow} \cup V^{\downarrow}$, and $\preceq_D$ such that it respects the orders on B and V^o and $B \prec B_\alpha \prec V^o$ for all $o \in \mathbb{O}$. The variable sets $V^{\uparrow}$ and $V^{\downarrow}$ are incomparable. We want to define a TCES $\mathcal{T}_D$ enforcing that if $z \in L(\mathcal{T}_D)$, then there is an $o \in \mathbb{O}$ and a $y^o \in L(\mathcal{S}^o)$ such that $z = y^o|_B$. To do so, we guess two solutions $y^o \in L(\mathcal{S}^o)$ (one for each $o \in \mathbb{O}$) with the property that $y^o|_B + x^o = y^{\overline{o}}|_B$ (Lemma 21). Then we want to ensure that $z = y^{\uparrow}|_B$ or $z = y^{\downarrow}|_B$. To allow that one equality does not hold, we use the additional variables $\alpha^{\uparrow}$ to express the constraints $z = y^o|_B + \alpha^o \cdot x^o$. By enforcing that exactly one of $\alpha^{\uparrow}$ and $\alpha^{\downarrow}$ is 1, we obtain the desired system. Finally, to make the system linear, we encode the multiplication $\alpha^o \cdot x^o$. This is possible, because x^o does not depend on y^o and can be defined before defining the following TCES:

$$\begin{aligned} & y^{\uparrow} \in L(\mathcal{S}^{\uparrow}), \quad y^{\downarrow} \in L(\mathcal{S}^{\downarrow}), \quad 1 = \alpha^{\uparrow} + \alpha^{\downarrow} && \\ & z(u) = y^{\uparrow}(u) = y^{\downarrow}(u) && \text{if } x^{\uparrow}(u) = x^{\downarrow}(u) = 0, u \in B \\ & z(u) = y^{\uparrow}(u) + \alpha^{\uparrow} = y^{\downarrow}(u) + \alpha^{\downarrow} && \text{if } x^{\uparrow}(u) = x^{\downarrow}(u) = 1, u \in B \end{aligned}$$

where y^o is indexed by V^o and z is indexed by B and ranges over $\mathbb{Z}_2$. If the variable α^o is assigned to 1, then $z = y^o|_{B^o} + x^o$ and $z = y^o|_{B^o}$ otherwise. Because of the cyclic constraint $1 = \alpha^{\uparrow} + \alpha^{\downarrow}$, we add the vector x^o to a solution y^o of $\mathcal{S}^o$ for exactly one orientation $o \in \mathbb{O}$. One verifies the construction with Lemmas 19 and 21. ◀

Now, we defined all operation on TCESs needed and conclude:

▶ **Theorem 23.** *Canonization of 2-injective double-CFI-free q-bounded gadget quotient structures is CPT-definable.*

This **proves Theorem 1**, because the involved classes of structures are double-CFI-free after applying Theorem 2. In particular CPT captures PTIME on these classes.

8 Conclusion

We separated a relational structure into 2-injective subdirect products and quotients, gave a classification of all 2-injective subdirect products of dihedral and cyclic groups, and used this classification to canonize relational structures with bounded dihedral colors of arity at most 3. We showed that the structure decomposes into reflection components and that in these components either all color classes have to be reflected or none. If we exclude a single 2-injective subdirect product, namely the double CFI group, the reflection components can only have abelian dependencies. This is always true for graphs, because the said group cannot be realized by graphs with dihedral colors. In fact, we demonstrated the increase of complexity when considering structures of arity 3 instead of 2. Apart from the fact that the double CFI group does not appear, a classification of 1-injective 2-factor subdirect products of dihedral groups is much easier. Considering higher arity, already 3-injective 4-factor subdirect products of dihedral groups cannot be classified to be (almost) abelian or reflect-or-rotate groups. If one instead tries to reduce the arity of the structures, one needs not only to work with a class of groups closed under taking quotients and subgroups (which is the case for dihedral and cyclic groups), but also closed under taking direct products.

One natural way to exclude the double CFI group is a restriction to odd dihedral colors. The difficulty with even dihedral groups might indicate that looking at odd (non-dihedral) groups could be a reasonable next step. A natural graph class with odd automorphism groups are tournaments. Since such groups are solvable there is hope for an inductive approach exploiting the abelian case. It could be possible that the techniques developed in this paper transfer to this case. Just like dihedral groups, odd groups are closed under taking quotients and subgroups. However, they are also closed under direct products (and are solvable), which would allow a reduction of the arity. Thus, it is possible to apply our reduction to quotients and 2-injective groups. As a next step, one could try to follow a similar strategy as for dihedral colors: identify components of the graph, in which the complexity of all color classes decreases simultaneously, when a single color class is made easier (similar to reflection components). This might not immediately result in abelian groups, but recursion on the complexity of the groups could be a reasonable option, e.g. on the length of the composition series or on the nilpotency class. All the mentioned avenues remain as future work.

References

1 László Babai. Monte carlo algorithms in graph isomorphism testing. Technical Report D.M.S. No. 79-10, Université de Montréal, 1979.

2 László Babai and Eugene M. Luks. Canonical labeling of graphs. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 171–183. ACM, 1983. doi:10.1145/800061.808746.

3 Andreas Blass, Yuri Gurevich, and Saharon Shelah. Choiceless polynomial time. *Ann. Pure Appl. Log.*, 100(1-3):141–187, 1999. doi:10.1016/S0168-0072(99)00005-6.

4 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992. doi:10.1007/BF01305232.

5 Ashok K. Chandra and David Harel. Structure and complexity of relational queries. *J. Comput. Syst. Sci.*, 25(1):99–128, 1982. doi:10.1016/0022-0000(82)90012-5.

6 Anuj Dawar, David Richerby, and Benjamin Rossman. Choiceless polynomial time, counting and the Cai-Fürer-Immerman graphs. *Ann. Pure Appl. Logic*, 152(1-3):31–50, 2008. doi:10.1016/j.apal.2007.11.011.

7 Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of computation (Proc. SIAM-AMS Sympos. Appl. Math., New York, 1973)*, pages 43–73. SIAM–AMS Proc., Vol. VII, 1974.

8 Merrick L. Furst, John E. Hopcroft, and Eugene M. Luks. Polynomial-time algorithms for permutation groups. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 36–41. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.34.

9 Erich Grädel and Martin Grohe. Is polynomial time choiceless? In Lev D. Beklemishev, Andreas Blass, Nachum Dershowitz, Bernd Finkbeiner, and Wolfram Schulte, editors, *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, volume 9300 of *Lecture Notes in Computer Science*, pages 193–209. Springer, 2015. doi:10.1007/978-3-319-23534-9_11.

10 Martin Grohe. Isomorphism testing for embeddable graphs through definability. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 63–72. ACM, 2000. doi:10.1145/335305.335313.

11 Martin Grohe. The quest for a logic capturing PTIME. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 267–271. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.11.

12 Martin Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 179–188. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.22.

13 Martin Grohe. *Descriptive Complexity, Canonization, and Definable Graph Structure Theory*. Cambridge University Press, 2017.

14 Martin Grohe and Julian Mariño. Definability and descriptive complexity on databases of bounded tree-width. In Catriel Beeri and Peter Buneman, editors, *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*, volume 1540 of *Lecture Notes in Computer Science*, pages 70–82. Springer, 1999. doi:10.1007/3-540-49257-7_6.

15 Martin Grohe and Daniel Neuen. Canonisation and definability for graphs of bounded rank width. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785682.

16 Martin Grohe, Pascal Schweitzer, and Daniel Wiebking. Deep Weisfeiler Leman, 2020. arXiv:arXiv:2003.10935.

17 Yuri Gurevich. Logic and the challenge of computer science. In Egon Boerger, editor, *Current Trends in Theoretical Computer Science*, pages 1–57. Computer Science Press, 1988.

18 Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987. doi:10.1137/0216051.

19 Sandra Kiefer and Daniel Neuen. The power of the Weisfeiler-Leman algorithm to decompose graphs. In *MFCS*, volume 138 of *LIPIcs*, pages 45:1–45:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

20 Bastian Laubner. Capturing polynomial time on interval graphs. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 199–208. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.42.

21 Moritz Lichter and Pascal Schweitzer. Canonization for bounded and dihedral color classes in choiceless polynomial time, 2020. arXiv:arXiv:2010.12182.

22 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982.

23 Daniel Neuen and Pascal Schweitzer. Subgroups of 3-factor direct products. *Tatra Mt. Math. Publ.*, 73:19–38, 2019.

24 Martin Otto. *Bounded variable logics and counting - a study in finite models*, volume 9 of *Lecture Notes in Logic*. Springer, 1997.

25 Wied Pakusa. *Linear Equation Systems and the Search for a Logical Characterisation of Polynomial Time*. PhD thesis, RWTH Aachen, 2015.

26 Wied Pakusa, Svenja Schalthöfer, and Erkal Selman. Definability of Cai-Fürer-Immerman problems in choiceless polynomial time. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 19:1–19:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.19.

27 Benjamin Rossman. Choiceless computation and symmetry. In Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, editors, *Fields of Logic and Computation, Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume 6300 of *Lecture Notes in Computer Science*, pages 565–580. Springer, 2010. doi:10.1007/978-3-642-15025-8_28.

28 Pascal Schweitzer and Daniel Wiebking. A unifying method for the design of algorithms canonizing combinatorial objects. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1247–1258. ACM, 2019. doi:10.1145/3313276.3316338.

29 Faried Abu Zaid, Erich Grädel, Martin Grohe, and Wied Pakusa. Choiceless polynomial time on structures with small abelian colour classes. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 50–62. Springer, 2014. doi:10.1007/978-3-662-44522-8_5.

Preservation Theorems Through the Lens of Topology

Aliaume Lopez
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LSV, Gif-sur-Yvette, France
aliaume.lopez@ens-paris-saclay.fr

Abstract

In this paper, we introduce a family of topological spaces that captures the existence of preservation theorems. The structure of those spaces allows us to study the relativisation of preservation theorems under suitable definitions of surjective morphisms, subclasses, sums, products, topological closures, and projective limits. Throughout the paper, we also integrate already known results into this new framework and show how it captures the essence of their proofs.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Mathematics of computing → Discrete mathematics; Mathematics of computing → Point-set topology

Keywords and phrases Preservation theorem, Pre-spectral space, Noetherian space, Spectral space

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.32

Related Version A full version of the paper is available at `https://arxiv.org/abs/2007.07879`.

Acknowledgements I thank Jean Goubault-Larrecq and Sylvain Schmitz for their help and support in writing this paper.

1 Introduction

In classical model theory, *preservation theorems* characterise first-order definable sets enjoying some semantic property as those definable in a suitable syntactic fragment [6, Section 5.2]. A well-known instance is the Łoś-Tarski Theorem [37, 28]: a first-order sentence φ is preserved under extensions on all structures – i.e., $A \models \varphi$ and A is an induced substructure of B imply $B \models \varphi$ – if and only if it is equivalent to an existential sentence.

A major roadblock for applying these results in computer science is that preservation theorems generally do not relativise to classes of structures, and in particular to the class of all finite structures (see the discussions in [31, Section 2] and [25, Section 3.4]). In fact, the only case where a classical preservation theorem was shown to hold on all finite structures is Rossman's Theorem [32]: a first-order sentence is preserved under homomorphisms on all finite structures if and only if it is equivalent to an existential positive sentence. This long-sought result has applications in database theory, where existential positive formulæ correspond to unions of conjunctive queries (also known as select-project-join-union queries and arguably the most common database queries in practice [1]). For instance, it is related in [12, Theorem 17] to the existence of homomorphism-universal models (as constructed by chase algorithms) for databases with integrity constraints, in [38, Theorem 3.4] to a characterisation of schema mappings definable via source-to-target tuple-generating dependencies, and in [18, Corollary 4.14] to the naïve evaluation of queries over incomplete databases under open-world semantics. These applications would benefit directly from preservation theorems for more restricted classes of finite structures or for other semantic properties – corresponding to other classes of queries and other semantics of incompleteness – and this has been an active area of research [5, 4, 9, 22, 17]. Like Rossman's result, these proofs typically rely on careful model-theoretic arguments – typically using Ehrenfeucht-Fraïsse games and locality – and each new attempt at proving a preservation theorem seemingly needs to restart from scratch.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 32; pp. 32:1–32:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we develop a general topological framework for investigating preservation theorems, where preservation theorems, both old and new, can be obtained as byproducts of topological constructions.

As pointed out in the literature, the classical proofs of preservation theorems fail in the finite because the Compactness Theorem does not apply. As we will see in Section 2, one can reinterpret in topological terms the two applications of the Compactness Theorem in the classical proofs of preservation theorems like the Łoś-Tarski Theorem. Here, the topology of interest has the sets of structures closed under extension as its open sets, and one application of the Compactness Theorem shows that the definable open sets are compact (Claim 2.2) while the other shows that the sets definable by existential sentences form a base for the definable open sets (Claim 2.1). In Section 3, we capture these two ingredients in general through the definitions of *logically presented pre-spectral spaces* and *diagram bases* which lead to a generic preservation theorem (Theorem 3.4): under mild hypotheses – which are met in all the preservation results over classes of finite structures in the literature – preservation holds if and only if the space under consideration is logically presented pre-spectral.

The benefit of this abstract, topological viewpoint, is that preservation results can now be proven by constructing new logically presented pre-spectral spaces from known ones.

Here, the topological core of our definition is the one of *pre-spectral* spaces, which generalise both Noetherian spaces and spectral spaces [19, 13]; see Section 4. From this point onwards the use of the word *stability* will always be used to describe *closure under some operations* and will never be used as the Model Theoretic notion of *stability*, this choice is motivated by the fact that in topology *closure* has a specific meaning, and we already are using the word *preservation* to describe *preservation theorems*. To some extent, we can rely on the stability of spectral spaces under various topological constructions to investigate the same constructions for pre-spectral spaces. We focus however in the paper on the *logically presented* pre-spectral spaces, which is where the main difficulty lies when attempting to prove preservation over classes of finite structures, and for which stability must take the logical aspect into account. Accordingly, Section 5 shows the stability of logically presented pre-spectral spaces under typical constructions: under a carefully chosen notion of morphisms, under subclasses provided a sufficient condition is met, and under finite sums and finite products.

Where the topological viewpoint really shines is when it comes to stability for various kinds of "limits" of classes of structures enjoying a preservation property. We show in Section 6 that the limit of a *single* class of structures, when it can be construed as the *closure* in a suitable topology of a logically presented pre-spectral space, is also logically presented pre-spectral. This allows us to show that Rossman's Theorem – i.e., homomorphism preservation in the finite – extends to the class of structures with the finite model property, and also extends to countable unions of finite structures (the latter was also shown in [30, Chapter 10]). In Section 7, we show that the limit of a *family* of pre-spectral spaces, when built as a *projective limit*, is also pre-spectral. We use this to show that Rossman's proof of homomorphism preservation in the finite can be re-cast in our framework as building exactly such a projective limit.

Due to space constraints, detailed proofs and additional examples will be found in the full paper.

2 Preservation Theorems

In this section, we revisit classical preservation theorems, whose proofs can be found in many books such as [6, Section 5.2]. We will recall the needed definitions, and illustrate the proof techniques in order to highlight the two ingredients that motivate our definitions of pre-spectral spaces and diagram bases later in Section 3.

2.1 Classical Preservation Theorems

Notations. A σ-structure A over a finite relational signature σ (without constants) is given by a domain $|A|$ and, for each symbol $R \in \sigma$ of arity n, a relation $\mathbf{R}^A \subseteq |A|^n$; A is finite if $|A|$ is finite. The binary symbol "$=$" will always be interpreted as equality, and will not be explicitly listed in our signatures. We write $\mathrm{Struct}(\sigma)$ for the set[1] of all the σ-structures and $\mathrm{Fin}(\sigma)$ for the finite ones. We assume the reader is familiar with the syntax and semantics of first-order logic over σ. We write $\mathsf{FO}[\sigma]$ for the set of first-order sentences over σ. For such a sentence φ, we write $[\![\varphi]\!]_X \triangleq \{A \in X \mid A \models \varphi\}$ for its set of models over a class of structures $X \subseteq \mathrm{Struct}(\sigma)$; by extension, we let $[\![\mathsf{F}]\!]_X \triangleq \{[\![\varphi]\!]_X \mid \varphi \in \mathsf{F}\}$ denote the collection of F-definable subsets of X for a fragment F of $\mathsf{FO}[\sigma]$.

Abstract Preservation. A preservation theorem over a class of structures $X \subseteq \mathrm{Struct}(\sigma)$ shows that first-order sentences enjoying some semantic property are equivalent to sentences from a suitable a syntactic fragment. More precisely, one can model a semantic property as a collection $\mathcal{O} \subseteq \wp(X)$ of "semantic observations" and consider a fragment $\mathsf{F} \subseteq \mathsf{FO}[\sigma]$: we will say that X has the $(\mathcal{O}, \mathsf{F})$ *preservation property* if

1. for all $\psi \in \mathsf{F}$, $[\![\psi]\!]_X \in \mathcal{O}$, and,
2. for all $\varphi \in \mathsf{FO}[\sigma]$ such that $[\![\varphi]\!]_X \in \mathcal{O}$, there exists $\psi \in \mathsf{F}$ such that $[\![\varphi]\!]_X = [\![\psi]\!]_X$.

In this definition, item 1 is usually proven by a straightforward induction on the formulæ in F, and the challenge is to establish item 2. Item 2 is also where *relativisation* to a subset $Y \subseteq X$ might fail, because a set $U \notin \mathcal{O}$ might still be such that $U \cap Y \in \{V \cap Y \mid V \in \mathcal{O}\}$, and thus there might be new first-order sentences enjoying the semantic property and requiring an equivalent sentence in F.

Put more succinctly, X has the $(\mathcal{O}, \mathsf{F})$ preservation property if

$$\mathcal{O} \cap [\![\mathsf{FO}[\sigma]]\!]_X = [\![\mathsf{F}]\!]_X \ . \tag{1}$$

This formulation explicitly shows how a semantic condition (the left-hand side in (1)) is matched with a syntactic one (the right-hand side). As preservation is of interest beyond first-order logic [20, 15, 10, 17], we will say in full generality that a set X equipped with a lattice $\mathcal{L}$ of sets definable in the logic of interest has the $(\mathcal{O}, \mathcal{L}')$ *preservation property* if

$$\mathcal{O} \cap \mathcal{L} = \mathcal{L}' \tag{2}$$

In the rest of this paper we will assume that $\mathcal{O}$ contains $\emptyset$, contains X, is closed under finite intersections and arbitrary unions. This is equivalent to $\mathcal{O}$ being a collection of open sets and defining a *topology* on X.

[1] In order to work over sets instead of proper classes and thereby avoid delicate but out-of-topic foundational issues, every σ-structure in this paper will be assumed to be of cardinality bounded by some suitable infinite cardinal. In particular, the Löwenheim-Skolem Theorem justifies that this is at no loss of generality when working with first-order logic.

Table 1 Classical preservation theorems and their relativisations to the finite case.

preservation theorem	**quasi-ordering $\leq$**	**fragment F**	**holds in Fin(σ)**
homomorphism	$\rightarrow$	EPFO	yes [32]
Tarski-Lyndon	$\subseteq$	EPFO$^{\neq}$	no [3]
Łoś-Tarski	$\subseteq_i$	EFO	no [36, 21, 11]
dual Lyndon	$\twoheadleftarrow$	NFO	no [2, 34]

Monotone Preservation. In a number of cases, which are especially relevant in the applications to database theory mentioned in the introduction [12, 18], the semantic property of interest is a form of monotonicity for some quasi-ordering $\leq$ of Struct(σ). We say that a sentence φ is *monotone* in $X \subseteq$ Struct(σ) if $[\![\varphi]\!]_X$ is *upwards-closed*, meaning that if $A \in [\![\varphi]\!]_X$ and B is a σ-structure in X such that $A \leq B$, then $B \in [\![\varphi]\!]_X$. In terms of abstract preservation, this corresponds to choosing $\mathcal{O}$ as the collection of upwards-closed subsets of X, which is also known as the *Alexandroff topology* and is denoted by $\tau_{\leq}$.

The quasi-ordering $\leq$ in question is typically defined through some class of homomorphisms. Recall that there is a *homomorphism* between two σ-structures A and B, noted $A \rightarrow B$, if there exists $f\colon |A| \rightarrow |B|$ such that, for all relation symbols R of σ and all tuples $(a_1, \ldots, a_n) \in \mathbf{R}^A$, $(f(a_1), \ldots, f(a_n)) \in \mathbf{R}^B$. When f is injective, this entails that A is (isomorphic to) a (not necessarily induced) *substructure* of B and we write $A \subseteq B$; when f is furthermore *strong* – meaning that for all R and $(a_1, \ldots, a_n) \in |A|^n$, $(f(a_1), \ldots, f(a_n)) \in \mathbf{R}^B$ implies $(a_1, \ldots, a_n) \in \mathbf{R}^A$ – , this entails that A is (isomorphic to) an *induced substructure* of B and we write $A \subseteq_i B$; finally, we write $A \twoheadrightarrow B$ when f is surjective.

Table 1 summarises what is known about monotone preservation theorems. In this table, EFO denotes the set of existential first-order sentences, NFO the set of negative ones (namely negative atoms closed under $\vee$, $\wedge$, $\exists$, and $\forall$), EPFO the set of existential positive ones, and EPFO$^{\neq}$ the set of existential positive ones extended with atoms of the form $x \neq y$ (interpreted as inequality). Note that Lydon's Theorem, which states that a first-order sentence closed under surjective homomorphisms on all structures is equivalent to a positive one, is presented in Table 1 in its *dual* form with inverse surjective homomorphisms and negative sentences. For all these fragments F and associated quasi-orderings $\leq$, the fact that $[\![\mathsf{F}]\!]_X \subseteq \tau_{\leq}$ is mostly straightforward.

2.2 The Łoś-Tarski Theorem in Topological Terms

We propose now to inspect the proof of the Łoś-Tarski Theorem on a finite relational signature σ, as found for instance in [6, Theorem 3.2.2] or [24, Section 5.4]. We work here with the collection $\mathcal{O} \triangleq \tau_{\subseteq_i}$ of upwards-closed subsets of $X \triangleq$ Struct(σ) for $\subseteq_i$ (this is the Alexandroff topology of the quasi-order $\subseteq_i$) and the fragment F $\triangleq$ EFO[σ]. The Łoś-Tarski Theorem corresponds to the following instantiation of (1):

$$\tau_{\subseteq_i} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} = [\![\mathsf{EFO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} . \tag{3}$$

The proof of the Łoś-Tarski Theorem can be decomposed into two steps, here corresponding to the upcoming claims 2.1 and 2.2, and each invoking the Compactness Theorem. When translated in topological terms, the first shows that EFO defines a *base* for the definable open sets, while the second shows that definable open sets are *compact*.

"Syntactic" Base. Recall that a *base* $\mathcal{B}$ of a topology τ is a collection of open sets such that every open set of τ is a (possibly infinite) union of elements from $\mathcal{B}$. Equivalently, $\mathcal{B}$ is a base of a topology τ whenever $\forall U \in \tau, \forall A \in U, \exists V \in \mathcal{B}, A \in V \subseteq U$. A *subbase* is a collection of open sets such that every open set of τ is a (possibly infinite) union of finite intersections of elements of the subbase. The topology $\langle \mathcal{O} \rangle$ *generated* by a collection $\mathcal{O}$ of sets is the smallest topology containing those sets; $\mathcal{O}$ is then a subbase of $\langle \mathcal{O} \rangle$.

We first prove a weaker version of Equation (3) by proving the equality on the generated topologies. Because $[\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)}$ and $[\![\mathsf{EFO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)}$ are lattices, those generated topologies can be seen as generated by infinite disjunctions of sentences in $\mathsf{FO}[\sigma]$ (resp. $\mathsf{EFO}[\sigma]$).

▷ Claim 2.1. The topologies generated by $\tau_{\subseteq_i} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)}$ and $[\![\mathsf{EFO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)}$ are the same, i.e., $\left\langle \tau_{\subseteq_i} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} \right\rangle = \left\langle [\![\mathsf{EFO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} \right\rangle$.

Proof. First of all, any sentence in $\mathsf{EFO}[\sigma]$ defines an upwards-closed set for $\subseteq_i$, and moreover $\mathsf{EFO}[\sigma] \subseteq \mathsf{FO}[\sigma]$, hence $\left\langle [\![\mathsf{EFO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} \right\rangle \subseteq \left\langle \tau_{\subseteq_i} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} \right\rangle$.

For the converse inclusion, it suffices to show that $\mathsf{EFO}[\sigma]$ defines a base of the topology $\langle \tau_{\subseteq_i} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} \rangle$. Consider for this a monotone sentence $\varphi \in \mathsf{FO}[\sigma]$ and a structure A such that $A \models \varphi$. Following the classical proofs (e.g., [6, Theorem 3.2.2] or [24, Corollary 5.4.3]), define $\hat{A}$ as the expansion of A with one additional constant c_a for each $a \in |A|$, interpreted by $c_a^{\hat{A}} \triangleq a$. The *diagram* $\mathrm{Diag}(A)$ of A is the set of all quantifier-free sentences over this extended signature that hold in $\hat{A}$. For a structure $\hat{B} \in \mathrm{Struct}(\sigma \cup \{c_a\}_{a \in A})$, we write B for its reduct in $\mathrm{Struct}(\sigma)$ obtained by removing the constants $\{c_a\}_{a \in A}$.

Let $T \triangleq \mathrm{Diag}(A) \cup \{\neg\varphi\}$, and consider $\hat{B} \in \mathrm{Struct}(\sigma \cup \{c_a\}_{a \in A})$ such that $\hat{B} \models T$. Because $\hat{B} \models \mathrm{Diag}(A)$, by construction $A \subseteq_i B$ (in particular, the sentence $\neg(c_a = c_b)$ belongs to $\mathrm{Diag}(A)$ for all $a \neq b$ in $|A|$), and thus $B \models \varphi$ because φ is monotone, and finally $\hat{B} \models \varphi$ because the constants c_a do not occur in φ. Therefore, $\hat{B} \models \varphi \wedge \neg\varphi$, which is absurd: the theory T is inconsistent, and by the Compactness Theorem for first-order logic, there exists a finite conjunction ψ_0 of sentences in $\mathrm{Diag}(A)$, which is already inconsistent with $\neg\varphi$.

Let ψ_A be the existential closure of the formula obtained by replacing each symbol c_a with a variable x_a in ψ_0; note that ψ_A is an existential sentence. By construction, $A \models \psi_A$, and if $B \models \psi_A$, there exists an interpretation of the constants $\{c_a\}_{a \in A}$ allowing to build an expansion $\hat{B}$ such that $\hat{B} \models \psi_0$. As we just saw that the implication $\psi_0 \implies \varphi$ is valid, $\hat{B} \models \varphi$, and since no constant symbol occurs in φ, $B \models \varphi$.

To conclude, for any open set $U \in \langle \tau_{\subseteq_i} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} \rangle$ and for any $A \in U$, there exists a monotone sentence φ such that $A \in [\![\varphi]\!]_{\mathrm{Struct}(\sigma)}$, and we have proven that there exists $[\![\psi_A]\!]_{\mathrm{Struct}(\sigma)} \in [\![\mathsf{EFO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)}$ such that $A \in [\![\psi_A]\!]_{\mathrm{Struct}(\sigma)} \subseteq [\![\varphi]\!]_{\mathrm{Struct}(\sigma)} \subseteq U$. Therefore, $[\![\mathsf{EFO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)}$ is a base of $\langle \tau_{\subseteq_i} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} \rangle$. ◁

Compactness. The second step relies on the compactness of the sets $[\![\varphi]\!]_{\mathrm{Struct}(\sigma)}$ for monotone sentences φ. Recall that a subset K is *compact* in a topological space τ if, for any *open cover* $(U_i)_{i \in I}$ of K – i.e., a collection of open sets such that $K \subseteq \bigcup_{i \in I} U_i$ – , there exists a finite subset $I_0 \subseteq I$, such that $K \subseteq \bigcup_{i \in I_0} U_i$ (beware that this definition is also called *quasi-compact* in the literature, because we do not require any separation property here). If $\tau = \langle \mathcal{O} \rangle$, by *Alexander's Subbase Lemma*, K is compact if and only if, from every open cover of K using only sets from $\mathcal{O}$, we can extract a finite open cover of K. As open compact sets play a key role in this paper, we introduce here the notation $\mathcal{K}^\circ(X) \triangleq \{U \in \tau \mid U \text{ is compact}\}$. When the topology τ is not clear from the context, we shall write $\mathcal{K}^\circ(X, \tau)$.

▷ Claim 2.2. Every monotone sentence defines a compact open subset in the topology $\langle \tau_{\subseteq_i} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} \rangle$, i.e., $\tau_{\subseteq_i} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} \subseteq \mathcal{K}^\circ\big(\mathrm{Struct}(\sigma), \langle \tau_{\subseteq_i} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} \rangle\big)$.

Proof. Consider a monotone sentence $\varphi \in \mathsf{FO}[\sigma]_{\mathrm{Struct}(\sigma)}$. Let $(U_i)_{i\in I}$ be an open cover of $[\![\varphi]\!]_{\mathrm{Struct}(\sigma)}$. By Alexander's Subbase Lemma, we can assume that for each $i \in I$, $U_i = [\![\varphi_i]\!]_{\mathrm{Struct}(\sigma)}$ for some monotone sentence φ_i. Consider the theory $T \triangleq \{\neg\varphi_i \mid i \in I\} \cup \{\varphi\}$. Because $(U_i)_{i\in I}$ is an open cover, this theory has no models. By the Compactness Theorem for first order logic, there exists a finite set I_0 such that $T_0 \triangleq \{\neg\varphi_i \mid i \in I_0\} \cup \{\varphi\}$ is not satisfiable, proving that $(U_i)_{i\in I_0}$ is an open cover of $[\![\varphi]\!]_{\mathrm{Struct}(\sigma)}$. ◁

▶ Remark 2.3 (Compact sets in $\tau_\leq$). As we will often deal with the Alexandroff topology $\tau_\leq$ of a quasi-order $(X, \leq)$, it is worth noting that $U \in \tau_\leq$ is compact if and only if it is the upward closure $U = \uparrow F$ of some finite subset $F \subseteq_{\mathrm{fin}} X$; this is equivalent to saying that U has finitely many minimal elements up to $\leq$-equivalence [19, Exercise 4.4.22]. Thus Claim 2.2 states that any monotone sentence has finitely many $\subseteq_i$-minimal models in $\mathrm{Struct}(\sigma)$.

Proof of the Łoś-Tarski Theorem. A simple structural induction on the formulæ shows that $[\![\mathsf{EFO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)} \subseteq \tau_{\subseteq_i} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)}$. Regarding the converse inclusion in Equation (3), consider a sentence $\varphi \in \mathsf{FO}[\sigma]$ defining an open set in $\tau_{\subseteq_i}$. By Claim 2.1, there exists a family $(\psi_i)_{i\in I}$ of existential sentences such that $[\![\varphi]\!]_{\mathrm{Struct}(\sigma)} = \bigcup_{i\in I} [\![\psi_i]\!]_{\mathrm{Struct}(\sigma)}$. By Claim 2.2, there is a finite set $I_0 \subseteq_{\mathrm{fin}} I$ for which the equality still holds. Because $\mathsf{EFO}[\sigma]$ is a lattice, this proves the existence of an existential sentence $\psi \triangleq \bigvee_{i\in I_0} \psi_i$ such that $[\![\varphi]\!]_{\mathrm{Struct}(\sigma)} = [\![\psi]\!]_{\mathrm{Struct}(\sigma)}$. ◀

The two properties singled out in claims 2.1 and 2.2 are of different nature. Claim 2.2 really holds for any topology τ and not only for the Alexandroff topology $\tau_{\subseteq_i}$, as opposed to Claim 2.1. Moreover, Claim 2.1 appears to be the most involved one here, but is often easily proven on classes of finite structures.

3 Pre-spectral Spaces and Diagram Bases

Following the two-step decomposition of the proof of the Łoś-Tarski Theorem given in Section 2.2, we define in this section *logically presented pre-spectral spaces* and *diagram bases*, before showing in Theorem 3.4 how they characterise when a preservation theorem holds.

3.1 Pre-spectral Spaces

As a preliminary step toward our main definition, let us first propose a definition of topological spaces (X, τ) where the compact open sets form a *bounded sublattice* of $\wp(X)$ (by which we mean that $\emptyset$ and X belong to the lattice) that generates the topology.

▶ **Definition 3.1** (Pre-spectral space). *A topological space (X, τ) is a* pre-spectral space *whenever $\mathcal{K}^\circ(X)$ is a bounded sublattice of $\wp(X)$ that generates τ, i.e., $\langle \mathcal{K}^\circ(X) \rangle = \tau$.*

The name "pre-spectral" comes from the theory of *spectral* spaces [13], for which the definition is almost identical (see Section 4.2). Pre-spectral spaces will allow us to tap into the rich topological toolset that has been developed for spectral spaces.

Logical presentations. As seen in Claim 2.2, the topology of interest in a preservation theorem is generated by combining a topological space (X, τ) with a bounded sublattice $\mathcal{L}$ of subsets of X, which will be called the *definable* subsets of X. Let us write $\langle X, \tau, \mathcal{L} \rangle$ for the topological space $(X, \langle \tau \cap \mathcal{L} \rangle)$. The following definition is then a direct generalisation of the statement of Claim 2.2.

▶ **Definition 3.2** (Logically presented pre-spectral space)**.** *Let (X, τ) be a topological space and $\mathcal{L}$ be a bounded sublattice of $\wp(X)$. Then $\langle X, \tau, \mathcal{L}\rangle$ is a* logically presented pre-spectral space *(a* lpps*) if its definable open subsets are compact, i.e., if $\tau \cap \mathcal{L} \subseteq \mathcal{K}^\circ(X)$.*

Whenever σ is a finite relational signature, $X \subseteq \mathrm{Struct}(\sigma)$ for a topological space (X, τ) and $\mathcal{L} = [\![\mathsf{FO}[\sigma]]\!]_X$, we denote it by $\langle X, \tau, \mathsf{FO}[\sigma]\rangle$ for simplicity; e.g., $\langle \mathrm{Struct}(\sigma), \tau_{\subseteq_i}, \mathsf{FO}[\sigma]\rangle$ is a lpps by Claim 2.2.

As $\tau \cap \mathcal{L}$ is closed under finite intersection, any open set in $\langle \tau \cap \mathcal{L}\rangle$ is a union of sets from $\tau \cap \mathcal{L}$, thus any compact open set in $\mathcal{K}^\circ(X)$ is a finite union of sets from $\tau \cap \mathcal{L}$. As $\tau \cap \mathcal{L}$ is also closed under finite unions, this shows the inclusion $\mathcal{K}^\circ(X) \subseteq \tau \cap \mathcal{L}$. Thus, in a lpps, $\mathcal{K}^\circ(X) = \tau \cap \mathcal{L}$ is a bounded lattice and any lpps is indeed a pre-spectral space. Conversely, $\langle X, \tau, \mathcal{K}^\circ(X)\rangle$ is well-defined whenever (X, τ) is a pre-spectral space; in this case $\langle X, \tau, \mathcal{K}^\circ(X)\rangle$ is a lpps and it equals (X, τ) (they have the same points and opens).

Beware however that $(X, \langle \tau \cap \mathcal{L}\rangle) = \langle X, \tau, \mathcal{L}\rangle$ being pre-spectral does not entail that it is a lpps; see Remark 3.6 at the end of the section. While pre-spectral spaces capture the topological core behind Claim 2.2 with a simple definition, the logically presented ones are the real objects of interest as far as preservation theorems are concerned, and most of the technical difficulties arising in the remainder of the paper will be concerned with those.

3.2 Diagram Bases

Regarding Claim 2.1, we simply turn the statement of the claim into a definition, which is typically instantiated with $\mathcal{L} = [\![\mathsf{FO}[\sigma]]\!]_X$ and $\mathcal{L}' = [\![\mathsf{F}]\!]_X$ for a fragment F of $\mathsf{FO}[\sigma]$.

▶ **Definition 3.3** (Diagram base)**.** *Let (X, τ) be a topological space, and $\mathcal{L}$ be a bounded sublattice of $\wp(X)$. Then $\mathcal{L}' \subseteq \mathcal{L}$ is a* diagram base *of $\langle X, \tau, \mathcal{L}\rangle$ if $\langle \tau \cap \mathcal{L}\rangle = \langle \mathcal{L}'\rangle$.*

In particular, if $\mathsf{F} \subseteq \mathsf{FO}[\sigma]$ is stable under finite conjunction, this means that any definable open set in X can be written as an infinite disjunction of F-definable sets. Over $\mathrm{Struct}(\sigma)$, this was the "difficult" step in the classical proof of the Łoś-Tarski Theorem. When $X \subseteq \mathrm{Fin}(\sigma)$, this becomes considerably simpler: for every fragment F in Table 1 and any finite structure A, there exists a *diagram* sentence ψ^{F}_A in F such that $A \leq B$ if and only if $B \models \psi^{\mathsf{F}}_A$ for the corresponding quasi-ordering. Therefore, if φ is monotone and $A \in [\![\varphi]\!]_X$, then $A \in [\![\psi^{\mathsf{F}}_A]\!]_X \subseteq [\![\varphi]\!]_X$, showing that $[\![\mathsf{F}]\!]_X$ is a base of $\langle \tau_\leq \cap [\![\mathsf{FO}[\sigma]]\!]_X\rangle$.

3.3 A Generic Preservation Theorem

We have already seen in the proof of the Łoś-Tarski Theorem why logically presented pre-spectral spaces with a diagram base yield preservation. The following theorem also proves the converse direction, under mild hypotheses on $\mathcal{L}'$: $\mathcal{L}'$ must be a lattice and must define compact sets in X for the topology generated by $\mathcal{L}'$. We usually instantiate the theorem with $X \subseteq \mathrm{Struct}(\sigma)$, $\mathcal{L} = [\![\mathsf{FO}[\sigma]]\!]_X$, and $\mathcal{L}' = [\![\mathsf{F}]\!]_X$ where F is a fragment of $\mathsf{FO}[\sigma]$.

▶ **Theorem 3.4** (Generic preservation)**.** *Let τ be a topology on X, $\mathcal{L}$ a bounded sublattice of $\wp(X)$, and $\mathcal{L}'$ a sublattice of $\mathcal{L}$. The following are equivalent:*

1. *X has the $(\tau, \mathcal{L}')$ preservation property and $\mathcal{L}'$ defines only compact sets for the topology $\langle \mathcal{L}'\rangle$.*
2. *$\langle X, \tau, \mathcal{L}\rangle$ is a lpps and $\mathcal{L}'$ defines a diagram base of it.*

Proof. We prove the two implications separately.

1 $\implies$ 2 Assume that X has the $(\tau, \mathcal{L}')$ preservation property. Consider a set $U \in \mathcal{L} \cap \tau$: by the preservation property, $U \in \mathcal{L}'$. This already shows that $\mathcal{L}'$ defines a diagram base of $\langle X, \tau, \mathcal{L}\rangle$. Hence $\langle \mathcal{L}'\rangle = \langle \tau \cap \mathcal{L}\rangle$. Since $U \in \mathcal{L}'$, U is compact in $\langle \mathcal{L}'\rangle$, which means that U is compact in X. Therefore X is a lpps.

2 $\implies$ 1 Assume that $\mathcal{L}'$ defines a diagram base of $\langle X, \tau, \mathcal{L}\rangle$. If $U \in \tau \cap \mathcal{L}$, then it can be written as a possibly infinite union of elements in $\mathcal{L}'$. Also assume that $\langle X, \tau, \mathcal{L}\rangle$ is a lpps: then by compactness, U can be written as a finite union of elements in $\mathcal{L}'$, hence as a single element in $\mathcal{L}'$ since $\mathcal{L}'$ is a lattice. This proves that X has the $(\tau, \mathcal{L}')$ preservation property. Finally, sets in $\mathcal{L}'$ define compact sets in $\langle \mathcal{L}'\rangle$ because it is precisely the topology of $\langle X, \tau, \mathcal{L}\rangle$. ◀

The additional hypotheses on $\mathcal{L}'$ in items 1 and 2 above are somewhat at odds. Asking for $\mathcal{L}'$ to define a diagram base is asking for $\langle \mathcal{L}'\rangle$ to have enough sets, but asking for $\mathcal{L}'$ to only define compact sets is asking for $\langle \mathcal{L}'\rangle$ not to contain too many sets.

▶ Remark 3.5 (Generic monotone preservation). The condition that F must define compact sets in X in Theorem 3.4.1 is actually mild. Consider the preservation results from Table 1 for a fragment F and $\tau = \tau_{\leq}$ the Alexandroff topology of the associated quasi-ordering $\leq$. Assume that X is a $\leq$-downwards-closed subset of $\mathrm{Struct}(\sigma)$ – this is the setting of the known preservation results for classes of finite structures [5, 4, 32, 9, 22].

Observe that, in each case, $[\![\psi]\!]_{\mathrm{Struct}(\sigma)}$ for a sentence $\psi \in \mathsf{F}$ has finitely many $\leq$-minimal models up to $\leq$-equivalence. Because X is downwards-closed, $[\![\psi]\!]_X$ has the same finitely many $\leq$-minimal models. Thus, by Remark 2.3, $[\![\psi]\!]_X$ is compact in $\tau_{\leq}$, and since $[\![\mathsf{F}]\!]_X \subseteq \tau_{\leq}$, it is also compact in the topology generated by $[\![\mathsf{F}]\!]_X$.

In the case of $X = \mathrm{Fin}(\sigma)$, this downward closure condition is fulfilled and F defines a base, thus $(\tau_{\leq}, \mathsf{F})$ preservation holds if and only if $\langle \mathrm{Fin}(\sigma), \tau_{\leq}, \mathsf{FO}[\sigma]\rangle$ is a lpps.

Theorem 3.4 is a generic relationship between pre-spectral spaces and preservation theorems. The downward closure hypothesis in Remark 3.5 is necessary for the equivalence between the preservation property and pre-spectral spaces to hold, as will be shown later in Example 4.2.

▶ Remark 3.6. For each of the fragments F and associated quasi-orderings $\leq$ of Table 1, $\langle \mathrm{Fin}(\sigma), \tau_{\leq}, \mathsf{FO}[\sigma]\rangle = (\mathrm{Fin}(\sigma), \langle \tau_{\leq} \cap [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Fin}(\sigma)}\rangle)$ is a pre-spectral space. Indeed, by Remark 2.3, any compact open K from $\mathcal{K}^{\circ}(\mathrm{Fin}(\sigma))$ is the upward closure $K = \uparrow F$ of a finite set $F \subseteq_{\mathrm{fin}} \mathrm{Fin}(\sigma)$, thus $K = [\![\bigvee_{A \in F} \psi_A^{\mathsf{F}}]\!]_{\mathrm{Fin}(\sigma)}$, which shows that $\mathcal{K}^{\circ}(\mathrm{Fin}(\sigma)) \subseteq [\![\mathsf{F}]\!]_{\mathrm{Fin}(\sigma)}$. As any $\psi \in \mathsf{F}$ has finitely many $\leq$-minimal models in $\mathrm{Fin}(\sigma)$, $\mathcal{K}^{\circ}(\mathrm{Fin}(\sigma)) \supseteq [\![\mathsf{F}]\!]_{\mathrm{Fin}(\sigma)}$, and since F defines a base, $\langle \mathrm{Fin}(\sigma), \tau_{\leq}, \mathsf{FO}[\sigma]\rangle$ is pre-spectral. However, by Remark 3.5 and the non-preservation results of [36, 21, 3, 2, 34], $\langle \mathrm{Fin}(\sigma), \tau_{\subseteq_i}, \mathsf{FO}[\sigma]\rangle$, $\langle \mathrm{Fin}(\sigma), \tau_{\subseteq}, \mathsf{FO}[\sigma]\rangle$, and $\langle \mathrm{Fin}(\sigma), \tau_{\twoheadleftarrow}, \mathsf{FO}[\sigma]\rangle$ are not lpps: the condition $\tau \cap \mathcal{L} \subseteq \mathcal{K}^{\circ}(X)$ is crucial in order to derive preservation results.

Another way of reaching the topological definitions of this section is to consider a folklore result employed in several proofs of preservation theorems over classes of finite structures for fragments F of EFO [32, 4, 5, 7]: if X is downwards-closed for $\leq$, a monotone sentence φ is equivalent to a sentence from F if and only if it has finitely many $\leq$-minimal models in X (up to $\leq$-equivalence). By Remark 2.3, this says that $[\![\varphi]\!]_X$ is compact, while the folklore result itself is essentially using the fact that F defines a base.

4 Related Notions

Pre-spectral spaces generalise two notions arising from order theory, topology, and logics: Noetherian spaces and spectral spaces.

4.1 Well-Quasi-Orderings and Noetherian Spaces

A topological space in which all subsets are compact, or, equivalently, all open subsets are compact, is called *Noetherian* [19, Section 9.7]. A Noetherian space (X, τ) and a bounded sublattice $\mathcal{L}$ of $\wp(X)$ always define a lpps $\langle X, \tau, \mathcal{L} \rangle$. A related notion, considering a quasi-order instead of a topology, leads to the well-known notion of *well-quasi-orders* [26]: a quasi-order is a well-quasi-order if and only if its Alexandroff topology is Noetherian [19, Proposition 9.7.17]. Thus, if $(X, \leq)$ is a well-quasi-order and $\mathcal{L}$ is a bounded sublattice of $\wp(X)$, then $\langle X, \tau_{\leq}, \mathcal{L} \rangle$ is a lpps.

Applications of Noetherian Spaces to Preservation. Let us denote by $\mathcal{G}$ the class of finite simple undirected graphs and by $\sigma_{\mathcal{G}}$ the signature with a single binary edge relation E; then the induced substructure ordering $\subseteq_i$ coincides with the induced subgraph ordering over $\mathcal{G}$.

▶ **Example 4.1** (Finite graphs of bounded tree-depth)**.** Recall that the *tree-depth* $\mathrm{td}(G)$ of a graph G is the minimum height of the comparability graphs F of partial orders such that G is a subgraph of F [30, Chapter 6]. Let $\mathcal{T}_{\leq n}$ be the set of finite graphs of tree-depth at most n ordered by the induced substructure relation $\subseteq_i$. This is a well-quasi-order [14], thus $\langle \mathcal{T}_{\leq n}, \tau_{\subseteq_i}, \mathsf{FO}[\sigma_{\mathcal{G}}] \rangle$ is a lpps, and therefore $\mathcal{T}_{\leq n}$ enjoys the $(\tau_{\subseteq_i}, \mathsf{EFO}[\sigma_{\mathcal{G}}])$-preservation property by Theorem 3.4.

▶ **Example 4.2** (Finite cycles)**.** Consider the class $\mathcal{C} \subseteq \mathcal{G}$ of all finite simple cycles. As is well known, $(\mathcal{C}, \subseteq_i)$ is not a well-quasi-order because any two different cycles are incomparable for the induced substructure ordering [14]. In particular, every singleton is an open set: $(\mathcal{C}, \tau_{\subseteq_i})$ is actually a topological space with the *discrete topology*, and its only compact sets are the finite sets: $\langle \mathcal{C}, \tau_{\subseteq_i}, \mathsf{FO}[\sigma_{\mathcal{G}}] \rangle$ is not a lpps.

By standard locality arguments, for any sentence φ, there exists a finite threshold n_0 on the size of cycles, above which φ is either always true or always false (see the full paper for details). Let τ_n be the topology over $\mathcal{C}$ generated by the definable co-finite sets and the definable sets containing only cycles of size at most n. This is a variation of the *co-finite topology*, and is also Noetherian. Hence, $\langle \mathcal{T}_{\leq n}, \tau_{\subseteq_i}, \mathsf{FO}[\sigma_{\mathcal{G}}] \rangle$ is a lpps, and as $\mathsf{EFO}[\sigma_{\mathcal{G}}]$ defines a diagram base of it, we can apply Theorem 3.4 to deduce preservation. Now, given a monotone sentence φ, either φ has finitely many models or it has co-finitely many. In both cases, this sentence defines an open set in τ_n for some n that is definable in $\mathsf{EFO}[\sigma_{\mathcal{G}}]$. Thus the set of finite cycles has the $(\tau_{\subseteq_i}, \mathsf{EFO}[\sigma_{\mathcal{G}}])$ preservation property.

The previous example shows that the closure condition of Remark 3.5 was necessary, by proving that a space of structures can enjoy a preservation theorem while not defining a lpps.

Relativisation. The following proposition shows that, if we are looking for classes of structures where preservation theorems *always* relativise, then we should endow them with a Noetherian topology.

▶ **Proposition 4.3.** *Let (X, τ) be a pre-spectral space such that for all $Y \subseteq X$, Y with the induced topology is pre-spectral. Then X is Noetherian.*

Proof. Consider any subset Y of X: by assumption, Y is pre-spectral, hence compact in the induced topology, hence compact in (X, τ). ◀

4.2 Spectral Spaces

Spectral spaces are a class of topological spaces appearing naturally in the study of logics and algebra as a generalisation of the Stone Duality theory. Throughout this section we refer to two books and keep the notations consistent with them [19, 13]. A closed subset F of a topological space X is *irreducible* whenever F is non-empty and is not the disjoint union of two non-empty closed sets. The *closure* of a set Y in a space X is the smallest closed set containing Y and is denoted by $\overline{Y}^X$ or $\overline{Y}$ when X is clear from the context. A topological space X is *sober* whenever any irreducible closed subset F is the closure of exactly one point $x \in X$, which translates formally to $\exists x \in X, \overline{\{x\}} = F$ and $\forall y \in X, \overline{\{y\}} = F \Rightarrow y = x$. A *spectral space* is a pre-spectral space that is sober [13, Definition 1.1.5].

When a space (X, τ) is not sober, it is possible to build a *sobrified* version of this space as follows [19, Definition 8.2.17]: $\mathcal{S}(X)$ is the set of irreducible closed sets of X, and the topology is generated by the sets $\Diamond U \triangleq \{F \in \mathcal{S}(X) \mid F \cap U \neq \emptyset\}$ where U is an open set of X. It can be shown that this construction leads to a sober space, is idempotent up to homeomorphism, and constructs the *free* sober space over X [19, Theorem 8.2.44]. This leads to the following correspondence between pre-spectral spaces and spectral spaces.

▶ **Fact 4.4** (Spectral versus pre-spectral). *A space X is pre-spectral if and only if $\mathcal{S}(X)$ is spectral.*

The connection with spectral spaces is of particular interest, because the sobrification functor gives a tool to translate result from the rich theory of spectral spaces to pre-spectral spaces which will be extensively used in Section 5.

5 Basic Closure Properties

To study preservation theorems, we not only want to ensure that the space is pre-spectral, but also to see that the lattice of compact open sets is obtained through a restriction of the logic. Therefore, one of our main concerns with closure properties is to characterise the lattice of compact sets, which must use properties of the definable sets and cannot rely solely on topological constructions.

5.1 Morphisms

Spectral Maps. Let us first introduce the notion of morphism between pre-spectral spaces, inherited from the case of spectral spaces [13, Definition 1.2.2]. A map $f\colon (X, \tau) \to (Y, \theta)$ is a *spectral map* whenever it is continuous and the pre-image of a compact-open set of Y is a compact-open set of X. We will write **PreSpec** for the category of pre-spectral spaces and spectral maps.

▶ **Fact 5.1.** *The image of a pre-spectral space through a spectral map is pre-spectral.*

A crucial role of spectral maps is to guard the definition of *pre-spectral subspaces*, mimicking the one of *spectral subspaces* [13, Section 2.1]. A pre-spectral subspace is not only a subset where the induced topology happens to be pre-spectral, but has the additional property that the *inclusion map* is a spectral map.

Logical Maps. In the case of a lpps, a map $f\colon \langle X, \tau, \mathcal{L}\rangle \to \langle Y, \theta, \mathcal{L}'\rangle$ is a *logical map* whenever it is continuous and the pre-image of a definable open set of Y is a definable open set of X. A map between logically defined pre-spectral spaces is logical if and only if it is spectral, since compact open subsets and definable open subsets coincide in that case. However, the use of logical maps is to prove that some spaces are pre-spectral by transferring logical properties rather than topological ones.

▶ **Fact 5.2.** *The image of a lpps $\langle X, \tau, \mathcal{L}\rangle$ through a logical map is a lpps.*

Of particular interest are the logical maps obtained through syntactic constructions. Let us define an FO-*interpretation* $f \colon X \to Y$ where $X \subseteq \mathrm{Struct}(\sigma_1)$ and $Y \subseteq \mathrm{Struct}(\sigma_2)$ through "relation" formulæ ρ_R for all $R \in \sigma_2$, where ρ_R has as many free variables as the arity of R, and an additional "domain" formula $\delta \in \mathsf{FO}[\sigma_1]$ with one free variable. The image of a σ_1-structure $A \in X$ is the σ_2-structure $f(A)$ with domain $|f(A)| \triangleq \{a \in |A| \mid A \models \delta(a)\}$ and such that $(a_1, \dots, a_n) \in \mathbf{R}^{f(A)}$ if and only if $A \models \rho_R(a_1, \dots, a_n)$. This is a simple model of logical interpretations: many different notions can be found in the literature [8].

An FO-interpretation $f \colon X \to Y$ allows to transfer logical properties from one class of structures to another: if $\varphi \in \mathsf{FO}[\sigma_2]$ is a formula on the structures of Y, then there exists a formula $f^{-1}(\varphi) \in \mathsf{FO}[\sigma_1]$ such that $A \models f^{-1}(\varphi)(\vec{a})$ if and only if $f(A) \models \varphi(f(\vec{a}))$ [24, Section 4.3]; thus, the pre-image of a definable set is definable.

▶ **Fact 5.3.** *An* FO*-interpretation is a logical map if and only if it is continuous.*

This provides us with a proof scheme to show that a space $\langle Y, \tau_2, \mathsf{FO}[\sigma_2]\rangle$ is a lpps: first, build a lpps $\langle X, \tau_1, \mathsf{FO}[\sigma_1]\rangle$, then build a FO-interpetation that is surjective and continuous from X to Y, and conclude that Y is a lpps. This is used for instance by [30, Corollary 10.7] to show that the class of all p *subdivisions* of finite graphs enjoys homomorphism preservation (using a slightly more general notion of FO-interpretations).

5.2 Relativisation

Preservation theorems do not relativise in general, but the stronger notion of being pre-spectral shows that non-trivial sufficient conditions for relativisation exists. However, unlike the theory of spectral spaces, there is not yet a full characterisation of the pre-spectral subsets of a pre-spectral space; see the full paper for a discussion.

▶ **Proposition 5.4** (Sufficient condition for relativisation). *Let $\langle X, \tau, \mathsf{FO}[\sigma]\rangle$ be a lpps, Y be a Boolean combination of compact-open subsets of X, and θ be the topology induced by τ on Y. Then $\langle Y, \theta, \mathsf{FO}[\sigma]\rangle$ is a lpps.*

Proof. It suffices to prove that any definable open set U of Y is the restriction to Y of some definable open set of X. This stronger hypothesis is stable under finite unions and finite intersections, thus we only need to deal with the cases where Y is a definable open of X or the complement of one.

Let us first consider the case where Y is a definable open set of X. Then $U = U \cap Y$ is the restriction to Y of an open definable set of X. Let us next consider the case where Y is a definable closed set of X. Remark that $V \triangleq U \cup (X \setminus Y)$ is an open set of X, and is still definable. Therefore $U = V \cap Y$ with V a definable open set of X. ◀

5.3 Disjoint Unions and Products

Rather than using an already existing pre-spectral space and considering sub-spaces to build new smaller ones, it can be a rather efficient method to combine existing spaces to build bigger spaces. However, to build preservation theorems out of these constructions, it is necessary to represent those them as spaces of structures over some relational signature, which will be the role of definitions 5.5 and 5.7.

▶ **Definition 5.5** (Logical sum). *Let $(\langle X_i, \tau_i, \mathsf{FO}[\sigma_i]\rangle)_{i\in I}$ be a family of spaces. The* logical sum $\langle X, \tau, \mathsf{FO}[\sigma]\rangle$ *is defined as follows:*

1. *The signature σ is the disjoint union of the signatures $(\sigma_i)_{i\in I}$.*
2. *The set X is the union (disjoint by construction) $\bigcup_{i\in I} f_i(X_i)$ where, for all $i \in I$, $f_i : X_i \to \mathrm{Struct}(\sigma)$ is defined by $|f_i(A)| \triangleq |A|$ and $(a_1, \ldots, a_n) \in \mathbf{R}^{f_i(A)}$ if and only if $R \in \sigma_i$ and $(a_1, \ldots, a_n) \in \mathbf{R}^A$.*
3. *The topology τ is generated by the sets $f_i(U)$ where $U \in \tau_i$ and $i \in I$.*

The logical sum space is a simple translation of the topological sum space, which leads to the following result (see the full paper for a proof).

▶ **Proposition 5.6** (Stability under finite logical sum). *Let $(\langle X_i, \tau_i, \mathsf{FO}[\sigma_i]\rangle)_{i\in I}$ be a finite family of lpps. The logical sum of those spaces is a lpps homeomorphic to the sum of those spaces in* **PreSpec**.

In the case of products, a sentence over a product is not simply obtained by projecting on each component. This is handled in our proof of Proposition 5.8 in the full paper by reducing the first-order theory of the product to the first-order theories of its components thanks to Feferman-Vaught decompositions [16, 29].

▶ **Definition 5.7** (Logical product). *Let $(\langle X_i, \tau_i, \mathsf{FO}[\sigma_i]\rangle)_{i\in I}$ be a family of spaces. The* logical product $\langle X, \tau, \mathsf{FO}[\sigma]\rangle$ *is defined as follows:*

1. *The signature σ is the disjoint union of the signatures $(\sigma_i)_{i\in I}$ with additional unary predicates ε_i for each $i \in I$.*
2. *The set X is the image of $\prod_{i\in I} X_i$ through the map $f\colon \prod_{i\in I} X_i \to \mathrm{Struct}(\sigma)$ that associates to each $(A_i)_{i\in I}$ the disjoint union of the structures A_i with ε_i true on the structure A_i for $i \in I$.*
3. *The topology τ generated by the sets U such that $f^{-1}(U)$ is an open set of $\prod_{i\in I}(X_i, \tau_i)$.*

▶ **Proposition 5.8** (Stability under finite logical product). *Let $(\langle X_i, \tau_i, \mathsf{FO}[\sigma_i]\rangle)_{i\in I}$ be a finite family of lpps. The logical product of those spaces is a lpps homeomorphic to the product of the spaces X_i in* **PreSpec**.

6 Logical Closure

Consider a set Z equipped with a bounded sublattice $\mathcal{L}$ of $\wp(Z)$. In this section, we provide a way to consider the closure of a space $X \subseteq Z$ in a suitable topology so that if X is a lpps, then its closure also is. Let us write $\tau_{\mathcal{L}} \triangleq \langle \mathcal{L} \cup \{U^c \mid U \in \mathcal{L}\}\rangle$ for the topology generated by the sets of $\mathcal{L}$ and their complements. We call the closure $\overline{X}$ of X in $(Z, \tau_{\mathcal{L}})$ its *logical closure*.

We show in the full paper that lpps are stable under logical closures. For $X \subseteq Z$ and a sublattice $\mathcal{L}$ of $\wp(Z)$, we write $\mathcal{L}_X \triangleq \{U \cap X \mid U \in \mathcal{L}\}$ for the lattice induced by X.

▶ **Proposition 6.1** (Stability under logical closure). *Let $X \subseteq Y \subseteq \overline{X}$ and τ be a topology on Y. If $\langle X, \tau_X, \mathcal{L}_X\rangle$ is a lpps for the topology τ_X induced by τ on X, then so is $\langle Y, \tau, \mathcal{L}_Y\rangle$. If $\mathcal{L}'$ is a sublattice of $\mathcal{L}$ and $\mathcal{L}'_X$ is a diagram base of X then $\mathcal{L}'_Y$ is a diagram base of Y.*

Applications of logical closures. We now show that Proposition 6.1 allows to restate known preservation theorems and derive new ones. We consider the case where $Z = \mathrm{Struct}(\sigma)$ and $\mathcal{L} = [\![\mathsf{FO}[\sigma]]\!]_{\mathrm{Struct}(\sigma)}$, and we write τ_{FO} for the topology $\tau_{\mathcal{L}}$.

$$\begin{tikzcd}
X_i \arrow[dd, "f_{i,j}"'] & & \\
& X \arrow[ul, "f_i"'] \arrow[dl, "f_j"] & Y \arrow[l, dashed, "g"] \arrow[ull, bend right, "g_i"'] \arrow[dll, bend left, "g_j"] \\
X_j & &
\end{tikzcd}$$

Figure 1 The commutative diagram of a projective system.

Let us define $\mathrm{FMP}(\sigma) \subseteq \mathrm{Struct}(\sigma)$ as the set of structures whose first-order theory satisfies the *finite model property*: any definable subset of $\mathrm{FMP}(\sigma)$ has a finite model. We prove that homomorphism preservation can be lifted from $\mathrm{Fin}(\sigma)$ (where it holds by Rossman's Theorem) to $\mathrm{FMP}(\sigma)$ in Corollary 6.2. To our knowledge this is a new result. This follows from Proposition 6.1 and the fact that $\mathrm{FMP}(\sigma)$ is the closure of $\mathrm{Fin}(\sigma)$ in the topology τ_{FO} (see the full paper for the proof).

▶ **Corollary 6.2** (Homomorphism preservation for structures with the finite model property)**.** $\mathrm{FMP}(\sigma)$ *has the* $(\tau_{\to}, \mathsf{EPFO}[\sigma])$ *preservation property.*

Let $\mathrm{Fin}^{\uplus}(\sigma)$ be the set of countable disjoint unions of finite structures over a finite relational signature σ. We state in Corollary 6.3 another consequence of Rossman's Theorem and Proposition 6.1, using the fact $\mathrm{Fin}(\sigma) \subsetneq \mathrm{Fin}^{\uplus}(\sigma) \subsetneq \mathrm{FMP}(\sigma) = \overline{\mathrm{Fin}(\sigma)}$; the same result was first shown by Nešetřil and Ossona de Mendez in [30, Theorem 10.6].

▶ **Corollary 6.3** (Homomorphism preservation for countable unions of finite structures)**.** $\mathrm{Fin}^{\uplus}(\sigma)$ *has the* $(\tau_{\to}, \mathsf{EPFO}[\sigma])$ *preservation property.*

7 Limits of Projective Systems

A natural construction in the category of topological spaces is the *projective limit*, and the category **Spec** of spectral spaces and spectral maps is closed under this construction [13, Corollary 2.3.8]. As an illustration, we show in Section 7.2 that $\langle \mathrm{Fin}(\sigma), \tau_{\to}, \mathsf{FO}[\sigma]\rangle$ is the projective limit of a system of Noetherian spaces, which provides an alternative understanding of Rossman's Theorem [32]. In fact, as we show in Section 7.3, any pre-spectral space is the limit of a projective system of Noetherian spaces.

7.1 Projective Systems

A *projective system* $\mathcal{F}$ in a category $\mathbf{C}$ assigns to each element i of a directed partially ordered set I an object X_i and to each ordered pair $i \leq j$ a so-called *bonding map* $f_{i,j} \colon X_i \to X_j$ so that, for all $i, j, k \in I$ with $k \leq j \leq i$, we have $f_{i,i} = \mathrm{id}_{X_i}$ and $f_{j,k} \circ f_{i,j} = f_{i,k}$. The *projective limit* of a projective system $\mathcal{F}$ is an object X with maps $f_i \colon X \to X_i$ *compatible* with the system $\mathcal{F}$, which means that, for all $i \geq j$, $f_{i,j} \circ f_i = f_j$. Moreover, X satisfies a universal property: whenever $\{g_i \colon Y \to X_i\}_{i \in I}$ is a family of maps compatible with $\mathcal{F}$, there exists a unique map $g \colon Y \to X$ such that $g_i = f_i \circ g$ for all $i \in I$.

Unfortunately, there exists projective systems in **PreSpec** that do not have limits, as can be witnessed by a slight adaptation of [35, Example 3]. Let us introduce here the category of topological spaces and continuous maps, denoted by **Top**. A projective system in **PreSpec** is a projective system of topological spaces in **Top**. A projective system in **PreSpec** always has a limit when considered as a projective system in **Top**; we give a sufficient condition for this space to be the limit in **PreSpec** (see the full paper for the proof).

▶ **Lemma 7.1** (Transfer of projective limits). *Let $\mathcal{F}$ be a projective system of pre-spectral spaces in* **PreSpec***. If $\{f_i \colon X \to X_i\}_{i \in I}$ is the limit of $\mathcal{F}$ in* **Top** *where the maps $f_i \colon X \to X_i$ are spectral, then it is the limit of $\mathcal{F}$ in* **PreSpec***. Moreover, $\mathcal{K}^\circ(X) = \bigcup_{i \in I} \{f_i^{-1}(V) \mid V \in \mathcal{K}^\circ(X_i)\}$.*

7.2 Application to the Homomorphism Preservation Theorem

Throughout this section, we fix a finite relational signature σ and a downwards-closed subset X of $\mathrm{Fin}(\sigma)$ for the homomorphism ordering $\to$, i.e., X is co-homomorphism closed. We will see how Rossman's Theorem can be explained as the existence of a projective limit.

n-Homomorphisms. Let us define the *tree-depth* $\mathrm{td}(A)$ of a finite structure A as the tree-depth $\mathrm{td}(\mathcal{G}(A))$ of its associated *Gaifman graph* $\mathcal{G}(A)$ [27, Definition 4.1]. Following the idea of the original proof in [32, Section 3.2], we are going to use quasi-orders that are *coarser* than the homomorphism quasi-order, and refine those progressively. For every $n \in \mathbb{N}$, we define $A \to_n B$ if for every structure C of tree-depth at most n, $C \to A$ implies $C \to B$. Note that on finite structures, $A \to B$ if and only if $A \to_{\mathrm{td}(A)} B$. Then the intersection of all the $\to_n$ relations is $\to$. Let us consider the corresponding Alexandroff topologies: $X \triangleq \langle X, \tau_{\to}, \mathsf{FO}[\sigma] \rangle$ and for $n \in \mathbb{N}$, let $X_n \triangleq \langle X, \tau_{\to_n}, \mathsf{FO}[\sigma] \rangle$.

Rossman's Lemma. In his paper [32], Rossman provides a function $\rho \colon \mathbb{N} \to \mathbb{N}$ and relates indistinguishability in the fragment $\mathsf{FO}_n[\sigma]$ of first-order logic with quantifier rank at most n to $\rho(n)$-homomorphism equivalence [32, Corollary 5.14]. We state this result in a self-contained manner below (see also [30, Theorem 10.5]).

▶ **Lemma 7.2** (Rossman's Lemma [32]). *There exists $\rho \colon \mathbb{N} \to \mathbb{N}$ such that, for all $n \in \mathbb{N}$, if $\varphi \in \mathsf{FO}_n[\sigma]$ is closed under homomorphisms, then it is closed under $\rho(n)$-homomorphisms.*

Rossman's Lemma is the combinatorial heart of Rossman's Theorem, so the developments in this section are only meant to show how the pre-spectral framework can capture his arguments translating the technical statement from Lemma 7.2 into a proof of homomorphism preservation in the finite.

Projective System. We are now ready obtain X as a limit of a projective system in **PreSpec**. We are going to exploit Lemma 7.2 through the definition of the topological spaces $Y_n \triangleq \langle X, \tau_{\to}, \mathsf{FO}_n[\sigma] \rangle$ for all n. We will use the following consequence of Rossman's Lemma (see the full paper for a proof).

▷ Claim 7.3. $\forall n \geq 1, \mathcal{K}^\circ(Y_n) \subseteq \mathcal{K}^\circ\big(X_{\rho(n)}\big) \subseteq \mathcal{K}^\circ(X)$.

The following theorem was famously first shown by Rossman in [32, Corollary 7.1]. A more recent proof in [33] uses lower bounds from circuit complexity. Similar results were shown in [30, Section 10.7] when assuming essentially the same statement as Lemma 7.2; in fact, carefully unwrapping the hypotheses of the *topological preservation theorem* of [30, Theorem 10.3] leads to the very definition of a projective system.

▶ **Theorem 7.4.** *Let σ be a finite relational signature and X be a non-empty downwards-closed subset of* $\mathrm{Fin}(\sigma)$ *for $\to$. Then X has the $(\tau_{\to}, \mathsf{EPFO}[\sigma])$ preservation property.*

Proof. Consider the projective system $\mathcal{F} \triangleq \{\mathrm{id}_{i,j} \colon Y_i \to Y_j\}_{i \leq j \in I}$ indexed by $I \triangleq \mathbb{N} \setminus \{0\}$. Each space Y_i is Noetherian for all $i \in I$ because $\mathsf{FO}_i[\sigma]$ contains finitely many non-equivalent sentences, hence Y_i contains finitely many open sets. Hence $\mathcal{K}^\circ(Y_i) = \tau_\to \cap [\![\mathsf{FO}_i[\sigma]]\!]_X$. Also, the maps $\mathrm{id}_{i,j}$ are spectral and $\mathcal{F}$ is a projective system in **PreSpec**. Claim 7.3 shows that the identity map $\mathrm{id}_i : X \to Y_i$ is a spectral map for all $i \in I$.

Assume that $\{g_i \colon Z \to Y_i\}_{i \in I}$ is a collection of morphisms in **Top** such that $\forall i \geq j \in I, g_j = \mathrm{id}_{i,j} \circ g_i$. Since $\mathrm{id}_{i,j}$ is the identity map, all the maps $(g_i)_{i \in I}$ are equal. In particular, one can build $g \colon Z \to X$ defined by any one of them. Let us show that g is a continuous map. If U is a definable open set of X, then U is a definable open set in Y_n for some n, hence $g^{-1}(U) = g_n^{-1}(U)$ is open. Since X has a base of definable open sets, this proves that g is continuous.

Assume that g' is an other continuous map making the diagram commute. As I is non empty, consider some $i \in I$, we have $g_i = \mathrm{id}_i \circ g = \mathrm{id}_i \circ g'$. Since f_i is the identity map we conclude $g = g'$.

We have shown that X is the limit of $\mathcal{F}$ in **Top**. Since the maps $\mathrm{id}_i \colon X \to X_i$ are spectral, Lemma 7.1 shows that X is a pre-spectral space such that $\mathcal{K}^\circ(X) = \bigcup_{i \in I} \mathcal{K}^\circ(Y_i) = \tau_\to \cap [\![\mathsf{FO}[\sigma]]\!]_X$. In particular, X is a lpps. As X is downwards-closed, by Remark 3.5 it has the $(\tau_\to, \mathsf{EPFO}[\sigma])$-preservation property. ◀

7.3 Completeness

We are now going to prove that any pre-spectral space can be obtained as a solution to a projective system of pre-spectral spaces, showing that the proof method of the previous sub-section is in some sense complete. In fact, this system is going to contain only Noetherian spaces (see the full paper). It is analoguous to the fact that any spectral space is a projective limit of finite T_0 spaces [23, Proposition 10].

▶ **Proposition 7.5** (Pre-spectral spaces are limits of Noetherian spaces). *Let (X, τ) be a pre-spectral space, there exists a projective system of Noetherian spaces in* **PreSpec** *such that X is the limit of this projective system.*

8 Concluding Remarks

In this paper, we have introduced a general framework for preservation results, mixing topological and model-theoretic notions. The key notion here is the one of *logically presented pre-spectral spaces*, which requires the (topological) compactness of the definable sets of interest. This definition captures simultaneously the classical proofs of preservation theorems over the class of all structures (we detailed the case of the Łoś-Tarski Theorem in Section 2.2) and all the known preservation results over classes of finite structures in the literature (see Remark 3.5). Our approach is comparable to the one adopted in the *topological preservation theorem* of [30, Theorem 10.3], in that we employ topological concepts to present a generic preservation theorem; however we believe our formulation to be considerably simpler and more flexible.

We have developed a mathematical toolbox for working with logically presented pre-spectral spaces, allowing to build new spaces from known ones. Besides relatively mundane stability properties under suitable notions of morphisms, subspaces, finite sums, and finite products – which still required quite some care in order to account for first-order definability – , we have shown that more exotic constructions through topological closures or projective limits of topological spaces could also be employed. Those last two constructions give an alternative viewpoint on Rossman's proof of homomorphism preservation over the class of finite structures (Theorem 7.4), and a new homomorphism preservation result over the class of structures with the finite model property (Corollary 6.2).

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 Miklós Ajtai and Yuri Gurevich. Monotone versus positive. *Journal of the ACM*, 34(4):1004–1015, 1987. doi:10.1145/31846.31852.

3 Miklós Ajtai and Yuri Gurevich. Datalog vs first-order logic. *Journal of Computer and System Sciences*, 49(3):562–588, 1994. doi:10.1016/S0022-0000(05)80071-6.

4 Albert Atserias, Anuj Dawar, and Martin Grohe. Preservation under extensions on well-behaved finite structures. *SIAM Journal on Computing*, 38(4):1364–1381, 2008. doi:10.1137/060658709.

5 Albert Atserias, Anuj Dawar, and Phokion G. Kolaitis. On preservation under homomorphisms and unions of conjunctive queries. *Journal of the ACM*, 53(2):208–237, 2006. doi:10.1145/1131342.1131344.

6 Chen Chung Chang and H. Jerome Keisler. *Model Theory*, volume 73 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1990.

7 Yijia Chen and Jörg Flum. Forbidden induced subgraphs and the Łoś-Tarski Theorem. Preprint, 2020. URL: https://arxiv.org/abs/2008.00420.

8 Bruno Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75, 1994. doi:10.1016/0304-3975(94)90268-2.

9 Anuj Dawar. Homomorphism preservation on quasi-wide classes. *Journal of Computer and System Sciences*, 76(5):324–332, 2010. doi:10.1016/j.jcss.2009.10.005.

10 Anuj Dawar and Stephan Kreutzer. On Datalog vs. LFP. In *Proceedings of ICALP'08*, volume 5126 of *Lecture Notes in Computer Science*, pages 160–171, 2008. doi:10.1007/978-3-540-70583-3_14.

11 Anuj Dawar and Abhisekh Sankaran. Extension preservation in the finite and prefix classes of first order logic. Preprint, 2020. URL: https://arxiv.org/abs/2007.05459.

12 Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In *Proceedings of PODS'08*, pages 149–158, 2008. doi:10.1145/1376916.1376938.

13 Max Dickmann, Niels Schwartz, and Marcus Tressl. *Spectral Spaces*, volume 35 of *New Mathematical Monographs*. Cambridge University Press, 2019.

14 Guoli Ding. Subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 16:489–502, 1992. doi:10.1002/jgt.3190160509.

15 Tomás Feder and Moshe Y. Vardi. Homomorphism closed vs. existential positive. In *Proceedings of LICS'03*, pages 311–320, 2003. doi:10.1109/LICS.2003.1210071.

16 Solomon Feferman and Robert Vaught. The first order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47(1):57–103, 1959. doi:10.4064/fm-47-1-57-103.

17 Diego Figueira and Leonid Libkin. Pattern logics and auxiliary relations. In *Proceedings of CSL-LICS'14*, pages 40:1–40:10, 2014. doi:10.1145/2603088.2603136.

18 Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. Naïve evaluation of queries over incomplete databases. *ACM Transactions on Database Systems*, 39(4):1–42, 2014. doi:10.1145/2691190.2691194.

19 Jean Goubault-Larrecq. *Non-Hausdorff Topology and Domain Theory*, volume 22 of *New Mathematical Monographs*. Cambridge University Press, 2013.

20 Martin Grohe. Existential least fixed-point logic and its relatives. *Journal of Logic and Computation*, 7(2):205–228, 1997. doi:10.1093/logcom/7.2.205.

21 Yuri Gurevich. Toward logic tailored for computational complexity. In *Computation and Proof Theory, Proceedings of LC'84*, volume 1104 of *Lecture Notes in Mathematics*, pages 175–216. Springer, 1984. doi:10.1007/BFb0099486.

22 Frederik Harwath, Lucas Heimberg, and Nicole Schweikardt. Preservation and decomposition theorems for bounded degree structures. In *Proceedings of CSL-LICS'14*, pages 49:1–49:10, 2014. doi:10.1145/2603088.2603130.

23 Melvin Hochster. Prime ideal structure in commutative rings. *Transactions of the American Mathematical Society*, 142:43–60, 1969. doi:10.1090/S0002-9947-1969-0251026-X.

24 Wilfrid Hodges. *A shorter model theory*. Cambridge University Press, 1997.

25 Phokion G. Kolaitis. Reflections on finite model theory. In *Proceedings of LICS'07*, pages 257–269, 2007. doi:10.1109/LICS.2007.39.

26 Joseph B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *Journal of Combinatorial Theory, Series A*, 13(3):297–305, 1972. doi:10.1016/0097-3165(72)90063-5.

27 Leonid Libkin. *Elements of finite model theory*. Springer, 2012.

28 Jerzy Łoś. On the extending of models (I). *Fundamenta Mathematicae*, 42(1):38–54, 1955. doi:10.4064/fm-42-1-38-54.

29 Johann A. Makowsky. Algorithmic uses of the Feferman–Vaught Theorem. *Annals of Pure and Applied Logic*, 126(1–3):159–213, 2004. doi:10.1016/j.apal.2003.11.002.

30 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Springer, 2012.

31 Eric Rosen. Some aspects of model theory and finite structures. *Bulletin of Symbolic Logic*, 8(3):380–403, 2002. doi:10.2178/bsl/1182353894.

32 Benjamin Rossman. Homomorphism preservation theorems. *Journal of the ACM*, 55(3):15:1–15:53, 2008. doi:10.1145/1379759.1379763.

33 Benjamin Rossman. An improved homomorphism preservation theorem from lower bounds in circuit complexity. *ACM SIGLOG News*, 3(4):33–46, 2016. doi:10.1145/3026744.3026746.

34 Alexei P. Stolboushkin. Finitely monotone properties. In *Proceedings of LICS'95*, pages 324–330, 1995. doi:10.1109/LICS.1995.523267.

35 Arthur H. Stone. Inverse limits of compact spaces. *General Topology and its Applications*, 10(2):203–211, 1979. doi:10.1016/0016-660X(79)90008-4.

36 William W. Tait. A counterexample to a conjecture of Scott and Suppes. *Journal of Symbolic Logic*, 24(1):15–16, 1959. doi:10.2307/2964569.

37 Alfred Tarski. Contributions to the theory of models. I. *Indagationes Mathematicae (Proceedings)*, 57:572–581, 1954. doi:10.1016/S1385-7258(54)50074-0.

38 Balder ten Cate and Phokion G. Kolaitis. Structural characterizations of schema-mapping languages. In *Proceedings of ICDT'09*, pages 63–72, 2009. doi:10.1145/1514894.1514903.

Choiceless Computation and Symmetry: Limitations of Definability

Benedikt Pago
Mathematical Foundations of Computer Science, RWTH Aachen University, Germany
pago@logic.rwth-aachen.de

Abstract

The search for a logic capturing PTIME is a long standing open problem in finite model theory. One of the most promising candidate logics for this is *Choiceless Polynomial Time* with counting (CPT). Abstractly speaking, CPT is an isomorphism-invariant computation model working with hereditarily finite sets as data structures.

While it is easy to check that the evaluation of CPT-sentences is possible in polynomial time, the converse has been open for more than 20 years: Can every PTIME-decidable property of finite structures be expressed in CPT?

We attempt to make progress towards a negative answer and show that Choiceless Polynomial Time cannot compute a *preorder* with colour classes of *logarithmic size* in every hypercube. The reason is that such preorders have super-polynomially many automorphic images, which makes it impossible for CPT to define them.

While the computation of such a preorder is not a decision problem that would immediately separate P and CPT, it is significant for the following reason: The so-called Cai-Fürer-Immerman (CFI) problem is one of the standard "benchmarks" for logics and maybe best known for separating fixed-point logic with counting (FPC) from P. Hence, it is natural to consider this also a potential candidate for the separation of CPT and P. The strongest known positive result in this regard says that CPT is able to solve CFI if a preorder with logarithmically sized colour classes is present in the input structure.

Our result implies that this approach cannot be generalised to unordered inputs. In other words, CFI on unordered hypercubes is a PTIME-problem which provably cannot be tackled with the state-of-the-art choiceless algorithmic techniques.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Mathematics of computing → Permutations and combinations

Keywords and phrases finite model theory, descriptive complexity, choiceless computation, symmetries of combinatorial objects

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.33

Acknowledgements I would like to thank my advisor Erich Grädel for helpful comments and discussions.

1 Introduction

One of the big open questions in descriptive complexity theory is whether there exists a logic capturing PTIME (see [4], [10], [11], [13]). Towards an answer to this question, several logics of increasing expressive power within PTIME have been devised, the best-studied of which is probably FPC, *fixed-point logic with counting* (see [5] for a survey). However, FPC only corresponds to a strict subset of PTIME because it cannot express the so-called *CFI query*, a version of the graph isomorphism problem on certain graphs constructed by Cai, Fürer and Immerman in 1992 [3]. This problem is in P and has turned out to be extremely valuable as a benchmark for PTIME-logics as well as for certain classes of graph isomorphism algorithms.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 33; pp. 33:1–33:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The most important candidate logics for capturing PTIME, which have not yet fallen prey to the CFI problem, are *Rank logic* [6] and *Choiceless Polynomial Time* (CPT). CPT was introduced in 1999 by Blass, Gurevich and Shelah [2] as a machine model that comes as close to Turing machines as possible, while enforcing *isomorphism-invariance* of the computations – this property is precisely the main difference between logics and classical Turing machines. Since its original invention, various different formalisations of CPT have emerged but the underlying principle is always the same: *Symmetric* computation on *polynomially-sized hereditarily finite sets* as data structures.

Not many lower bound results for Choiceless Polynomial Time are known so far, and of course, no *decision problem* in P has been shown to be undefinable in CPT. However, what has been achieved is a non-definability statement for a *functional problem*: Rossman showed that CPT cannot define the dual of any given finite vector space [15]. Our contribution is a result of a similar kind, but stronger in a sense: We show non-definability not only for a concrete functionally determined object, but for all objects satisfying a certain set of properties. Concretely, no CPT program can define a hereditarily finite set representing a preorder with colour classes of logarithmic size in every hypercube. This can be seen – potentially – as a first step towards a non-definability result for a decision problem: the already mentioned CFI query; this would separate CPT from PTIME. Let us explain what undefinable preorders have to do with the CFI problem (see Section 3 for details).

Recall that a preorder in a structure can be seen as a linear order on a collection of *colour classes*, which form a partition of the universe: These colour classes are subsets of the structure whose elements are pairwise indistinguishable. The smaller the colour classes are, the "finer" is the preorder, and the more closely it resembles a linear order. By the famous Immerman-Vardi Theorem ([12], [17]), fixed-point logic, and therefore also CPT, captures PTIME on linearly ordered structures. Therefore, intuitively speaking, hard problems like CFI should become easier to handle if CPT is able to define a sufficiently fine preorder, or even a linear order, on the input structure. Indeed, Pakusa, Schalthöfer and Selman showed that CPT can define the CFI query if a preorder with colour classes of logarithmic size is available [14]. This is the strongest known positive result concerning the solvability of CFI in CPT.

Our contribution implies that this result cannot be generalised to the CFI problem on unordered input structures: Instances of CFI can be obtained by applying the Cai-Fürer-Immerman construction to any family of connected graphs, in particular also to hypercubes. Since CPT cannot define a sufficiently fine preorder in all hypercubes, and the CFI construction preserves the hypercube-structure, the algorithmic technique from [14] which heavily relies on such preorders cannot be applied to all unordered CFI structures.

Therefore, if CFI on unordered structures is solvable in CPT, entirely new choiceless algorithmic techniques are needed to show this. Otherwise, if CFI is indeed a separating problem for CPT and P, one possible approach to prove this would be to identify further hereditarily finite sets over hypercubes which are not CPT-definable.

Technically, what we show in this paper is a statement concerning the orbit size of certain hereditarily finite objects over hypercubes: For every $n \in \mathbb{N}$, fix a h.f. object representing a preorder in the n-dimensional hypercube. If the colour classes of each preorder are of logarithmic size w.r.t. the hypercube, then the orbit size (w.r.t. the hypercube-automorphisms) of these h.f. objects grows super-polynomially in 2^n, which is the size of the n-dimensional hypercube.

Since CPT is a logic and therefore isomorphism-invariant, it has to define any object together with its entire orbit – if the size of the orbit is not polynomially bounded, then this is not possible in Choiceless Polynomial Time. In fact, we can interpret this non-definability result as an inherent weakness of choiceless polynomial time computation in general: It holds for any isomorphism-invariant polynomial time (or even polynomial space) computation model on hereditarily finite sets. Hence, should it be the case that CPT fails to capture PTIME because of a super-polynomial orbit argument like this one, we could conclude that the quest for a PTIME-logic should continue with other data structures than hereditarily finite sets.

Finally, we remark that the main combinatorial tool we use in our proof – so-called *supporting partitions* – is taken from [1], where Anderson and Dawar show a correspondence between FPC and Symmetric Circuits. There, it is used for the calculation of orbit sizes of circuit gates. The fact that this tool also helps to understand the symmetries of hereditarily finite objects over hypercubes demonstrates its versatility and usefulness for the study of symmetric objects in general.

2 Choiceless computation and the undefinability of preorders

In this paper, we will not give a definition of CPT, but only state its properties that our lower bound depends on. Thereby, our result also holds for a much broader class of choiceless computation models that includes CPT.
For details on CPT, we refer to the literature: A concise survey on the subject can be found in [8]. It should be noted that there are multiple different ways to formalise CPT: The original definition was via abstract state machines [2], but there are also more “logic-like” presentations such as Polynomial Interpretation Logic (see [9], [16]) and BGS-logic [15]. The latter is essentially a fixed-point logic that allows for the isomorphism-invariant creation and manipulation of *hereditarily finite sets* over the input structure. In fact, it has been shown in [7] that any CPT-program (the words “program” and “sentence” are often used interchangeably in the context of CPT) is equivalent to a sentence in FPC (fixed-point logic with counting) evaluated in the input structure enriched with all the necessary hereditarily finite sets. Therefore, let us make this notion precise.

Hereditarily finite sets and choiceless computation

Let A be a nonempty set. The set of *hereditarily finite objects* over A, $\mathrm{HF}(A)$, is defined as $\bigcup_{i \in \mathbb{N}} \mathrm{HF}_i(A)$, where $\mathrm{HF}_0(A) := A \cup \{\emptyset\}, \mathrm{HF}_{i+1}(A) := \mathrm{HF}_i(A) \cup 2^{\mathrm{HF}_i(A)}$. The size of an h.f. set $x \in \mathrm{HF}(a)$ is measured in terms of its *transitive closure* $\mathrm{tc}(x)$: The set $\mathrm{tc}(x)$ is the least transitive set such that $x \in \mathrm{tc}(x)$. Transitivity means that for every $a \in \mathrm{tc}(x)$, $a \subseteq \mathrm{tc}(x)$.
If the atom set A is the universe of a structure $\mathfrak{A}$, then the action of $\mathrm{Aut}(\mathfrak{A}) \subseteq \mathrm{Sym}(A)$, the automorphism group of $\mathfrak{A}$, extends naturally to $\mathrm{HF}(A)$: For $x \in \mathrm{HF}(A)$, $\pi \in \mathrm{Aut}(\mathfrak{A})$, x^π is obtained from x by replacing each occurrence of an atom a in x with $\pi(a)$.
The *orbit* (w.r.t. the action of $\mathrm{Aut}(\mathfrak{A})$) of an object $x \in \mathrm{HF}(A)$ is the set of all its automorphic images, i.e. $\{x^\pi \mid \pi \in \mathrm{Aut}(\mathfrak{A})\}$. The *stabiliser* $\mathrm{Stab}(x)$ of x is the subgroup $\{\pi \in \mathrm{Aut}(\mathfrak{A}) \mid x^\pi = x\}$.

▶ **Definition 1.** *Let $\mathfrak{A}$ be a finite relational structure with universe A, and $p : \mathbb{N} \longrightarrow \mathbb{N}$ a polynomial. We say that a h.f. object $x \in HF(A)$ is*

- symmetric *(w.r.t. $\mathfrak{A}$) if x is stabilised by all automorphisms of $\mathfrak{A}$, i.e. $Stab(x) = Aut(\mathfrak{A})$;*
- p-bounded *if $|tc(x)| \leq p(|A|)$.*

Every CPT-program comes with an explicit polynomial bound p that limits both the length of its runs as well as the size of the h.f. sets that it may use in the computation. Further, due to its nature as a logic, all operations of CPT are symmetry-invariant. This is already everything that our lower bound depends on. The following abstract view on the execution of CPT-programs is true regardless of the concrete presentation of CPT, and this level of abstraction is sufficient for the purposes of this paper:

Let Π be a CPT-program with bound p, and $\mathfrak{A}$ be a structure of matching signature. Then the run of Π on $\mathfrak{A}$ is a sequence of h.f. sets $x_1, x_2, ... \in \mathrm{HF}(A)$, each of which is symmetric and p-bounded w.r.t. $\mathfrak{A}$.

Consequently, no CPT-program – and generally, no computation model operating on symmetric p-bounded h.f. sets – can compute a h.f. set x with super-polynomial orbit size because the corresponding stage of the run must contain x along with its entire orbit in order to fulfil the symmetry-condition. Now, we are almost ready to state our general lower bound theorem, which applies to CPT as a special case by the facts just mentioned.

Preorders and colour classes

A *preorder* $\prec$ on a set A induces a partition of A into *colour classes* $C_1, ..., C_m$. A colour class is a set of $\prec$-incomparable elements, and $\prec$ induces a linear order on the colour classes. The canonical representation of such a preorder as a h.f. set is $\{C_1, \{C_2, \{C_3, \{...\}\}\}\}$. However, our lower bound holds for any representation that places elements from the same colour class at the same "nesting depth" within the h.f. set (see Section 4 for the formal definition). This is sufficient because even a representation that does not distinguish colour classes by nesting depth can easily be transformed into the canonical representation above by a CPT-program.

▶ **Theorem 2.** *Let $(H_n)_{n\in\mathbb{N}}$ be the sequence of n-dimensional hypercubes. In each H_n, fix any preorder $\prec_n$ on the vertex set with colour classes of size $\mathcal{O}(n) = \mathcal{O}(\log |H_n|)$. Let x_n be any* symmetric *(w.r.t. H_n) h.f. set over H_n that contains a h.f. representation of $\prec_n$. Then there exists no polynomial p such that x_n is also p-*bounded *w.r.t. the corresponding H_n.*

The proof can be found in Section 6. As already explained, this implies the following nondefinability statement for CPT.

▶ **Corollary 3.** *There is no CPT-program that computes in every hypercube H_n a (h.f. set representation of a) total preorder with colour classes of size $\mathcal{O}(n)$.*

3 Previous work and the significance of undefinable preorders

As already mentioned, our contribution is a non-definability result for a *functional problem*, the computation of certain preorders.

However, our research is motivated by the study of a *decision problem* which is seen as a potential candidate for the separation of CPT from PTIME: The so-called *CFI problem*, that we briefly introduce next. Whether CFI in its general version is solvable in CPT is an open question, but at least for restricted versions, where the structures possess a certain degree of built-in order, it is known to be in CPT. Our non-definability result implies that being able to solve the restricted version of CFI in CPT is of no help for solving CFI in the general case.

The CFI problem

For a detailed account of the CFI problem and the construction of the so-called CFI graphs, we refer the reader to the original paper [3] by Cai, Fürer and Immerman. Here, we only review it to an extent sufficient for our purposes.

Essentially, CFI is the Graph Isomorphism problem on specific pairs of graphs that are obtained by applying the so-called CFI construction to a family of connected graphs, for example, to hypercubes. These are referred to as the *underlying graphs*. The construction replaces every edge and every vertex of the underlying graph with a gadget. Importantly, the symmetries of the underlying graph are preserved this way. Any underlying graph G can be transformed into an odd and an even CFI graph, G_0 and G_1. It holds $G_0 \ncong G_1$, and there is a simple polynomial time algorithm which can determine, given a CFI graph G_x, whether it is odd or even, i.e. if $G_x \cong G_0$, or $G_x \cong G_1$. This is what the CFI problem asks for. However, on the logical side, that is, in FO with counting, G_0 and G_1 can only be distinguished with a linear number of variables. As a consequence, no FPC-sentence can solve the CFI-problem (on a suitable class of underlying graphs). Since this very expressive "reference logic" within PTIME fails to solve CFI, this raises the question whether CPT is strong enough to achieve this, or if CFI is indeed a problem that separates CPT from P.

Solving CFI in CPT

If the underlying graphs of the CFI construction satisfy certain properties, then CFI can be solved in CPT:

▶ **Theorem 4** ([14]). *Let $\mathcal{K}$ be the class of connected, preordered graphs $G = (V, E, \prec)$ where the size of each colour class is bounded by $\log |V|$. The CFI problem on underlying graphs in $\mathcal{K}$ can be solved in Choiceless Polynomial Time.*

This is the strongest known positive result concerning CFI and CPT. It is a generalisation of the CPT-algorithm by Dawar, Richerby and Rossman from [7] for the CFI problem on linearly ordered graphs. Note that not the CFI graphs G_0, G_1 are ordered/preordered in these settings, but only the underlying graph G (otherwise, the Immerman-Vardi Theorem could be applied). The order/preorder on G allows for the algorithmic creation of a so-called "super-symmetric" h.f. object with polynomial orbit which makes it possible to determine the parity of the input CFI graph G_x. This object reflects in its structure the preorder on the input, and is therefore not definable in unordered inputs according to our Theorem 2: It can be checked that Theorem 2 not only holds for hypercubes but also for the CFI graphs obtained from them; this is true because $\text{Aut}(G)$ embeds into $\text{Aut}(G_x)$ for any graph G and corresponding CFI graph G_x. Hence, the algorithmic technique that proves Theorem 4 cannot be generalised to the CFI problem on unordered graphs. In fact, any CPT algorithm that is to solve the unordered CFI problem must avoid the construction of a h.f. object whose nesting structure induces a too fine preorder on the input.

We remark that there are of course families of graphs where the undefinability of such preorders is much easier to show than on hypercubes. For instance, on complete graphs, it is clear that the orbit of a preorder with logarithmic colour classes grows super-polynomially. However, the size of any CFI graph G_0 is exponential in the maximal degree of G. Therefore, the polynomial resources of CPT suffice to solve CFI on unordered graphs of linear maximal degree (this is another result from [14]). Hence, complete graphs do not yield hard CFI instances. In contrast, CFI on hypercubes is well-suited as a benchmark for CPT because their degree is logarithmic and thus the CFI construction only increases the size polynomially.

Our lower bound is a first piece of evidence that the CFI problem on hypercubes is hard (and maybe even unsolvable) for CPT and we believe that it deserves further investigation. The results in [7] indirectly suggest a systematic way to do so: Namely, Dawar, Richerby and Rossman showed that – as long as the CFI structures satisfy a certain homogeneity condition – solving the CFI problem in CPT always requires the construction of a h.f. set which contains a large subset of the input structure as atoms. If it were possible to show that no sufficiently large h.f. object over hypercubes has a polynomial orbit, then this could be used to separate CPT from PTIME. Our result is a step in that direction as it suggests that this large object cannot be structurally similar to a preorder.

4 Analysing orbits of hereditarily finite objects over hypercubes

Let $H_n = (V_n, E_n)$ be the n-dimensional hypercube, i.e. $V_n = \{0,1\}^n$,
$E_n = \{\{u,v\} \in V_n^2 \mid d(u,v) = 1\}$, where $d(u,v)$ is the Hamming-distance.
It is well-known that its automorphism group $\text{Aut}(H_n)$ is isomorphic to the semidirect product of Sym_n and $(\{0,1\}^n, \oplus)$, where Sym_n is the symmetric group on $[n] = \{1, 2, ..., n\}$, and $(\{0,1\}^n, \oplus)$ is the group formed by the length-n binary strings together with the bitwise XOR-operation. More precisely, any automorphism $\sigma \in \text{Aut}(H_n)$ corresponds to the pair $(\pi, w) \in \text{Sym}_n \times \{0,1\}^n$ with $\sigma(v) = v^\pi \oplus w$, where $v^\pi = v_{\pi^{-1}(1)} v_{\pi^{-1}(2)} ... v_{\pi^{-1}(n)}$ (i.e. v^π is obtained from v by permuting the positions of the word according to π). This means: $|\text{Aut}(H_n)| = n! \cdot 2^n$. Note that it is the factor $n!$ which makes the size of this group super-polynomial in $|V_n| = 2^n$.

Our main technical theorem, Theorem 13 concerns a fixed sequence of h.f. objects over the n-dimensional hypercubes, $(x_n)_{n \in \mathbb{N}}$, where $x_n \in \text{HF}(V_n)$. We aim for a lower bound on the orbit size of the objects x_n w.r.t. the action of $\text{Aut}(H_n)$ extended to $\text{HF}(V_n)$. For our purposes, it only matters whether this lower bound is super-polynomial in $2^n = |V_n|$, or not. For this question, we can restrict ourselves to automorphisms corresponding to permutation-word pairs of the form $(\pi, 0^n)$, for $\pi \in \text{Sym}_n$. Therefore, for the rest of this paper, we simply let Sym_n act on V_n by permuting the positions of the binary strings as described above. In this sense, Sym_n embeds into $\text{Aut}(H_n)$, and hence, whenever an object $x \in \text{HF}(V_n)$ has a super-polynomial orbit with respect to this action of Sym_n, this is also true with respect to the action of $\text{Aut}(H_n)$.

To sum up, our task is to lower-bound the orbit-sizes of h.f. objects over length-n binary strings with respect to Sym_n acting on the positions of the strings. We do this via the Orbit-Stabiliser Theorem. Let $\rho : \text{Sym}_n \longrightarrow \text{Aut}(H_n)$ denote the aforementioned embedding. For the rest of the paper, let $\text{Stab}_n(x_n)$ and $\text{Orbit}_n(x_n)$ denote the stabiliser and orbit, respectively, of x_n w.r.t. the action of Sym_n on the string positions:

$$\text{Stab}_n(x_n) := \{\pi \in \text{Sym}_n \mid x_n^{\rho(\pi)} = x_n\}, \ \text{Orbit}_n(x_n) := \{x_n^{\rho(\pi)} \mid \pi \in \text{Sym}_n\}.$$

(we will usually write x^π instead of $x^{\rho(\pi)}$).

▶ **Proposition 5** (Orbit-Stabiliser).

$$|Orbit_n(x_n)| = \frac{|Sym_n|}{|Stab_n(x_n)|} = \frac{n!}{|Stab_n(x_n)|}.$$

This means that we have to upper-bound $|\text{Stab}_n(x_n)|$. As arbitrary nested sets are not easy to handle, we will not analyse x_n directly, but work with an abstraction that is just a collection of sets over $V_n = \{0,1\}^n$. We call these sets the *levels* of x_n. To define them formally, let $\text{HF}_n := (\text{HF}(V_n), \in)$ be the directed acyclic graph whose nodes are all h.f. objects over V_n and whose edges are given by the element-relation. Then,

$$\text{Level}_i(x_n) := \{v \in V_n \mid \text{in HF}_n \text{ there is an } \in\text{-path of length } i \text{ from } x_n \text{ to } v\}.$$

We let $I(x_n) \subseteq \mathbb{N}$ be the index-set of the non-empty levels of x_n, i.e. $I(x_n) := \{i \in \mathbb{N} \mid \text{Level}_i(x_n) \neq \emptyset\}$. It is easy to see that any automorphism that stabilises x_n must stabilise each of its levels (not necessarily pointwise, but as a set).

▶ **Proposition 6.**

$$Stab_n(x_n) \subseteq \bigcap_{i \in I(x_n)} Stab_n(Level_i(x_n)).$$

In other words, we have reduced our problem to upper-bounding the size of the simultaneous stabiliser group of a collection of sets of bitstrings. In the next section, we introduce a tool that we need in order to accomplish this: So-called *supporting partitions.*

5 Approximating permutation groups with supporting partitions

The notions and results in this section are mostly taken from the paper on Symmetric Circuits and FPC by Anderson and Dawar [1].

▶ **Definition 7.** *Let $\mathcal{P}$ be a partition of $[n]$.*

- *The* pointwise *stabiliser of $\mathcal{P}$ is $Stab^\bullet_n(\mathcal{P}) := \{\pi \in Sym_n \mid \pi(P) = P \text{ for all } P \in \mathcal{P}\}$.*
- *The* setwise *stabiliser of $\mathcal{P}$ is $Stab_n(\mathcal{P}) := \{\pi \in Sym_n \mid \pi(P) \in \mathcal{P} \text{ for all } P \in \mathcal{P}\}$ (these are all $\pi \in Sym_n$ that induce a permutation on the parts of $\mathcal{P}$).*

▶ **Definition 8** (Supporting Partition, [1])**.** *Let $G \subseteq Sym_n$ be a group. A supporting partition $\mathcal{P}$ of G is a partition of $[n]$ such that $Stab^\bullet_n(\mathcal{P}) \subseteq G$.*

A group $G \subseteq \mathrm{Sym}_n$ may have several supporting partitions but there always exists a unique *coarsest supporting partition.* A partition $\mathcal{P}'$ is as coarse as a partition $\mathcal{P}$, if every part in $\mathcal{P}$ is contained in some part in $\mathcal{P}'$. For any two partitions $\mathcal{P}, \mathcal{P}'$ there exists a finest partition $\mathcal{E}(\mathcal{P}, \mathcal{P}')$ that is as coarse as either of them:

▶ **Definition 9** ([1])**.** *Let $\mathcal{P}, \mathcal{P}'$ be partitions of $[n]$. Let $\sim$ be a binary relation on $[n]$ such that $x \sim y$ iff there exists a part $P \in \mathcal{P}$ or $P \in \mathcal{P}'$ such that $x, y \in P$. Then $\mathcal{E}(\mathcal{P}, \mathcal{P}')$ is the partition of $[n]$ whose parts are the equivalence classes of $[n]$ under the transitive closure of $\sim$.*

As shown in [1], the property of being a supporting partition of a group $G \subseteq \mathrm{Sym}_n$ is preserved under the operation $\mathcal{E}$. Therefore it holds:

▶ **Lemma 10** ([1])**.** *Each permutation group $G \subseteq Sym_n$ has a unique* coarsest supporting partition*, denoted $SP(G)$.*

When we write $\mathrm{SP}(a)$ for $a \in \mathrm{HF}(\{0,1\}^n)$, we mean $\mathrm{SP}(\mathrm{Stab}_n(a))$, that is, the coarsest supporting partition of the stabiliser of a, where – as in the previous section – we consider the stabiliser as the subgroup of Sym_n acting on the positions of the binary strings. Note that if $a \in \{0,1\}^n$, then $\mathrm{SP}(a)$ is just the partition of $[n]$ into $\{k \in [n] \mid a_k = 0\}$ and $\{k \in [n] \mid a_k = 1\}$.

The reason why coarsest supporting partitions are useful for estimating the sizes of certain stabiliser subgroups is the following result:

▶ **Lemma 11** ([1])**.** *Let $G \subseteq Sym_n$ be a group. Then:*

$$Stab^\bullet_n(SP(G)) \subseteq G \subseteq Stab_n(SP(G)).$$

This lemma enables us to upper-bound stabilisers of arbitrary objects in $\mathrm{HF}(\{0,1\}^n)$ by the stabilisers of their supporting partitions.
Finally, because we will frequently need it later in our proof, we define the operation $\sqcap$ as the "intersection" of two partitions:

▶ **Definition 12** (Intersection of partitions)**.** *Let $\mathcal{P}, \mathcal{P}'$ be partitions of $[n]$. The* intersection *$\mathcal{P} \sqcap \mathcal{P}'$ is defined like this:*

$$\mathcal{P} \sqcap \mathcal{P}' := \{\mathcal{P}(k) \cap \mathcal{P}'(k) \mid k \in [n]\}.$$

Here, $\mathcal{P}(k), \mathcal{P}'(k)$ denote the parts of the respective partition that contain k.

6 The Super-Polynomial Orbit Theorem

Our main technical theorem reads as follows:

▶ **Theorem 13.** *Let $(x_n)_{(n \in \mathbb{N})}$ be a sequence with $x_n \in HF(V_n)$ (recall that $V_n = \{0,1\}^n$). Assume that the x_n satisfy the following two properties:*

1. *In each x_n, every $v \in V_n$ occurs as an atom.*
2. *The function $\max_{i \in I(x_n)} |Level_i(x_n)|$ is in $\mathcal{O}(n)$.*

Then, $|Orbit_n(x_n)|$ (as defined in Section 4) grows asymptotically faster than any polynomial in $2^n = |V_n|$.

From this, Theorem 2 follows because – as discussed in Section 2 – the canonical h.f. set representation of a preorder with logarithmic colour classes satisfies the two conditions of Theorem 13, and because any *symmetric* (see Definition 1) h.f. object that contains x_n must necessarily contain $\text{Orbit}_n(x_n)$, too.
We start to explain the proof idea of Theorem 13 by stating the following summary of Proposition 6 and Lemma 11:

▶ **Corollary 14.**

$$Stab_n(x_n) \subseteq \bigcap_{i \in I(x_n)} Stab_n(Level_i(x_n))) \subseteq \bigcap_{i \in I(x_n)} Stab_n(SP(Level_i(x_n))).$$

We are going to employ the Orbit-Stabiliser Theorem in order to obtain our lower bound for the orbit size. Hence, we need to bound $|\text{Stab}_n(x_n)|$ from above, and Corollary 14 already indicates the basic principle of our proof: Splitting up x_n into its levels and analysing the stabilisers of their respective supporting partitions.
Our analysis of $|\text{Stab}_n(x_n)|$ is divided into two main cases that we treat separately. The distinction is with respect to the maximum size of the coarsest support of any level of x_n, viewed as a function of n:

Let $B_n \subseteq \{0,1\}^n$ be the level of x_n such that $|\text{SP}(B_n)|$ (i.e. its number of parts) is maximal in $\{|\text{SP}(\text{Level}_i(x_n))| \mid i \in I(x_n)\}$. Then the two cases we distinguish are:

(1) The maximal level-support size grows sublinearly: $|\text{SP}(B_n)| \in o(n)$.
(2) The maximal level-support size grows linearly: $|\text{SP}(B_n)| \in \Theta(n)$.

We deal with the two cases in the next two subsections. Their results are summarised in Lemma 15 and Lemma 21. Together they imply the theorem. Due to space restrictions, we can only give proof sketches for most lemmas; for some of them, full proofs can be found in the appendix. In the following lemmas, we always refer to the objects and the setting of Theorem 13, as well as to the set level B_n just defined.

6.1 The case of sublinearly bounded supports

The result of this subsection is:

▶ **Lemma 15.** *Assume the following three conditions hold:*

1. *In each x_n, every $v \in V_n$ occurs as an atom.*
2. *The function $\max_{i \in I(x_n)} |Level_i(x_n)|$ is in $\mathcal{O}(n)$.*
3. *$|SP(B_n)| \in o(n)$.*

Then the orbit size of x_n w.r.t. Sym_n acting on the positions of the binary strings grows faster than any polynomial in 2^n.

We prove this lemma on the next few pages. From now on, we use the abbreviation $\mathrm{SP}_i(x_n) := \mathrm{SP}(\mathrm{Level}_i(x_n))$. Let us begin by outlining the proof idea. We have to bound $\mathrm{Stab}_n(x_n) \subseteq \bigcap_{i\in I(x_n)} \mathrm{Stab}_n(\mathrm{SP}_i(x_n))$ (see Corollary 14). Hence, we have to count the permutations in Sym_n that simultaneously stabilise the supports of the levels of x_n.
For a level $i \in I(x_n)$, $\mathrm{Sym}(\mathrm{SP}_i(x_n))$ denotes the symmetric group on the parts of $\mathrm{SP}_i(x_n)$ (in contrast, Sym_n is the symmetric group on the set $[n]$ that underlies this partition). Every $\pi \in \mathrm{Sym}_n$ that stabilises $\mathrm{SP}_i(x_n)$ as a set *induces* (or *realises*) a $\sigma \in \mathrm{Sym}(\mathrm{SP}_i(x_n))$ in the sense that $\sigma(P) = \{\pi(k) \mid k \in P\} \in \mathrm{SP}_i(x_n)$ for all $P \in \mathrm{SP}_i(x_n)$. This can also be extended to a set $J \subseteq I(x_n)$ of several levels: Every $\pi \in \bigcap_{i\in J} \mathrm{Stab}_n(\mathrm{SP}_i(x_n))$ induces a $\overline{\sigma} \in \bigtimes_{i\in J} \mathrm{Sym}(\mathrm{SP}_i(x_n))$. Here, $\overline{\sigma}$ is the tuple of permutations that π realises simultaneously on the parts of the respective $\mathrm{SP}_i(x_n)$.

Now in order to bound $|\mathrm{Stab}_n(x_n)|$, we will choose a subset $J \subseteq I(x_n)$ with certain properties that will enable us to bound two quantities: Firstly, each $\overline{\sigma} \in \bigtimes_{i\in J} \mathrm{Sym}(\mathrm{SP}_i(x_n))$ that can be realised by a $\pi \in \mathrm{Stab}_n(x_n)$ will only have a small number of distinct such realisations. Secondly, there will be a bound on the number of such $\overline{\sigma}$ that can be realised by a $\pi \in \mathrm{Stab}_n(x_n)$ at all. The product of these two bounds is then an upper bound for $|\mathrm{Stab}_n(x_n)|$.

We begin with a lemma that generally relates the number of possible distinct realisations of a given $\overline{\sigma} \in \bigtimes_{i\in[m]} \mathrm{Sym}(\mathrm{SP}(A_i))$, for sets $A_1, ..., A_m \subseteq \{0,1\}^n$, with the partition $\bigsqcap_{i=1}^m \mathrm{SP}(A_i)$ (recall Definition 12 for the meaning of $\sqcap$).

▶ **Lemma 16.** *Let $A_1, ..., A_m \subseteq \{0,1\}^n$ be a collection of sets of bitstrings. Fix any simultaneous permutation $\overline{\sigma}$ of the parts of the supports of the sets, i.e. $\overline{\sigma} \in \bigtimes_{i=1}^m \mathit{Sym}(\mathit{SP}(A_i))$. There exists a $\vartheta_{\overline{\sigma}} \in \mathit{Sym}(\bigsqcap_{i=1}^m \mathit{SP}(A_i))$ such that every $\pi \in \mathit{Sym}_n$ that realises $\overline{\sigma}$ also realises $\vartheta_{\overline{\sigma}}$.*

Proof sketch. Via induction on m. If $m = 1$, then the desired $\vartheta_{\overline{\sigma}}$ is just $\overline{\sigma} \in \mathrm{Sym}(\mathrm{SP}(A_1))$. For the induction step, let there be $m+1$ sets $A_1, ..., A_{m+1}$, and let $\overline{\sigma} \in \bigtimes_{i=1}^{m+1} \mathrm{Sym}(\mathrm{SP}(A_i))$ be fixed. Since every $\pi \in \mathrm{Sym}_n$ that realises $\overline{\sigma}$ in particular realises the first m entries in $\overline{\sigma}$, the induction hypothesis gives us a fixed permutation in $\mathrm{Sym}(\bigsqcap_{i=1}^m \mathrm{SP}(A_i))$ that each such π has to realise. Further, π has to realise $\overline{\sigma}_{m+1} \in \mathrm{Sym}(\mathrm{SP}(A_{m+1}))$. Putting these constraints on π together, the desired $\vartheta_{\overline{\sigma}} \in \mathrm{Sym}(\bigsqcap_{i=1}^{m+1} \mathrm{SP}(A_i))$ is obtained. ◀

So, intuitively speaking, the finer the partition $\bigsqcap_{i=1}^m \mathrm{SP}(A_i)$ is, the fewer realisations exist for any $\overline{\sigma} \in \bigtimes_{i=1}^m \mathrm{Sym}(\mathrm{SP}(A_i))$. Therefore, we will aim to select a subset of the levels of x_n such that the intersection over the supports is as fine as possible. More precisely, we would like it to consist of many singleton parts. For the rest of this subsection we denote by $S_n \subseteq [n]$ the set of positions which are in singleton parts in $\bigsqcap_{i\in I(x_n)} \mathrm{SP}_i(x_n)$, i.e.

$$S_n := \{k \in [n] \mid \{k\} \in \bigsqcap_{i\in I(x_n)} \mathrm{SP}_i(x_n)\}.$$

It turns out that there can only be few positions which are *not* in singleton parts in $\bigsqcap_{i\in I(x_n)} \mathrm{SP}_i(x_n)$; this is a consequence of the assumption that x_n contains every element of $\{0,1\}^n$, together with the size bound on the levels:

▶ **Lemma 17.** *Assume that $|\mathit{Level}_i(x_n)| \in \mathcal{O}(n)$ for each level i of x_n. Further, assume that in each x_n, every element of $\{0,1\}^n$ occurs as an atom. Then, for large enough n:*

$$|[n] \setminus S_n| < 8 \log n.$$

Proof sketch. Assume: $|[n] \setminus S_n| \geq 8 \log n$. We show that this entails the existence of a level $A \subseteq \{0,1\}^n$ in the object x_n such that $|A|$ is greater than $\mathcal{O}(n)$, which is a contradiction. The partition $\mathrm{SP}(A)$ is as least as coarse as $\bigsqcap_{i \in I(x_n)} \mathrm{SP}_i(x_n)$ and has therefore a certain number of positions within non-singleton parts according to our assumption. Permuting the positions within the non-singleton parts of $\mathrm{SP}(A)$ leaves the set A intact, by definition of supporting partitions. Hence, if A contains a string a with a balanced number of zeroes and ones within each $P \in \mathrm{SP}(A)$ with $|P| \geq 2$, it can be calculated that a has more than $\mathcal{O}(n)$ images under the mentioned permutations. Because every $a \in \{0,1\}^n$ occurs somewhere in x_n, such a level A of x_n indeed exists. ◀

We proceed to construct the announced subset $J \subseteq I(x_n)$ of the levels of x_n. Its two properties that are stated in the next lemma are crucial to bound $|\bigcap_{i \in J} \mathrm{Stab}_n(\mathrm{SP}_i(x_n))|$. A full proof of the lemma is included in the appendix.

▶ **Lemma 18.** *Let $f(n) \in o(n)$ such that for all levels $i \in I(x_n)$, $|SP_i(x_n)| \leq f(n)$. There exists a subset $J \subseteq I(x_n)$ of the levels of x_n with the following two properties:*

(1) *Every position in S_n is also in a singleton part of $\bigsqcap_{j \in J} SP_j(x_n)$.*

(2) *The following bound for the number of realisable simultaneous permutations of the supporting partitions holds:*

$$\Big|\{\overline{\sigma} \in \bigtimes_{j \in J} \mathit{Sym}(SP_j(x_n)) \mid \text{there is a } \pi \in \mathit{Sym}_n \text{ that realises } \overline{\sigma}\}\Big| \leq (f(n)!)^{n/(f(n)-1)} \cdot 2^n$$

Proof sketch. Construct the set J stepwise, starting with $J^0 = \emptyset$, and adding a new level j_i in each step i, such that $\bigsqcap_{j \in J^i} \mathrm{SP}_j(x_n)$ is a strict refinement of $\bigsqcap_{j \in J^{i-1}} \mathrm{SP}_j(x_n)$. This is done until property (1) is satisfied. Let

$$k_i := \Big|\bigsqcap_{j \in J^i} \mathrm{SP}_j(x_n)\Big| - \Big|\bigsqcap_{j \in J^{i-1}} \mathrm{SP}_j(x_n)\Big|.$$

The main part of the proof is to show that in step $i+1$, there are at most $(k_{i+1}+1)!$ permutations in $\mathrm{Sym}(\mathrm{SP}_{j_{i+1}}(x_n))$ that can be realised by some $\pi \in \mathrm{Sym}_n$ simultaneously with any other given $\overline{\sigma} \in \bigtimes_{j \in J^i} \mathrm{Sym}(\mathrm{SP}_j(x_n))$. Once this is established, we know that the set of simultaneous permutations from property (2) has size at most $\prod_{i=1}^{s}(k_i+1)!$. Further, for all i we have $|\mathrm{Sym}(\mathrm{SP}_i(x_n))| \leq f(n)!$. Using the fact that $\sum_{i=1}^{s} k_i \leq n$, one can now make a typical "redistribute weight argument" to show that the mentioned product of factorials is maximised if each k_i is either 1 or $f(n)-1$. This leads to the bound stated in property (2). ◀

▶ **Corollary 19.** *Assume the following three conditions hold:*

1. *In each x_n, every $v \in V_n$ occurs as an atom.*
2. *The function $\max_{i \in I(x_n)} |Level_i(x_n)|$ is in $\mathcal{O}(n)$.*
3. *$|SP(B_n)| \in o(n)$.*

Then, for sufficiently large n:

$$|\mathit{Stab}_n(x_n)| \leq (f(n)!)^{n/(f(n)-1)} \cdot 2^n \cdot (8 \log n)!$$

Proof. Consider the set $J \subseteq I(x_n)$ that exists by Lemma 18. By Corollary 14, every $\pi \in \mathrm{Stab}_n(x_n)$ induces a tuple of permutations $\overline{\sigma} \in \bigtimes_{i \in I(x_n)} \mathrm{Sym}(\mathrm{SP}_i(x_n))$, so in particular it also induces a $\overline{\sigma} \in \bigtimes_{i \in J} \mathrm{Sym}(\mathrm{SP}_i(x_n))$. By Lemma 18, there are at most $(f(n)!)^{n/(f(n)-1)} \cdot 2^n$ possibilities for such a $\overline{\sigma}$. Furthermore, each such $\overline{\sigma}$ can be realised by at most $(8 \log n)!$

distinct permutations $\pi \in \mathrm{Stab}_n(x_n)$: Due to Lemma 16 and property (1) of J (see Lemma 18), every π realising $\overline{\sigma}$ permutes the positions in S_n in the same way, and according to Lemma 17, there remain at most $8 \log n$ positions which may be permuted arbitrarily by π (that is, if all positions in $[n] \setminus S_n$ form a single part in $\bigsqcap_{i \in J} \mathrm{SP}_i(x_n)$). ◀

With this, we can estimate the asymptotic behaviour of $|\mathrm{Orbit}_n(x_n)|$, which proves Lemma 15.

▶ **Lemma 20.** *Under the assumptions of Corollary 19, $|Orbit_n(x_n)|$ can be estimated as follows: For any $k \in \mathbb{N}$, the limit*

$$\lim_{n\to\infty} \frac{|Orbit_n(x_n)|}{2^{kn}} = \lim_{n\to\infty} \frac{n!}{|Stab_n(x_n)| \cdot 2^{kn}} \geq \lim_{n\to\infty} \frac{n!}{(f(n)!)^{n/(f(n)-1)} \cdot 2^n \cdot (8 \log n)! \cdot 2^{kn}}$$

does not exist. That is to say, the orbit of x_n w.r.t. the action of Sym_n grows super-polynomially in 2^n.

Proof sketch. Replace all factorials with the Stirling Formula $n! \approx \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n$. With this, compute a lower bound for the above fraction that can be seen to tend to infinity as n grows. ◀

6.2 The case of linearly-sized supports

This subsection is dedicated to proving the following result for the case that $|\mathrm{SP}(B_n)| \in \Theta(n)$. In this case, we only need to analyse the orbit size of the level B_n of x_n with the largest supporting partition (see the beginning of Section 6 again for the definition of B_n).

▶ **Lemma 21.** *Assume that the following conditions hold for B_n:*

1. $|B_n| \in \mathcal{O}(n)$.
2. $|SP(B_n)| \in \Theta(n)$.

Then the orbit size of B_n (and therefore also of x_n) w.r.t. the action of Sym_n on the positions of the binary strings grows faster than any polynomial in 2^n.

Proving this lemma requires a case distinction again. The relevant measure here is the number of *singleton* parts in $\mathrm{SP}(B_n)$. Firstly, we show that if the number of singleton parts in $\mathrm{SP}(B_n)$ grows sublinearly in n, while the total number of parts $|\mathrm{SP}(B_n)|$ is linear, the stabiliser of $\mathrm{SP}(B_n)$ is small enough. This can be seen solely from the properties of the partition $\mathrm{SP}(B_n)$.

The difficult part of the proof is the case where the number of singleton parts grows linearly. In the worst case, $\mathrm{SP}(B_n)$ consists only of singletons; then, $\mathrm{Stab}_n(\mathrm{SP}(B_n)) = \mathrm{Sym}_n$. We solve this by not only looking at the partition $\mathrm{SP}(B_n)$ itself but also at properties of the set B_n that can be inferred from its supporting partition.
In the following, we always denote by $S_n \subseteq [n]$ the set of positions that are in singleton parts of $\mathrm{SP}(B_n)$, i.e.

$$S_n := \{k \in [n] \mid \{k\} \in \mathrm{SP}(B_n)\}.$$

(note that the definition of S_n was slightly different in the last subsection).

Subcase 1: Sublinear number of singleton parts

Let us begin with the easier case, where the number of singleton parts in $\mathrm{SP}(B_n)$ grows sublinearly. The size of $\mathrm{Stab}_n(\mathrm{SP}(B_n))$ can generally be bounded as follows:

▶ **Lemma 22.** *Let $s_n := |S_n|$, and $t_n := |SP(B_n)| - s_n$.*

$$|Stab_n(SP(B_n))| \leq s_n! \cdot t_n! \cdot (n - 2(t_n - 1))! \cdot 2^{t_n}.$$

Proof. The factors $s_n!$ and $t_n!$ account for the possible permutations of the parts: All the singleton parts of $\mathrm{SP}(B_n)$ can be mapped to each other, and every non-singleton part can at most be mapped to every other non-singleton part. An upper bound on the number of permutations within the non-singleton parts is $\ell_1! \cdot \ell_2! \cdot ... \cdot \ell_{t_n}!$, where the ℓ_i are the sizes of these parts. This product of factorials is maximised if one value ℓ_p is as large as possible ($\leq n - 2(t_n - 1)$), and $\ell_i = 2$ for all $i \neq p$. This is a standard "redistribute weight argument", which is also used in [1] multiple times. ◀

▶ **Corollary 23.** *Let $f(n) \in o(n)$ be a function such that $s_n \leq f(n)$ and let $c \leq 1$ be a positive constant such that $|SP(B_n)| \geq c \cdot n$ for large enough n. Then, for large enough n, the following bound holds:*

$$|Stab_n(SP(B_n))| \leq f(n)! \cdot 2^n \cdot \max\Big\{(n/2)! \cdot 2, (cn - f(n))! \cdot ((1 - 2c)n + 2f(n) + 2)!\Big\}.$$

Proof. We plug in the right values for s_n and $t_n = |\mathrm{SP}(B_n)| - s_n$ into Lemma 22, and use the simple bound $2^{t_n} \leq 2^n$. We have $s_n \leq f(n)$ by assumption. As $|\mathrm{SP}(B_n)| \geq c \cdot n$, and because every non-singleton part has at least two elements, we can bound t_n as follows:

$$c \cdot n - f(n) \leq t_n \leq \frac{n}{2}.$$

The bound from Lemma 22 contains a product of two factorials which both depend on t_n. By a redistribute-weight argument, one can see that this product is maximised if the two factorials are maximally imbalanced. This happens if t_n attains its maximum or minimum. ◀

This directly leads to a super-polynomial orbit: In the next lemma, we calculate the growth of $|\mathrm{Orbit}_n(x_n)| \geq \frac{n!}{|\mathrm{Stab}_n((\mathrm{SP}(B_n))|}$ (this is due to Lemma 11), using the stabiliser-bound from Corollary 23.

▶ **Lemma 24.** *Let $f(n) \in o(n)$ be a function such that $s_n \leq f(n)$ and $c \leq 1$ be a positive constant such that $|SP(B_n)| \geq c \cdot n$ for large enough n. Then for any $k \in \mathbb{N}$, the limit*

$$\lim_{n\to\infty} \frac{n!}{f(n)! \cdot 2^n \cdot \max\Big\{(n/2)! \cdot 2, (cn - f(n))! \cdot ((1 - 2c)n + 2f(n) + 2)!\Big\} \cdot 2^{kn}}$$

does not exist. That is to say, the orbit of B_n w.r.t. the action of Sym_n grows super-polynomially in 2^n.

Proof sketch. Similar to Lemma 20. ◀

This proves Lemma 21 under the assumption that $|S_n| \in o(n)$.

Subcase 2: Linear number of singleton parts

The idea for this case is similar to how we solved the case of sublinear supports. There, we related simultaneous permutations of the parts of the supports $\mathrm{SP}_i(x_n)$ to their realisations in Sym_n. Now we do the same with respect to permutations of the elements of B_n: Let $\mathrm{Sym}(B_n)$ be the group of all permutations of the strings in B_n. For $\pi \in \mathrm{Sym}_n$ and $\sigma \in \mathrm{Sym}(B_n)$, we say that π *realises* or *induces* σ, if $b^\pi = \sigma(b)$ for every $b \in B_n$. The aim is to show that only a bounded number of $\sigma \in \mathrm{Sym}(B_n)$ can be realised by a permutation $\pi \in \mathrm{Stab}_n(B_n)$ at all, and that each such σ only has a small number of realisations. In total, this yields a bound on $|\mathrm{Stab}_n(B_n)|$.

We will construct a subset $A \subseteq B_n$ such that: Any $\sigma \in \mathrm{Sym}(B_n)$ whose preimages are fixed on A can only be realised by few $\pi \in \mathrm{Sym}_n$, and A is small compared to B_n. This ensures that there are not too many ways to specify a $\sigma \in \mathrm{Sym}(B_n)$ on A (if $|B_n| \in \mathcal{O}(n)$, there are $\leq n^{|A|}$ options to fix $\sigma^{-1}(a)$ for all $a \in A$).

First of all, we show how to bound the number of possible realisations of any $\sigma \in \mathrm{Sym}(B_n)$ if σ is fixed on some subset $A \subseteq B_n$. The next lemma is similar to Lemma 16. We omit the proof as it is quite analogous.

▶ **Lemma 25.** *Let $B \subseteq \{0,1\}^n, A \subseteq B$. Let an injective mapping $p : A \longrightarrow B$ be given. Write $\bigsqcap A := \bigsqcap_{a\in A} SP(a)$.*
There is an assignment of positions to parts $Q_p : [n] \longrightarrow \bigsqcap A$ with the property that $|Q_p^{-1}(P)| = |P|$ for every $P \in \bigsqcap A$, and such that:
Every $\pi \in Sym_n$ realising any $\sigma \in Sym(B)$ with $\sigma^{-1}(a) = p(a)$ (if such a π exists) satisfies: $\pi(k) \in Q_p(k)$ for all $k \in [n]$.

We will mainly need this lemma for the restriction of the parts in $\bigsqcap A$ to the positions in S_n. Therefore, we state the following important corollary:

▶ **Corollary 26.** *Let $A \subseteq B_n$ be arbitrary, and let an injective mapping $p : A \longrightarrow B_n$ be given. Then every $\pi \in Sym_n$ that realises a $\sigma \in Sym(B_n)$ with $\sigma^{-1}(a) = p(a)$ for all $a \in A$ satisfies:*

$$\pi^{-1}(P \cap S_n) = Q_p^{-1}(P) \cap S_n \text{ for all } P \in \bigsqcap A,$$

where $Q_p : [n] \longrightarrow \bigsqcap A$ is the assignment that exists by the preceding lemma.

Proof. Lemma 25 says that $\pi^{-1}(P) = Q_p^{-1}(P)$. Since π realises a permutation in $\mathrm{Sym}(B_n)$, $\pi \in \mathrm{Stab}_n(B_n)$. Hence, by Lemma 11, $\pi \in \mathrm{Stab}_n(\mathrm{SP}(B_n))$. This means that $\pi(S_n) = S_n$, as singleton parts can only be mapped to singleton parts. Consequently, it must be the case that $\pi^{-1}(P \cap S_n) = Q_p^{-1}(P) \cap S_n$. ◀

Next, we select our desired subset $A \subseteq B_n$. It will be such that the partition $\bigsqcap A = \bigsqcap_{a\in A} \mathrm{SP}(a)$ is quite fine on S_n. In order to guarantee that A is much smaller than B_n, we only require a relaxed, but more complicated, notion of "fineness" here. The proof of the next lemma can be found in the appendix.

▶ **Lemma 27.** *There exists a subset $A \subseteq B_n$ of size $|A| \leq \frac{|S_n|}{2}$ such that for each part $P \in \bigsqcap A$, one of the following two statements is true:*

1. *$|P \cap S_n| \leq 2$;* ***or:***
2. *$|P \cap S_n| > 2$ and for every $b \in B_n \setminus A$, one of these two conditions holds:*
   - *b is constant on $P \cap S_n$;* ***or***
   - *$b[P \cap S_n]$ is* imbalanced *and, for every $P' \in \bigsqcap A$ with $P' \neq P$, $|P' \cap S_n| > 2$, b is constant on $P' \cap S_n$.*

By $b[P \cap S_n]$ we mean the substring of b at the positions in $P \cap S_n$, and being imbalanced *means that $b[P \cap S_n]$ contains exactly one 0 and there is a 1 at all other positions, or vice versa (exactly one 1 and the rest 0).*

Proof sketch. Construct A stepwise, starting with $A^0 = \emptyset$ and adding a new string a_i in each step i. Choose a_{i+1} such that progress is made. That means, a_{i+1} should split some part $(P \cap S_n)$, for a $P \in \prod A^i$ with $|P \cap S_n| > 2$ ("split" means, a_{i+1} is non-constant on $P \cap S_n$). However, we take care that a_{i+1} either splits two parts, or if it splits only one part, it does not split off a singleton part. This ensures that at most $|S_n|/2$ such construction steps can be performed. If no such $a_{i+1} \in B_n \setminus A^i$ exists, then the constructed set fulfils the properties stated in the lemma. ◀

Before we can use this to bound $|\mathrm{Stab}_n(B_n)|$, we need one more lemma concerning those parts $P \in \prod A$ with $|P \cap S_n| > 2$. We show that any $\pi \in \mathrm{Stab}_n(B_n)$ is already fully determined when it is specified only on the parts $P \in \prod A$ with $|P \cap S_n| \leq 2$. The full proof of this is also in the appendix.

▶ **Lemma 28.** *Let $A \subseteq B_n$ be the subset that exists by Lemma 27, and let $p : A \longrightarrow B_n$ be an injective function. Let*

$$\Gamma_p := \{\pi \in Stab_n(B_n) \mid p(a)^\pi = a \text{ for all } a \in A\}.$$

Further, let

$$P_{>2} := \{k \in S_n \mid |P(k) \cap S_n| > 2, \text{ where } P(k) \in \prod A \text{ is the part that } k \text{ is in}\}.$$

Then for any $\pi, \pi' \in \Gamma_p$ such that $\pi^{-1}|_{([n] \setminus P_{>2})} = \pi'^{-1}|_{([n] \setminus P_{>2})}$, it also holds $\pi^{-1}|_{P_{>2}} = \pi'^{-1}|_{P_{>2}}$.

Proof sketch. Assume for a contradiction the existence of $\pi, \pi' \in \Gamma_p$ such that their preimages are the same on $[n] \setminus P_{>2}$ but there is a position x such that $\pi(x) \in P_{>2}$ and $\pi(x) \neq \pi'(x)$. It can then be shown – using the second statement of Lemma 27 and the fact that $\pi, \pi' \in \mathrm{Stab}_n(B_n)$ – that the transposition $(\pi(x)\ \pi'(x))$ is also an element of $\mathrm{Stab}_n(B_n)$. This, however, is a contradiction to the fact that $\pi(x), \pi'(x)$ are in distinct singleton parts in $\mathrm{SP}(B_n)$, which is the coarsest possible supporting partition. ◀

▶ **Lemma 29.** *Let c be a constant such that $|B_n| \leq c \cdot n$ (for large enough n). Then, for large enough n, it holds:*

$$|Stab_n(B_n)| \leq (2cn)^{|S_n|/2} \cdot (n - |S_n|)!$$

Proof. Let $A \subseteq B_n$ be the subset of B_n whose existence is stated in Lemma 27. Fix any injective function $p : A \longrightarrow B_n$. Let Γ_p and $P_{>2}$ be as in Lemma 28.
We bound $|\Gamma_p|$ by counting the number of possible $\pi \in \Gamma_p$. We know by Lemma 28 that we only have to count the number of possibilities to choose the preimages of the elements in $[n] \setminus P_{>2}$. For every part $P \in \prod A$ with $|P \cap S_n| \leq 2$, we know by Corollary 26 that $\pi^{-1}(P \cap S_n) \subseteq S_n$ is the same fixed set of size ≤ 2 for all $\pi \in \Gamma_p$, so we only have two options how π^{-1} can behave on $P \cap S_n$. The number of such parts P is at most $|S_n|/2$.
For $i \in [n] \setminus S_n$, we can only say that $\pi^{-1}(i) \notin S_n$ (by Lemma 11). Hence, every $\pi \in \Gamma_p$ can in principle permute the set $[n] \setminus S_n$ arbitrarily. In total, we conclude:

$$|\Gamma_p| \leq 2^{(|S_n|/2)} \cdot (n - |S_n|)!$$

This is for a fixed function p. The number of possible choices for p is bounded by $(cn)^{|S_n|/2}$, since $|A| \leq |S_n|/2$ (Lemma 27) and we are assuming $|B_n| \leq cn$.
Every $\pi \in \mathrm{Stab}_n(B_n)$ must occur in at least one of the sets Γ_p for some choice of p, so indeed, $(2cn)^{|S_n|/2} \cdot (n - |S_n|)!$ is an upper bound for $|\mathrm{Stab}_n(B_n)|$. ◀

Based on Lemma 29, the orbit size of B_n can be estimated:

▶ **Lemma 30.** *Let c be a constant such that $|B_n| \leq c \cdot n$, and $\delta > 0$ be a constant such that $|S_n| \geq \delta \cdot n$ (for large enough n). Then for any $k \in \mathbb{N}$, the limit*

$$\lim_{n\to\infty} \frac{|\mathit{Orbit}_n(B_n)|}{2^{kn}} = \lim_{n\to\infty} \frac{n!}{|\mathit{Stab}_n(B_n)| \cdot 2^{kn}} \geq \lim_{n\to\infty} \frac{n!}{(2cn)^{(\delta n)/2} \cdot ((1-\delta)n)! \cdot 2^{kn}}$$

does not exist. That is to say, the orbit of B_n w.r.t. the action of Sym_n grows super-polynomially in 2^n.

Proof sketch. Again a calculation using Stirling's approximation for the factorials. ◀

This lemma together with Lemma 24 proves Lemma 21.

7 Concluding remarks and future research

A question that remains open is what exactly is the threshold of "fineness" of a preorder where the orbit size changes from super-polynomial to polynomial. In other words: What is the largest colour class size for which our Super-Polynomial Orbit Theorem for hypercubes still holds?
One can check that all parts of our proof can be modified such that it also goes through if we allow colour classes (i.e. levels) of size $o(n^2)$. If the size is in $\Theta(n^2)$, though, the bound in Lemma 29 becomes too large for Lemma 30 to hold.
On the other hand, the finest preorder with a polynomial orbit that we know so far is one where the colour class sizes are in $\mathcal{O}(2^n/\sqrt{n})$: It corresponds to the partition of $\{0,1\}^n$ according to Hamming-weight. Obviously, this is precisely the orbit-partition of the vertex-set (w.r.t. the action of Sym_n on the positions). Its largest colour class has size $\binom{n}{n/2} \in \Theta(2^n/\sqrt{n})$.
Determining the finest preorder that is in principle CPT-definable in hypercubes would potentially allow to better judge whether a preorder-based CPT-algorithm like the one in [14] can at all be a candidate for a solution of the unordered CFI problem.
Moreover, it would be helpful to identify further h.f. objects that are undefinable in hypercubes for symmetry reasons.

References

1 Matthew Anderson and Anuj Dawar. On symmetric circuits and fixed-point logics. *Theory of Computing Systems*, 60(3):521–551, 2017. doi:10.1007/s00224-016-9692-2.

2 Andreas Blass, Yuri Gurevich, and Saharon Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100(1-3):141–187, 1999. doi:10.1016/S0168-0072(99)00005-6.

3 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992. doi:10.1007/BF01305232.

4 Ashok K Chandra and David Harel. Structure and complexity of relational queries. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 333–347. IEEE, 1980. doi:10.1109/SFCS.1980.41.

5 Anuj Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1):8–21, 2015.

6 Anuj Dawar, Martin Grohe, Bjarki Holm, and Bastian Laubner. Logics with rank operators. In *2009 24th Annual IEEE Symposium on Logic In Computer Science*, pages 113–122. IEEE, 2009. doi:10.1109/LICS.2009.24.

7 Anuj Dawar, David Richerby, and Benjamin Rossman. Choiceless polynomial time, counting and the Cai–Fürer–Immerman graphs. *Annals of Pure and Applied Logic*, 152(1-3):31–50, 2008. doi:10.1016/j.apal.2007.11.011.

8 Erich Grädel and Martin Grohe. Is Polynomial Time Choiceless? In *Fields of Logic and Computation II*, pages 193–209. Springer, 2015. doi:10.1007/978-3-319-23534-9_11.

9 Erich Grädel, Wied Pakusa, Svenja Schalthöfer, and Łukasz Kaiser. Characterising Choiceless Polynomial Time with First-order Interpretations. In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 677–688, 2015. doi:10.1109/LICS.2015.68.

10 Martin Grohe. The quest for a logic capturing PTIME. In *2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 267–271. IEEE, 2008. doi:10.1109/LICS.2008.11.

11 Yuri Gurevich. Logic and the Challenge of Computer Science. In *Current Trends in Theoretical Computer Science*. Computer Science Press, 1988.

12 Neil Immerman. Relational queries computable in polynomial time. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 147–152, 1982. doi:10.1145/800070.802187.

13 Wied Pakusa. *Linear Equation Systems and the Search for a Logical Characterisation of Polynomial Time*. PhD thesis, RWTH Aachen, 2015.

14 Wied Pakusa, Svenja Schalthöfer, and Erkal Selman. Definability of Cai-Fürer-Immerman problems in Choiceless Polynomial Time. *ACM Transactions on Computational Logic (TOCL)*, 19(2):1–27, 2018. doi:10.1145/3154456.

15 Benjamin Rossman. Choiceless computation and symmetry. In *Fields of Logic and Computation, Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume 6300 of *Lecture Notes in Computer Science*, pages 565–580. Springer, 2010. doi:10.1007/978-3-642-15025-8_28.

16 Svenja Schalthöfer. *Choiceless Computation and Logic*. PhD thesis, RWTH Aachen, 2020.

17 Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146, 1982. doi:10.1145/800070.802186.

8 Appendix

8.1 Proof of Lemma 18

▶ **Lemma 18.** *Let $f(n) \in o(n)$ such that for all levels $i \in I(x_n)$, $|SP_i(x_n)| \leq f(n)$. There exists a subset $J \subseteq I(x_n)$ of the levels of x_n with the following two properties:*

(1) *Every position in S_n is also in a singleton part of $\bigsqcap_{j \in J} SP_j(x_n)$.*

(2) *The following bound for the number of realisable simultaneous permutations of the supporting partitions holds:*

$$\left|\{\overline{\sigma} \in \bigtimes_{j \in J} Sym(SP_j(x_n)) \mid \text{there is a } \pi \in Sym_n \text{ that realises } \overline{\sigma}\}\right| \leq (f(n)!)^{n/(f(n)-1)} \cdot 2^n$$

Proof. We construct J stepwise, starting with $J^0 := \emptyset$ and adding one new level $j_i \in I(x_n)$ in each step $i \geq 1$ in such a way that

$$\left|\bigsqcap_{j \in J^{i-1}} \mathrm{SP}_j(x_n) \sqcap \mathrm{SP}_{j_i}(x_n)\right| > \left|\bigsqcap_{j \in J^{i-1}} \mathrm{SP}_j(x_n)\right|.$$

Let s be the number of construction steps needed, i.e. $J := J^s$ is such that property (1) of the lemma holds for this subset of $I(x_n)$. By definition of S_n, it is clear that such a subset exists because $I(x_n)$ itself satisfies property (1).
For each construction step i, we let

$$\Gamma_i := \{\overline{\sigma} \in \bigtimes_{j \in J^i} \mathrm{Sym}(\mathrm{SP}_j(x_n)) \mid \text{ there is a } \pi \in \mathrm{Sym}_n \text{ that realises } \overline{\sigma}\}.$$

Furthermore, for each step i we let k_i be the increase in the number of parts in the intersection that is achieved in this step:

$$k_i := |\bigsqcap_{j \in J^i} \mathrm{SP}_j(x_n)| - |\bigsqcap_{j \in J^{i-1}} \mathrm{SP}_j(x_n)|.$$

The main part of the proof consists in showing the following

▷ Claim 31. For each step i, the size of $|\Gamma_i|$ is bounded by

$$|\Gamma_i| \leq \prod_{j=1}^{i} (\min\{(k_j+1), f(n)\})!$$

Proof. Via induction on i. For $i = 1$, we have $k_1 = |\mathrm{SP}_{j_1}(x_n)| \leq f(n)$, where j_1 is the level chosen in the first step of the construction of J. The group Γ_1 is a subgroup of $\mathrm{Sym}(\mathrm{SP}_{j_1}(x_n))$, whose size is bounded by $|\mathrm{SP}_{j_1}(x_n)|!$. Therefore, the claim holds. For the inductive step, consider the step $i+1$ of the construction. Let j_{i+1} be the level that is added in this step. In order to bound the size of Γ_{i+1}, we consider for each $\overline{\sigma} \in \Gamma_i$ the following set:

$$\Gamma^{\overline{\sigma}}_{i+1} := \{\sigma \in \mathrm{Sym}(\mathrm{SP}_{j_{i+1}}(x_n)) \mid \text{there is a } \pi \in \mathrm{Sym}_n \text{ that realises } \sigma \text{ and } \overline{\sigma}\}.$$

We need to show that for each $\overline{\sigma} \in \Gamma_i$, it holds $|\Gamma^{\overline{\sigma}}_{i+1}| \leq (\min\{(k_{i+1}+1), f(n)\})!$.
Since $|\mathrm{SP}_{j_{i+1}}(x_n)| \leq f(n)$, the bound $|\Gamma^{\overline{\sigma}}_{i+1}| \leq f(n)!$ is clear. It remains to show that for an arbitrary fixed $\overline{\sigma} \in \Gamma_i$, it holds $|\Gamma^{\overline{\sigma}}_{i+1}| \leq (k_{i+1}+1)!$.
For a part $P \in \mathrm{SP}_{j_{i+1}}(x_n)$, let

$$\mathbf{Q}(P) := \{Q \in \bigsqcap_{j \in J^i} \mathrm{SP}_j(x_n) \mid Q \cap P \neq \emptyset\}.$$

We define an equivalence relation $\sim \subseteq (\mathrm{SP}_{j_{i+1}}(x_n))^2$: For parts $P, P' \in \mathrm{SP}_{j_{i+1}}(x_n)$ we let

$$P \sim P' \text{ iff } \mathbf{Q}(P) = \mathbf{Q}(P').$$

The images of each part $\mathrm{SP}_{j_{i+1}}(x_n)$ under permutations in $\Gamma^{\overline{\sigma}}_{i+1}$ are contained in a single equivalence class of $\sim$: Every $\pi \in \mathrm{Sym}_n$ that realises any $\sigma \in \Gamma^{\overline{\sigma}}_{i+1}$ also realises $\overline{\sigma} \in \Gamma_i$. Hence, by Lemma 16, all such π induce the same $\vartheta_{\overline{\sigma}} \in \mathrm{Sym}(\bigsqcap_{j \in J^i} \mathrm{SP}_j(x_n))$. This means that for any $\sigma \in \Gamma^{\overline{\sigma}}_{i+1}$, and every part $P \in \mathrm{SP}_{j_{i+1}}(x_n)$, $Q \in \bigsqcap_{j \in J^i} \mathrm{SP}_j(x_n)$,

$$\sigma(P) \cap \vartheta_{\overline{\sigma}}(Q) \neq \emptyset \text{ iff } Q \in \mathbf{Q}(P).$$

Therefore, all possible images $\sigma(P) \in \mathrm{SP}_{j_{i+1}}(x_n)$, for all $\sigma \in \Gamma^{\overline{\sigma}}_{i+1}$ must be in the same equivalence class of $\sim$. Consequently, we can bound $|\Gamma^{\overline{\sigma}}_{i+1}|$ as follows: Let m be the number of equivalence classes of $\sim$ and let $\ell_1, .., \ell_m$ denote the sizes of the respective classes. Then from our observations so far it follows:

$$|\Gamma^{\overline{\sigma}}_{i+1}| \leq \prod_{t \in [m]} \ell_t! \qquad (\star)$$

Next, we establish a relationship between the properties of $\sim$ and the number k_{i+1}:

$$\begin{aligned}
k_{i+1} &= \Big|\mathrm{SP}_{j_{i+1}}(x_n) \sqcap \prod_{j\in J^i} \mathrm{SP}_j(x_n)\Big| - \Big|\prod_{j\in J^i} \mathrm{SP}_j(x_n)\Big| \\
&= \sum_{Q\in\prod_{j\in J^i}\mathrm{SP}_j(x_n)} (|\{P \in \mathrm{SP}_{j_{i+1}}(x_n) \mid Q \in \mathbf{Q}(P)\}| - 1) \\
&\geq \sum_{\substack{[P]_\sim,\\ P\in \mathrm{SP}_{j_{i+1}}(x_n)}} (|[P]_\sim| - 1) \\
&= |\mathrm{SP}_{j_{i+1}}(x_n)| - m
\end{aligned}$$

The first equality is due to the fact that each part $Q \in \prod_{j\in J^i}$ is split into as many parts as there are parts in $\mathrm{SP}_{j_{i+1}}(x_n)$ intersecting Q.

To see why the inequality holds, fix a choice function g that maps each equivalence class $[P]_\sim$ to a part $Q \in \mathbf{Q}(P)$. By definition of $\sim$, we have $g([P]_\sim) \in \mathbf{Q}(P')$ for every $P' \in [P]_\sim$. Hence, for every $Q \in \prod_{j\in J^i} \mathrm{SP}_j(x_n)$, it holds:

$|\{P \in \mathrm{SP}_{j_{i+1}}(x_n) \mid Q \in \mathbf{Q}(P)\}| - 1 \geq \sum_{[P]_\sim\in g^{-1}(Q)}(|[P]_\sim| - 1)$.

We can sum up the result of these considerations like this:

$$m \geq |\mathrm{SP}_{j_{i+1}}(x_n)| - k_{i+1}. \tag{$\star\star$}$$

Let us now finish the proof of the claim:

We have already established the upper bound ($\star$) for $|\Gamma^{\overline{\sigma}}_{i+1}|$. Let $p \in [m]$ be such that $\ell_p \geq \ell_t$ for all $t \in [m]$. A consequence of ($\star\star$) is: $\ell_p \leq k_{i+1} + 1$. It can be checked that the values $\ell_1, ...\ell_m$ that maximise the bound in ($\star$) and satisfy ($\star\star$) are such that $\ell_t = 1$ for all $t \neq p$. Therefore, ($\star$) becomes:

$$|\Gamma^{\overline{\sigma}}_{i+1}| \leq \ell_p! \leq (k_{i+1} + 1)!$$

This concludes the proof of the claim. ◁

Hence, in order to finish the proof of the lemma, we have to bound

$$|\Gamma_s| \leq \prod_{i=1}^{s} (\min\{(k_i + 1), f(n)\})!$$

from above (recall that s is the number of steps needed to construct J satisfying property (1)). We know that $\sum_{i=1}^{s} k_i$ is some fixed value $\leq n$. The value of the above product and the sum solely depends on the sequence $(k_i)_{i\in[s]}$. One can see by a "redistribute-weight argument" that the value of the product is maximised for a sequence $(k_i)_{i\in[s]}$, where every k_i is either 1 or $k_i = f(n) - 1$ (and there may be exactly one k_i with $1 < k_i < f(n) - 1$). For such a sequence of k_is, the value of the product is at most

$$|\Gamma_s| \leq \prod_{i=1}^{s} (\min\{(k_i + 1), f(n)\})! \leq f(n)!^{n/(f(n)-1)} \cdot 2^n$$ ◀

8.2 Proof of Lemma 27

For the proof of Lemma 27, we make use of the following small observation.

▶ **Lemma 32.** *Let $B \subseteq \{0, 1\}^n$. The partition $\prod B = \prod_{b\in B} SP(b)$ is a supporting partition for B.*

Proof. By the definition of the intersection, every string $b \in B_n$ is constant on every part $P \in \bigsqcap B$. Hence, $\mathrm{Stab}_n^\bullet(\bigsqcap B) \subseteq \mathrm{Stab}_n(B_n)$. This is the definition of a supporting partition. ◀

▶ **Lemma 27.** *There exists a subset $A \subseteq B_n$ of size $|A| \leq \frac{|S_n|}{2}$ such that for each part $P \in \bigsqcap A$, one of the following two statements is true:*

1. $|P \cap S_n| \leq 2$*; **or:***
2. $|P \cap S_n| > 2$ *and for every $b \in B_n \setminus A$, one of these two conditions holds:*
   - *b is constant on $P \cap S_n$; **or***
   - *$b[P \cap S_n]$ is* imbalanced *and, for every $P' \in \bigsqcap A$ with $P' \neq P$, $|P' \cap S_n| > 2$, b is constant on $P' \cap S_n$.*

By $b[P \cap S_n]$ we mean the substring of b at the positions in $P \cap S_n$, and being imbalanced *means that $b[P \cap S_n]$ contains exactly one 0 and there is a 1 at all other positions, or vice versa (exactly one 1 and the rest 0).*

Proof. We construct A stepwise, starting with $A^0 := \emptyset$, and adding one string $a_i \in B_n$ in step i. For step $i+1$ of the construction, assume we have constructed A^i. For $k \in [n]$, we write $P_i(k)$ for the part of $\bigsqcap A^i = \bigsqcap_{a \in A^i} \mathrm{SP}(a)$ that k is in. Now we let

$$K_i := \{k \in S_n \mid |P_i(k) \cap S_n| > 2\}.$$

This is the set of positions whose parts need to be refined more. If $K_i = \emptyset$, then the construction is finished because all parts of $\bigsqcap A^i$ satisfy condition 1 of the lemma. So assume $K_i \neq \emptyset$. By Lemma 32, $\bigsqcap B_n$ is a supporting partition for B_n and therefore at most as coarse as $\mathrm{SP}(B_n)$. Hence, all positions in S_n are in singleton parts of $\bigsqcap B_n$.
We conclude that for all $k \in K_i$, there must be a string $b \in B_n \setminus A^i$ that can be added to A^i in order to make $P_i(k) \cap S_n$ smaller when it is intersected with $\mathrm{SP}(b)$. In fact, there may be several such strings b that we could choose to add in this step of the construction. So let

$$C_k := \{b \in B_n \setminus A^i \mid b \text{ is non-constant on } P_i(k) \cap S_n\}$$

be the non-empty set of such candidate strings. We restrict our candidate set further:

$$\begin{aligned} \widehat{C}_k := \{b \in C_k \mid\ & \text{there are two distinct parts } P, P' \in \bigsqcap A^i \\ & \text{s.t. } b \text{ is non-constant on } P \cap S_n \text{ and } P' \cap S_n, \text{ and} \\ & |P \cap S_n| > 2 \text{ and } |P' \cap S_n| > 2\} \\ \cup \{b \in C_k \mid\ & b[P_i(k) \cap S_n] \text{ is not imbalanced}\}. \end{aligned}$$

We pick our next string a_{i+1} that is added in this step of the construction from one of the sets $\widehat{C}_k$, where k ranges over all positions in K_i. If $\widehat{C}_k = \emptyset$ for all these k, then A^i is already the desired set A because it satisfies the conditions of the lemma.
Otherwise, we choose a_{i+1} arbitrarily from one of the $\widehat{C}_k$ and set $A^{i+1} := A^i \cup \{a_{i+1}\}$. Then we proceed with the construction until $K_i = \emptyset$ or all $\widehat{C}_k$ are empty. In both cases, the constructed set is as required by the lemma.
It remains to show: $|A| \leq \frac{|S_n|}{2}$, i.e. that the construction process consists of at most $\frac{|S_n|}{2}$ steps. We do this by defining a potential function Φ that associates with any partition $\mathcal{P}$ of $[n]$ a natural number $\leq n$ that roughly says how many further refinement steps of $\mathcal{P}$ are at most possible. Concretely:

$$\Phi(\mathcal{P}) := \sum_{P \in \mathcal{P}} \max\{(|P \cap S_n| - 2), 0\}.$$

If $\mathcal{P}$ contains as its only part the whole set $[n]$, then $\Phi(\mathcal{P}) = |S_n| - 2$. Now observe that a necessary condition for adding a new string a_{i+1} to A is the existence of a part P with $|P \cap S_n| > 2$ in the current partition $\mathcal{P} = \bigsqcap A^i$. This is the case if and only if $\Phi(\mathcal{P}) > 0$. Therefore, all that remains to show is:

$$\Phi\left(\bigsqcap A^i\right) - \Phi\left(\bigsqcap A^{i+1}\right) \geq 2 \tag{$\star$}$$

for all construction steps i. Consider step $i + 1$: We add $a_{i+1} \in \widehat{C}_k$ (for some $k \in K_i$) to A^i_n. It can be checked that the definition of $\widehat{C}_k$ ensures that $(\star)$ holds for $\bigsqcap A^i$ and $\bigsqcap A^{i+1} = \bigsqcap A^i \sqcap a_{i+1}$: The new string a_{i+1} either splits two distinct parts, or, if it only splits one part, it splits it into two parts of size at least two. ◀

8.3 Proof of Lemma 28

▶ **Lemma 28.** *Let $A \subseteq B_n$ be the subset that exists by Lemma 27, and let $p : A \longrightarrow B_n$ be an injective function. Let*

$$\Gamma_p := \{\pi \in \mathit{Stab}_n(B_n) \mid p(a)^\pi = a \text{ for all } a \in A\}.$$

Further, let

$$P_{>2} := \{k \in S_n \mid |P(k) \cap S_n| > 2, \text{ where } P(k) \in \bigsqcap A \text{ is the part that } k \text{ is in}\}.$$

Then for any $\pi, \pi' \in \Gamma_p$ such that $\pi^{-1}|_{([n]\setminus P_{>2})} = \pi'^{-1}|_{([n]\setminus P_{>2})}$, it also holds $\pi^{-1}|_{P_{>2}} = \pi'^{-1}|_{P_{>2}}$.

Proof. For a contradiction, we assume that there exist $\pi, \pi' \in \Gamma_p$ such that $\pi^{-1}|_{([n]\setminus P_{>2})} = \pi'^{-1}|_{([n]\setminus P_{>2})}$, but $\pi^{-1}|_{P_{>2}} \neq \pi'^{-1}|_{P_{>2}}$. Then there is $x \in [n]$ such that $\pi(x) \in P_{>2}$, and $\pi'(x) \neq \pi(x)$ (i.e. $\pi(x)$ is the point where π^{-1} and π'^{-1} differ). Let $y := \pi(x), y' := \pi'(x)$. Let $P(y) \in \bigsqcap A$ be the part that y is in, and let $\widehat{P}(y) := P(y) \cap S_n$. We know that $y' \in P(y)$, too, because $\pi, \pi' \in \Gamma_p$, so this follows from Lemma 25. As $y \in P_{>2}$, in particular, $y \in S_n$. Hence, also $x, y' \in S_n$ because $\pi, \pi' \in \mathrm{Stab}_n(B_n)$, and by Lemma 11, singleton parts must be mapped to singleton parts. We conclude that we even have $y' \in \widehat{P}(y)$.

Now, our goal is to show that the transposition $\tau := (y\ y')$ is contained in $\mathrm{Stab}_n(B_n)$. This is a contradiction because in $\mathrm{SP}(B_n)$, y, y' are both in singleton parts, but if $\tau \in \mathrm{Stab}_n(B_n)$, then the coarsest supporting partition $\mathrm{SP}(B_n)$ can be coarsened to another supporting partition by letting $\{y, y'\}$ be one single part.

In order to show $\tau \in \mathrm{Stab}(B_n)$, we only need to deal with those strings in B_n which are not constant on the positions $\{y, y'\}$. More precisely, we have to show that every $b \in B_n$ with $b_y \neq b_{y'}$ has a "swapping partner" $b' \in B_n$ where $b'_y = b_{y'}$ and vice versa, and $b'_i = b_i$ for all other i.

So take any $b \in B_n$ such that w.l.o.g. $b_y = 0, b_{y'} = 1$. Note that $b \notin A$, as every string in A is constant on $P(y)$ (otherwise, $P(y)$ would not be a single part in $\bigsqcap A$). Furthermore, $|\widehat{P}(y)| > 2$, since $y \in P_{>2}$. Therefore, Lemma 27 implies that the substring $b[\widehat{P}(y)]$ is imbalanced and b is constant on every $P' \cap S_n$, for all $P' \in \bigsqcap A$ with $P' \neq P(y)$, $|P' \cap S_n| > 2$. W.l.o.g. let the imbalance of $b[\widehat{P}(y)]$ be such that $b_i = 1$ for every position $i \in \widehat{P}(y), i \neq y$. We claim that $b' := b^{\pi' \circ \pi^{-1}}$ is the desired swapping partner of b, i.e. $b^\tau = b'$ and vice versa. Note that $b' \in B_n$ because π, π' stabilise the set B_n.

To see that $b' = b^\tau$, consider firstly $b^{\pi^{-1}} \in B_n$. Obviously, $(b^{\pi^{-1}})_x = 0$. Hence, $(b^{\pi' \circ \pi^{-1}})_{y'} = 0$. Moreover, the substring $b^{\pi^{-1}}[\pi^{-1}(\widehat{P}(y))]$ is imbalanced just like $b[\widehat{P}(y)]$, so $(b^{\pi^{-1}})_j = 1$ for all $j \in \pi^{-1}(\widehat{P}(y)) \setminus \{x\}$. As a consequence of Corollary 26, we have $\pi^{-1}(\widehat{P}(y)) = \pi'^{-1}(\widehat{P}(y))$,

so $(\pi' \circ \pi^{-1})(\widehat{P}(y)) = \widehat{P}(y)$. Therefore, the substring $b^{\pi' \circ \pi^{-1}}[\widehat{P}(y)]$ is also imbalanced and has a 1 at each position except y'.
This shows that $(b^\tau)[\widehat{P}(y)] = b'[\widehat{P}(y)]$. It remains to show that $b_i = b'_i$ for all $i \in [n] \setminus \widehat{P}(y)$.
We have $(\pi' \circ \pi^{-1})(i) = i$ for $i \in [n] \setminus P_{>2}$, because $\pi^{-1}|_{([n] \setminus P_{>2})} = \pi'^{-1}|_{([n] \setminus P_{>2})}$, so $b_i = b'_i$ for $i \in [n] \setminus P_{>2}$.
For $i \in P_{>2} \setminus \widehat{P}(y)$, let $\widehat{P}(i)$ be the part of $\bigsqcap A$ that i is in, intersected with S_n. As already said, we know from Lemma 27 that b is constant on $\widehat{P}(i)$. Analogously to what we argued already for $\widehat{P}(y)$, we get that $(\pi' \circ \pi^{-1})(\widehat{P}(i)) = \widehat{P}(i)$, so also for $i \in P_{>2} \setminus \widehat{P}(y)$, we have $b_i = b'_i$.
In total, this shows that indeed, $b' = b^\tau$, and since $b' \in B_n$, we have $\tau \in \mathrm{Stab}_n(B_n)$. This is a contradiction and finishes the proof of the lemma. ◀

Typable Fragments of Polynomial Automatic Amortized Resource Analysis

Long Pham
Carnegie Mellon University, Pittsburgh, PA, USA
longp@andrew.cmu.edu

Jan Hoffmann
Carnegie Mellon University, Pittsburgh, PA, USA
janh@andrew.cmu.edu

Abstract

Being a fully automated technique for resource analysis, automatic amortized resource analysis (AARA) can fail in returning worst-case cost bounds of programs, fundamentally due to the undecidability of resource analysis. For programmers who are unfamiliar with the technical details of AARA, it is difficult to predict whether a program can be successfully analyzed in AARA. Motivated by this problem, this article identifies classes of programs that can be analyzed in type-based polynomial AARA. Firstly, it is shown that the set of functions that are typable in univariate polynomial AARA coincides with the complexity class PTIME. Secondly, the article presents a sufficient condition for typability that axiomatically requires every sub-expression of a given program to be polynomial-time. It is proved that this condition implies typability in multivariate polynomial AARA under some syntactic restrictions.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Resource consumption, Quantitative analysis, Amortized analysis, Typability

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.34

Related Version A full version of the paper is available at `https://arxiv.org/abs/2010.16353` [30].

Funding This article is based on research supported by DARPA under AA Contract FA8750-18-C-0092 and by the National Science Foundation under SaTC Award 1801369, CAREER Award 1845514, and SHF Awards 1812876 and 2007784. Any opinions, findings, and conclusions contained in this document are those of the authors and do not necessarily reflect the views of the sponsoring organizations. Long Pham gratefully acknowledges the support of the Funai Overseas Scholarship by the Funai Foundation of Information Technology, Japan.

1 Introduction

There exists a wide range of effective techniques for automatically or semi-automatically analyzing the resource consumption of programs. These techniques derive symbolic bounds on the worst-case [24], best-case [10, 28], or expected [7, 29] resource consumption and are based on type systems [8, 33, 9, 26, 2, 6, 12], recurrence relations [34, 11, 1, 25, 23], relational reasoning [6, 31], and term rewriting [3, 5, 19].

State-of-the-art resource analyses can automatically derive complex bounds for large programs, and making analyses more practical by improving their efficiency and range is a main driving force in this area. However, resource analysis for Turing-complete languages is undecidable, and even for the most sophisticated tools there will remain programs that cannot be analyzed automatically. Diagnosing the cause and modifying the program so that the analysis can derive a bound often require in-depth knowledge of the implemented techniques. As a result, the usability of more sophisticated analysis tools is hampered by their complexity.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 34; pp. 34:1–34:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

To improve the usability of automatic resource analysis for non-experts, this article develops easy-to-understand characterizations of the programs that can be analyzed with automatic amortized resource analysis (AARA). Such characterizations can serve as explanations for an unsuccessful resource analysis and guide program development without revealing technical details of the underlying analysis.

AARA is a type-based analysis that is based on the potential method of amortized analysis. It has been first introduced by Hofmann and Jost [18] for deriving linear heap-space bounds for a first-order language with lists. AARA has subsequently been extended to univariate polynomial bounds [16], multivariate polynomial bounds [13, 14], and exponential bounds [22]. Furthermore, AARA has been extended to other language features such as higher-order and polymorphic functions [20, 15], lazy evaluation [21], and probabilistic programming [29]. The analysis has been implemented in the programming language Resource-Aware ML (RaML) [15]. An overview of polynomial AARA can be found in Section 2. We are not aware of previous work that studies the characterization of typable fragments of AARA.

Our first contribution (Section 3) is a characterization of the (mathematical) functions that can be implemented in AARA. We demonstrate that it is possible to embed every polynomial-time Turing machine in AARA. That is, for every such Turing machine, there exists an equivalent polynomial-time program that is typable in polynomial AARA. This result shows that polynomial AARA corresponds to the complexity class PTime and is in the tradition of implicit computational complexity (ICC) [4, 27, 17], which studies linguistic characterizations of complexity classes. For a user of RaML, this result means that an implementation of a PTime function can always be rewritten so that a worst-case cost bound can be automatically derived. However, it does not provide guidance on how to rewrite an implementation.

An ideal resource analysis should automatically derive a cost bound for every program that has a polynomial bound. However, such an analysis does not exist, because the problem of deciding whether a given program runs in polynomial time is undecidable [13]. Moreover, AARA is a type-based analysis that derives the bound of an expression from its sub-expressions. So we can only expect to derive a bound for an expression which is *inherently polynomial time*, that is, every subexpression is in PTime if viewed as a function.

Our second contribution is an axiomatic definition of inherently polynomial time that implies typability in multivariate polynomial AARA for a Turing-complete first-order language with lists (Section 2) under some restrictions: Programs can only use primitive recursion instead of general recursion, some variables are affine, and the use of nested lists is restricted. Although this characterization is far from being a necessary condition, we believe that it can be a valuable guide to users. A key concept is the notion of *uniform resource annotations* which is essential in the proof that inherently polynomial time is a sufficient condition for typability in multivariate polynomial AARA.

2 Automatic Amortized Resource Analysis (AARA)

Among approaches to resource analysis is AARA. Given a program P, consider its history of execution, that is, a sequence of transitioning program states. As in Sleator and Tarjan's potential method in amortized analysis [32], we assign a certain (non-negative) amount of potential to the initial state of this sequence. If we can ensure that (i) the amount of potential never becomes negative throughout P's run and (ii) the actual computational cost in each transition of P is less than or equal to the change in the amount of potential, then we know that the total resource usage of P is bounded above by the initial potential. This is essentially how AARA works.

More concretely, each sub-expression of P is assigned a *resource-annotated type*: a conventional (i.e. simple) type augmented with an expression that indicates how much potential is stored. In polynomial AARA [16, 14], we use polynomial functions to express potential. Initially, AARA only assigns templates of resource-annotated types where coefficients of polynomials are left blank. AARA then collects constraints on these coefficients that respect the cost semantics of P. Finally, as these constraints are all linear, we can simply solve them using an off-the-shelf liner program solver, thereby inferring resource-annotated types. A worst-case cost bound of P can be extracted from its resource-annotated type.

2.1 Resource-Aware ML

Resource-Aware ML (RaML) is a Turing-complete functional programming language used in the study of AARA [16].

The original version of RaML is first-order (i.e. no higher-order types or functions appear in RaML) and only offers a relatively small set of language features. Subsequent versions of RaML support more language features such as higher-order functions and polymorphic functions [15]. In this section, we describe a variant of RaML that only differs from the original version in a few minor details; e.g. the tick construct and the support for sum types.

The base types (denoted by b) and simple types (denoted by τ) of RaML are formed by

$$\begin{array}{rll@{\qquad}rll}
b ::= & \mathbf{1} & \text{unit type} & \tau ::= & b & \text{base type} \\
 & b_1 + b_2 & \text{sum type} & & b_1 \to b_2 & \text{arrow type} \\
 & b_1 \times b_2 & \text{product type} & & & \\
 & L(b) & \text{list type.} & & &
\end{array}$$

The set of all base types will be denoted by $\mathbb{B}$.

Fix a set $\mathcal{V} = \{x, y, x_1, x_2, \dots\}$ of variable symbols and a set $\mathcal{F} = \{f, \dots\}$ of function symbols. The grammar of RaML is

$$\begin{array}{rll}
e ::= & x & \text{variable} \\
 & \mid \langle\,\rangle & \text{unit element} \\
 & \mid \ell \cdot x \mid r \cdot x \mid \mathsf{case}\ x\ \{\ell \cdot y \hookrightarrow e_\ell \mid r \cdot y \hookrightarrow e_r\} & \text{sum constructors and destructor} \\
 & \mid \langle x_1, x_2\rangle \mid \mathsf{case}\ x\ \{\langle x_1, x_2\rangle \hookrightarrow e\} & \text{pair constructor and destructor} \\
 & \mid [\,] \mid x_1 :: x_2 \mid \mathsf{case}\ x\ \{[\,] \hookrightarrow e_0 \mid (x_1 :: x_2) \hookrightarrow e_1\} & \text{list constructors and destructor} \\
 & \mid \mathsf{fun}\ f\ x = e & \text{function definition} \\
 & \mid f\ x & \text{function application} \\
 & \mid \mathsf{tick}\ q & \text{resource consumption; } q \in \mathbb{Q} \\
 & \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 & \text{let-binding} \\
 & \mid \mathsf{share}\ x\ \mathsf{as}\ x_1, x_2\ \mathsf{in}\ e & \text{variable sharing.}
\end{array}$$

In a function definition, e is allowed to mention f. Therefore, we can implement not only primitive recursion but also general recursion. As standard, we use the let-normal form, where we only permit function application of the form $x_1\ x_2$ as opposed to $e_1\ e_2$. For convenience in resource analysis, we require each variable symbol to be used in a affine manner (i.e. can only be used at most once). To use a variable symbol multiple times, we duplicate the symbol with the share construct.

In the interest space, we will not present a type system of this language here. It is available in Appendix B.1 of the full version of this article [30].

RaML programs are evaluated using the call-by-value strategy. Computational costs accrue only when tick q is executed, and this cost metric is known as the tick metric. The general cost semantics of RaML can be found in [16]. In the case of the running time, which is a specific cost metric, of RaML, the judgment of the cost semantics has the form

$$V \vdash e \Downarrow v \mid n,$$

where V is an environment (i.e. a set of pairs of variable symbols and semantic values), v is a semantic value, and $n \in \mathbb{N}$ is the running time of evaluating program e to v. The running time is formally defined in Appendix B.2 of [30].

2.2 Univariate AARA

In univariate AARA, each list is annotated with a polynomial indicating the amount of the potential stored in the list. Univariate AARA does not let us mix potential of two lists, that is, multiply polynomials of two lists' potential. This is why univariate AARA is called *univariate.*

Resource-annotated base types (denoted by b) and resource-annotated simple types (denoted by τ) are formed by the following grammar:

$$\begin{array}{lll@{\qquad}lll}
b ::= & \mathbf{1} & \text{unit type} & B ::= & \langle b, q \rangle & q \in \mathbb{Q}_{\geq 0} \\
 & b_1 + b_2 & \text{sum type} & \tau ::= & b & \text{base type} \\
 & b_1 \times b_2 & \text{product type} & & B_1 \to B_2 & \text{arrow type} \\
 & L^{\vec{q}}(b) & \text{list type.} & & &
\end{array}$$

Here, $\vec{q}$ is a finite vector of $\mathbb{Q}_{\geq 0}$.

Given a semantic value $v : b$, where b is a resource-annotated base type, the potential stored in v is inductively defined as

$$\begin{array}{rl}
\Phi(v : \mathbf{1}) := 0 & \Phi([\,] : L^{\vec{q}}(b)) := 0 \\
\Phi(\ell \cdot v : b_1 + b_2) := \Phi(v : b_1) & \Phi(v_1 :: v_2 : L^{\vec{q}}(b)) := \Phi(v_1 : b) + \phi(|v_1 :: v_2|, \vec{q}) \\
\Phi(r \cdot v : b_1 + b_2) := \Phi(v : b_2), &
\end{array}$$

where $|\cdot|$ denotes the length of an input list. Given $n \in \mathbb{N}$ and $\vec{q} = (q_1, \ldots, q_k)$, $\phi(n, \vec{q})$ is defined as $\phi(n, \vec{q}) := \sum_{i=1}^{k} q_i \binom{n}{i}$. If $n < i$, then $\binom{n}{i} = 0$.

The typing judgment of univariate AARA has the form

$$\Gamma_{\text{anno}}; p \vdash e : B,$$

where Γ_{anno} is a resource-annotated typing context and $p \in \mathbb{Q}_{\geq 0}$. We sometimes write $\Sigma_{\text{anno}}; \Gamma_{\text{anno}}; p \vdash e : B$, where a resource-annotated typing context is split into Σ_{anno} for arrow-type variables and Γ_{anno} for base-type variables. The type system of univariate AARA is available in Appendix C.1 of [30].

To give examples of judgments in univariate AARA, consider two programs: (i) *append* that appends the first input list to the second, and (ii) *quicksort* that performs quicksort. The running time of *append* is proportional to the size of the first input, and the running time of *quicksort* is bounded by the square of the input size. For simplicity, we will not work out the exact coefficients of polynomial bounds. Instead, we simply assume that the running time of *append* is bounded by the function $n, m \mapsto n$, where n and m are the lengths of the two input lists. Likewise, we assume that the running time of *quicksort* is bounded by

$n \mapsto n^2$, respectively. It then makes sense that these two programs can be typed in univariate AARA as

$$\textit{append}: \langle\langle L^1(b), L^0(b)\rangle, 0\rangle \to \langle L^0(b), 0\rangle \qquad \textit{quicksort}: \langle L^{(1,2)}(b), 0\rangle \to \langle L^0(b), 0\rangle.$$

The univariate resource annotation $(1,2)$ of *quicksort* represents polynomial $n \mapsto 1 \cdot \binom{n}{1} + 2 \cdot \binom{n}{2} = n^2$. The implementations of *append* and *quicksort* are given in Appendix C.3 of [30].

Univariate AARA is sound with respect to the cost semantics (specifically, the running time) of RaML:

▶ **Theorem 1** (Soundness of univariate AARA [16]). *Given term e, suppose $\Gamma_{anno}; p \vdash e : \langle b_{anno}, q\rangle$ is derived in univariate AARA. Let V be an environment such that $V \vdash e \Downarrow v \mid n$; that is, e runs in n units of time under V. We then have*

$$n \le p + \Phi(V : \Gamma_{anno}) - q - \Phi(v : b_{anno}),$$

where $\Phi(V : \Gamma_{anno}) = \sum_{x \in dom(\Gamma_{anno})} \Phi(V(x) : \Gamma_{anno}(x))$.

2.3 Multivariate AARA

In contrast to univariate AARA, multivariate AARA allows us to mix potential of different lists. For example, we can have $|\ell_1| \cdot |\ell_2|$'s worth of potential, where $|\cdot|$ denotes the length of a list, in multivariate AARA. Due to this multivariate nature, multivariate AARA has a single global resource annotation represented by a multivariate polynomial over all size variables occurring in a given term. This global resource annotation is separate from individual types in a typing context.

Multivariate AARA is strictly more expressive than univariate one. This is surprising in light of the fact that multivariate polynomials can always be bounded by univariate polynomials; e.g. xy is bounded by $x^2 + y^2$. Examples of programs that cannot be typed in univariate AARA but are typable in multivariate AARA are in Section 4.1 and Section 5.

Resource-Annotated Types

Resource-annotated types in multivariate AARA are formed by

$$\begin{array}{llllll} b ::= & \mathbf{1} & \text{unit type} & B ::= & \langle b, Q\rangle & \\ & b_1 + b_2 & \text{sum type} & \tau ::= & b & \text{base type} \\ & b_1 \times b_2 & \text{product type} & & B_1 \to B_2 & \text{arrow type} \\ & L(b) & \text{list type.} & & & \end{array}$$

In $\langle b, Q\rangle$, Q is a multivariate resource annotation over the size variables inside b. This will be formalized shortly.

Given a base type $b \in \mathbb{B}$, its base polynomial is a function of type $[\![b]\!] \to \mathbb{N}$, where $[\![b]\!]$ is the set of semantic values of type b. The set of base polynomials associated with type b, denoted by $\mathcal{B}(b)$, is inductively defined as follow:

$$\begin{aligned} \mathcal{B}(\mathbf{1}) &:= \{\lambda v.1\} \\ \mathcal{B}(b_1 + b_2) &:= \{\lambda(\ell \cdot v).p(v) \mid p \in \mathcal{B}(b_1)\} \cup \{\lambda(r \cdot v).p(v) \mid p \in \mathcal{B}(b_2)\} \\ \mathcal{B}(b_1 \times b_2) &:= \{\lambda\langle v_1, v_2\rangle.p_1(v_1) \cdot p_2(v_2) \mid p_i \in \mathcal{B}(b_i)\} \\ \mathcal{B}(L(b)) &:= \{\lambda[v_1, \dots, v_n]. \sum_{1 \le j_1 < \cdots < j_k \le n} \prod_{1 \le i \le k} p_i(v_{j_i}) \mid k \in \mathbb{N}, p_i \in \mathcal{B}(b)\}. \end{aligned}$$

For $b_1 + b_2$, we have a set of base polynomials for the ℓ-tag and another set for the r-tag. If a base polynomial is applied to a value with a wrong tag, we assume that the output is 0. For instance, if we feed a value $\ell \cdot \langle\,\rangle$ to $\lambda(r \cdot v).1$, the output should be 0. In the definition of $\mathcal{B}(L(b))$, if $n < k$, the function should return 0 since it is the identity of summation.

Given base type b, a resource polynomial $p : [\![b]\!] \to \mathbb{Q}_{\geq 0}$ is a non-negative linear combination of finitely many base polynomials from $\mathcal{B}(b)$. It is straightforward to prove that $\mathcal{B}(b)$ for any b contains $\lambda v.1$. Therefore, a resource polynomial is always capable of expressing constant potential.

For convenience, it is desirable to have a succinct notation for base polynomials. This is achieved by introducing indexes of base polynomials:

$$\begin{aligned}
\mathcal{I}(\mathbf{1}) &:= \{*\} \\
\mathcal{I}(b_1 + b_2) &:= \{\ell \cdot i \mid i \in \mathcal{I}(b_1)\} \cup \{r \cdot i \mid i \in \mathcal{I}(b_2)\} \\
\mathcal{I}(b_1 \times b_2) &:= \{\langle i_1, i_2\rangle \mid i_1 \in \mathcal{I}(b_1), i_2 \in \mathcal{I}(b_2)\} \\
\mathcal{I}(L(b)) &:= \{[i_1, \ldots, i_k] \mid k \in \mathbb{N}, i_j \in \mathcal{I}(b)\}.
\end{aligned}$$

An index is usually used as a subscript for a (meta)-variable representing a coefficient of a base polynomial. For instance, $q_{\langle *,* \rangle} \in \mathbb{Q}_{\geq 0}$ is a meta-variable representing a coefficient of base polynomial $\lambda\langle v_1, v_2\rangle.1$. For any base type b, we will write 0_b for the index $\lambda v.1$.

For example, consider $\mathcal{I}(L(\mathbf{1})) = \{*, [*], [*, *], [*, *, *], \ldots\}$. The index $[*, *]$ represents the polynomial function

$$\begin{aligned}
\lambda[v_1, \ldots, v_n]. \sum_{1 \leq j_1 < j_2 \leq n} \prod_{1 \leq i \leq 2} ((\lambda v.1)\ v_{j_i}) &= \lambda[v_1, \ldots, v_n]. \sum_{1 \leq j_1 < j_2 \leq n} 1 \\
&= \lambda[v_1, \ldots, v_n]. \binom{n}{2}.
\end{aligned}$$

Thus, the multivariate index $[*, *]$ represents a quadratic function on the input list's length.

The degree of an index is defined by

$$\mathsf{deg}(*) := 0 \qquad \mathsf{deg}(\langle i_1, i_2\rangle) := \mathsf{deg}(i_1) + \mathsf{deg}(i_2)$$

$$\mathsf{deg}(\ell \cdot i), \mathsf{deg}(r \cdot i) := \mathsf{deg}(i) \qquad \mathsf{deg}([i_1, \ldots, i_k]) := k + \sum_{1 \leq j \leq k} \mathsf{deg}(i_j).$$

Intuitively, $\mathsf{deg}(i)$ is equal to the degree of the polynomial function that index i represents. Because a resource polynomial can only have non-zero coefficients for finitely many base polynomials, any resource polynomial (or a finite set of resource polynomials) has a bounded degree. In practice, we ask a user of AARA to supply an upper bound on the degree of base polynomials.

Resource Annotations of Typing Contexts

Given a base-type typing context $\Gamma = \{x_1 : b_1, \ldots, x_n : b_n\}$, its multivariate resource annotation is given by a resource polynomial of type $b_1 \times \cdots \times b_n$. In other words, we treat a typing context as one big tuple and assign a single multivariate annotation to this tuple.

With regard to an arrow-type typing context $\Sigma = \{f_1 : b_{1,1} \to b_{1,2}, \ldots, f_m : b_{m,1} \to b_{m,2}\}$, its multivariate resource annotation has the form

$$\Sigma_{\text{anno}} = \{f_1 : B_{1,1} \to B_{1,2}, \ldots, f_m : B_{m,1} \to B_{m,2}\},$$

where each $B_{i,j}$ is a pair $\langle b_{i,j}, Q\rangle$ such that Q is a multivariate resource annotation of $b_{i,j}$.

Typing Judgment

The typing judgment of multivariate AARA takes the form

$$\Gamma; P \vdash e : \langle b, Q \rangle,$$

where Γ and b are free of resource annotations. P and Q are multivariate annotation over Γ and b, respectively. The type system of multivariate AARA is available in Appendix D.2 of [30].

To give examples of judgments in multivariate AARA, consider *append* $\langle \ell_1, \ell_2 \rangle$, which appends ℓ_1 to ℓ_2. Suppose that the output must store $n \mapsto n^2$ much potential, where n is the output's length. It is reasonable that the total potential required for this program is $|\ell_1| + (|\ell_1| + |\ell_2|)^2$, out of which $|\ell_1|$ is used to account for the running time. This can be expressed by the judgment $\ell_1 : L(\mathbf{1}), \ell_2 : L(\mathbf{1}); P \vdash \mathit{append}\ \langle \ell_1, \ell_2 \rangle : \langle L(\mathbf{1}), Q \rangle$, where the positive coefficients of P and Q are

$$\begin{array}{ll} P(\langle [*], * \rangle) = P(\langle [*, *], * \rangle) = P(\langle [*], [*] \rangle) = P(\langle *, [*, *] \rangle) = 2 & \qquad P(\langle *, [*] \rangle) = 1 \\ Q([*, *]) = 2 & \qquad Q([*]) = 1. \end{array}$$

P amounts to $2 \cdot \left(\binom{|\ell_1|}{1} + \binom{|\ell_1|}{2} + \binom{|\ell_1|}{1} \cdot \binom{|\ell_2|}{1} + \binom{|\ell_2|}{2} \right) + 1 \cdot \binom{|\ell_2|}{1}$, which is equal to $|\ell_1| + (|\ell_1| + |\ell_2|)^2$ as desired. Similarly, Q amounts to $2 \cdot \binom{n}{2} + 1 \cdot \binom{n}{1} = n^2$ as desired, where $n = |\ell_1| + |\ell_2|$.

The multivariate equivalent of the soundness theorem (Theorem 1) holds [14].

3 Embedding Polynomial-Time Turing Machines in AARA

In this section, we show that every polynomial-time Turing machine can be expressed as a typable RaML program while preserving the semantics and worst-case cost bounds. More formally, we have

▶ **Theorem 2** (Embedding of polynomial-time Turing machines in RaML). *Let M be a polynomial-time Turing machine that inputs and outputs bit strings from $\{0,1\}^*$. There exists a RaML program $M' : \{0,1\}^* \to \{0,1\}^*$ such that*

- *For every input $w \in \{0,1\}^*$, we have $M(w) = M'(w)$;*
- *The computational cost of M' (according to the tick metric) is larger than or equal to the running time of M;*
- *Univariate AARA can infer a polynomial upper bound of the computational cost of M'.*

Theorem 2 only tells us the existence of a RaML program M' that is typable in univariate AARA and that simulates M faithfully. In our proof of the theorem, we assume that a polynomial bound on the running time of M is known. Thus, if we do not have access to this polynomial bound, we cannot construct M'. In fact, the problem of determining whether a given Turing machine runs in polynomial time or not is undecidable [13].

It is fairly easy to prove that the cost of any program according to the tick metric is asymptotically bounded by its running time. Therefore, in the statement of Theorem 2, we can replace the "tick metric" with the "running time" of RaML.

A detailed proof of Theorem 2 is available in Appendix A of [30].

3.1 Preliminaries

▶ **Definition 3** (Turing machine). *A (deterministic) Turing machine M is specified by an 8-tuple $(Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_{\mathit{final}})$, where*

- *Q is a finite set of machine states.*

- *Σ is a finite input alphabet. Γ is a finite alphabet for symbols written on M's tape. Since an input will be initially placed on the tape, every input symbol is also a tape symbol.*
- *$\vdash \in \Gamma \setminus \Sigma$ is the left end marker that demarcates the left end of a semi-infinite working tape, and $\sqcup \in \Gamma \setminus \Sigma$ is the blank symbol for the tape.*
- *$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.*
- *$q_0 \in Q$ is the initial state, and $q_{final} \in Q$ is the final state.*

In the initial configuration of a Turing machine, an input string w is placed immediately after the left end marker $\vdash$ on the tape. The state of the machine is initially q_0, and the read/write head is positioned on the first symbol of w. The rest of the tape is filled with $\sqcup$.

The Turing machine first (i) reads the content of the cell currently under the tape head and (ii) identifies the current state of the machine. The machine then overwrites the current cell (if necessary), updates the machine's state, and moves the head to the left or right according to the transition function δ. The machine terminates as soon as it enters q_{final}. Upon termination, the content of the tape before the first blank symbol is considered as the machine's output. The running time is defined as the number of steps the Turing machine makes before termination.

Without loss of generality, we will henceforth only consider Turing machines with $\Sigma = \{0, 1\}$ and $\Gamma = \Sigma \cup \{\vdash, \sqcup\}$.

To enhance clarity, we will introduce two type aliases, State and Sym, which are defined as $L(\mathbf{1} + \mathbf{1})$; i.e. bit strings or natural numbers. The type State represents machine states of M, and Sym represents tape symbols of M. In fact, because M has finitely many machine states and tape symbols, State and Sym can alternatively be encoded as $\mathbf{1} + \cdots + \mathbf{1}$.

3.2 Embedding

Fix a polynomial-time Turing machine $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_{\text{final}})$. Assume that the running time of M is bounded above by $p(n)$ for some polynomial $p : \mathbb{N} \to \mathbb{N}$. The target program of the translation will be denoted by M', and this is what we are about to define. M' works as described in Algorithm 1. A RaML implementation of M' is available in Appendix A.3 of [30].

Algorithm 1 Operational working of target RaML program M'.

```
Require: $w \in \{0,1\}^*$
procedure $M'(w)$
    Create a singleton list $\ell_1 : L(\text{Sym})$ containing $\vdash$
    Create a list $\ell_2 : L(\text{Sym})$ of size $p(|w|)$ filled with $\sqcup$
    Prepend $\ell_2$ with $w$
    Create a list $ps : L(\mathbf{1})$ of size $p(|w|)$          ▷ Reservoir of potential
    $s \leftarrow q_0$                                            ▷ Initialize the current state
    while $s \neq q_{\text{final}} \wedge ps \neq [\,]$ do
        $ps \leftarrow$ tail $ps$                                  ▷ Potential is released
        Compute $\delta(s, \ell_2[0])$
        Update $s$ and $\ell_2[0]$ appropriately
        Update the tape head's position by moving the head of $\ell_1$ or $\ell_2$ to the other
    return append(reverse $\ell_1, \ell_2$)
```

The list ℓ_1 represents the region on M's tape to the left of the tape head (in the reverse order and excluding the cell where the tape head is currently on), and ℓ_2 represents the region to the right of the head (including the current cell). Since it is assumed that $p(|w|)$, where $|w|$ denotes the length of input list w, is an upper bound on M's running time, we are assured that M requires at most $p(|w|)$ many cells on the working tape. This is why ℓ_2 initially has size $p(|w|)$. In fact, because we prepend ℓ_2 with w in line 4, we have $|w|$ more cells than necessary.

The list *ps* acts as a reservoir of potential, storing constant potential in each element. As the head of *ps* is removed in line 8, the potential stored in this element is freed and will be consumed in subsequent lines inside the loop's body.

It is technically possible to store potential directly in ℓ_1 and ℓ_2, which together simulate M's working tape. However, not all cells on the working tape of M are accessed equally often – some cells are accessed more often than others, and the maximum number of accesses to a given cell may not be bounded by a constant. If we are to store potential in ℓ_1 and ℓ_2, each cell of ℓ_1 and ℓ_2 needs to store $p(n)$ units of potential at the beginning. As a result, the total amount of potential supplied to M' is $p^2(n)$, which is a gross over-approximation of the actual running time. Therefore, to have a tighter cost bound, a separate list, namely *ps*, is employed as a reservoir of potential.

4 Inherently Polynomial Time

Section 3 investigates the expressive power of AARA from the viewpoint of programming language semantics, disregarding the issue of how to algorithmically turn an arbitrary Turing machine into a typable RaML program. By contrast, in this section, we aim to identify a typable fragment of AARA that is defined statically/axiomatically. Henceforth, we will call the sufficient condition corresponding to the typable fragment that this section presents *inherently polynomial time*.

A key requirement is that the typable fragment should not resemble AARA's type system, which itself is also defined axiomatically. Otherwise, it would be trivial to prove that any term in this fragment is typable in AARA. Because we want users of AARA to benefit from our findings of the present work, another requirement is that the definition (or at least the informal definition) of inherently polynomial time should be easy to convey to users of AARA. On the other hand, it is not our priority to find as large a typable fragment as we can.

In the remaining of the article, we will focus on the running time as a cost metric of RaML, unless stated otherwise.

4.1 High-Level Design

By Theorem 1 (and its multivariate equivalent), AARA is sound: if a program is typable in AARA, its resource-annotated type is a correct upper bound on the running time. Hence, to be typable in AARA, the worst-case running time of a program must be polynomial. To ensure termination of programs, we first restrict recursion to primitive recursion.

Furthermore, the type system of AARA is compositional: if term e is typable, so is every sub-expression of e. Hence, in order for e to be typable, not only e but also all of its sub-expressions must be polynomial-time. This suggests that we should define the sufficient condition inductively, hence the name *inherently* polynomial time.

It is straightforward to determine whether each of the base cases of the inductive definition is typable or not. It remains to work out inductive cases in the inductively defined sufficient condition for typability. The most interesting case is primitive recursion. A primitive recursion will be written as

$$e := \mathsf{rec}\ x\ \{[\,] \hookrightarrow e_0 \mid (y :: ys)\ \mathsf{with}\ z \hookrightarrow e_1\},$$

where x is matched against $y :: ys$ in the second branch, and z is the result of a recursive call. The stepping function e_1 can only contain y, ys, and z as free variables; i.e. $\mathtt{FV}(e_1) \subseteq \{y, ys, z\}$. From compositionality, we know that e_0 and e_1 are both typable and hence run in polynomial time. Under what condition does the entire e run in polynomial time as well?

To answer this question, we first observe the following. Without any restrictions on e_0 and e_1 apart from that they should be typable, AARA may project e's worst-case time complexity to be exponential even if the actual running time of e is polynomial. To illustrate this, consider

$$e := \mathsf{rec}\ x\ \{[\,] \hookrightarrow [\,] \mid (y :: ys)\ \mathsf{with}\ z \hookrightarrow \mathsf{share}\ z\ \mathsf{as}\ z_1, z_2\ \mathsf{in}\ \mathit{append}\ \langle z_1, z_2\rangle\}. \tag{4.1}$$

Although the actual running time of e is $O(|x|)$ and hence is linear, e is untypable in polynomial AARA. The problem of (4.1) is that the stepping function doubles the input size. This makes AARA conclude (naïvely) that the worst-case total running time is $O(2^{|x|})$, and this cost bound is beyond the expressive power of AARA (exponential AARA [22], however, can handle exponential cost bounds).

To preclude the example (4.1), it is reasonable to require the running time of e_1 (i.e. a stepping function inside primitive recursion) to be constant in the size of z (i.e. the result of a recursive call). More concretely, if $T(|y|, |ys|, |z|)$ is the running time of a stepping function, we demand $T(|y|, |ys|, |z|) \leq p(|y|, |ys|)$, where $p(|y|, |ys|)$ is a polynomial in $|y|$ and $|ys|$ (i.e. the sizes of y's and ys's semantic values[1]). We will adopt this idea in the formulation of inherently polynomial time.

Although this idea results in a fairly simple inductive definition of inherently polynomial time, a major drawback is that some realistic programs are not admitted by the current formulation of inherently polynomial time. For instance, consider *multiply* that, given input lists ℓ_1 and ℓ_2, produces a list of size $|\ell_1| \cdot |\ell_2|$:

$$\mathit{multiply} := \lambda \ell_1.\lambda \ell_2.\mathsf{rec}\ \ell_1\ \{[\,] \hookrightarrow \langle \ell_2, [\,]\rangle \mid (y :: ys)\ \mathsf{with}\ z \hookrightarrow e_1\}, \tag{4.2}$$

where the stepping function of primitive recursion is

$$e_1 \equiv \mathsf{case}\ z\ \{\langle z_1, z_2\rangle \hookrightarrow \mathsf{share}\ z_1\ \mathsf{as}\ z_{1,1}, z_{1,2}\ \mathsf{in}\ \langle z_{1,1}, \mathit{append}\ \langle z_{1,2}, z_2\rangle\rangle\}.$$

The first component of z stores ℓ_2, while the second component of z acts as an accumulator. The running time of e_1 is polynomial in $|z_1|$ but constant in $|z_2|$. Therefore, e_1's running time is only polynomial *partially* in $|z|$. This is why the overall time complexity of e remains polynomial instead of becoming exponential. Nonetheless, (4.2) is not inherently polynomial time according to the current formulation, since the formulation does not allow e_1's running time to have any dependence on $|z|$.

Furthermore, (4.2) can only be typed in multivariate AARA and not in univariate AARA. This means our formulation of inherently polynomial time fails to capture some of the realistic programs that are typable only in multivariate AARA. In view of this, one might wonder whether inherently polynomial time is completely encapsulated by univariate AARA; that is, every inherently polynomial-time RaML program is typable in univariate AARA. The answer is negative.

As a counterexample, consider the standard *append* defined as

$$\mathit{append} := \lambda \ell_1.\lambda \ell_2.\mathsf{rec}\ \ell_1\ \{[\,] \hookrightarrow \ell_2 \mid (y :: ys)\ \mathsf{with}\ z \hookrightarrow y :: z\}. \tag{4.3}$$

[1] A formal definition of the size of RaML's base-type semantic values is not provided in this article. However, the idea is intuitive. For example, the size of a list is given by the sum of all elements' sizes.

Note that it is inherently polynomial time. *append* alone is typable in univariate AARA as well as multivariate AARA. However, if we require the output of *append* to carry quadratic potential (because it will be later fed to a function that demands quadratic potential from inputs, for example), then univariate AARA cannot type *append* – we need to resort to multivariate AARA to type it.

In summary, our formulation of inherently polynomial time goes beyond the remit of univariate AARA, but does not capture the full range of realistic programs that require multivariate potential.

4.2 Formulation of Inherently Polynomial Time

Restricting the Syntax of Resource-Aware ML

To ensure termination of programs, we require programs to use primitive recursion in place of general recursion. Hence, we will from now on work with a fragment of RaML wherein general recursion is replaced by primitive recursion. This fragment removes $\mathsf{fun}\ f\ x = e$ from the original RaML (Section 2.1) and adds the following:

1. $\lambda(x : b).e$ for a lambda abstraction, where $b \in \mathbb{B}$;
2. $\mathsf{rec}\ x\ \{[\,] \hookrightarrow e_0 \mid (y :: \mathit{ys})\ \mathsf{with}\ z \hookrightarrow e_1\}$, where z denotes the result of the recursive call.

In primitive recursion, e_1 is only allowed to mention $\{y, \mathit{ys}, z\}$. If e_1 needs access to a global variable v (i.e. a variable from outside the primitive recursion), v should be transferred to e_1 by placing v inside z.

The reason why we deny e_1 access to a global variable is that every variable symbol can only be accessed at most once in RaML. However, this is in fact already violated by e_1 having access to ys (because this means some elements of the input x are accessed multiple times during primitive recursion). Further, even if we let e_1 access global variables, AARA can be easily adapted. Also, it will result in a less strict formulation of inherently polynomial time that admits *multiply* in (4.2). Nonetheless, for simplicity, this article assumes that e_1 can only mention y, ys, and z.

Primitive recursion can be encoded using general recursion as

$$\mathsf{fun}\ f\ \langle x, \Gamma\rangle = \mathsf{case}\ x\ \{[\,] \hookrightarrow e_0 \mid y :: \mathit{ys} \hookrightarrow \mathsf{share}\ \mathit{ys}\ \mathsf{as}\ \mathit{ys}_1, \mathit{ys}_2\ \mathsf{in}\ \mathsf{let}\ z = f\ \langle \mathit{ys}_1, \Gamma\rangle\ \mathsf{in}\ e_1\}.$$

Here, Γ is a set/sequence of those variables that do not appear in e_1, but e_0. Variable ys_1 is passed to the recursive call, and ys_2 is used in e_1 (if e_1 mentions ys).

Judgments

The primary judgment of inherently polynomial time is

$$\Delta; \Gamma \vdash e\ \mathsf{inhpoly}(V), \tag{4.4}$$

where

- Γ is a typing context containing both base-type and arrow-type variables such that $\Gamma \vdash e : b$ for base type b.
- $V \subseteq \mathrm{dom}(\Gamma)$ is a set of variables.
- Δ is a set of f time, where $f \in \mathrm{dom}(\Gamma)$ is an arrow-type variable and $\mathsf{time} \in \{\mathsf{const}, \mathsf{poly}\}$.

Sometimes we split Γ into Σ for arrow-type variables and Γ for base-type variables, writing the judgment as $\Delta; \Sigma; \Gamma \vdash e\ \mathsf{inhpoly}(V)$. (4.4) is only applicable to base-type expressions e.

An informal interpretation of (4.4) is

- f const denotes that the running time of f is constant with respect to the input size, and likewise, f poly denotes that f's running time is polynomial[2] in the input size.
- The running time of e is (i) polynomial[3] in the sizes of those variables in V but (ii) constant in the sizes of $\mathrm{dom}(\Gamma) \setminus V$.
- Every sub-expression of e runs in polynomial time.

The judgments for an arrow-type expression e are

$$\Delta; \Gamma \vdash e \ \mathsf{const} \qquad \Delta; \Gamma \vdash e \ \mathsf{poly}, \tag{4.5}$$

$\Delta; \Gamma \vdash e$ const means e runs in constant time with respect to the input size, and $\Delta; \Gamma \vdash e$ poly likewise means e's running time is polynomial in the input size.

Inference Rules

The most important inference rules defining (4.4) are displayed in Figure 1. Throughout these rules, b denotes a base type, time is drawn from $\{\mathsf{const}, \mathsf{poly}\}$, and V is a set of variables. The remaining rules are deferred to Figure 10 in Appendix E of [30].

In (IP:Case-Sum), the notation $V[x \mapsto y]$ refers to the result of replacing x in V with y (if $x \in V$); otherwise, V remains intact. If the running time of $\mathsf{case}\ x\ \{\ell \cdot y \hookrightarrow e_\ell \mid r \cdot y \hookrightarrow e_r\}$ in the rule's conclusion is allowed to be polynomial in $|x|$ (i.e. $x \in V$), then $e_{i \in \{\ell, r\}}$ in the two premises is allowed to run in polynomial time in $|y| = |x| - 1$.

Similarly, in (IP:Case-Prod), $V[x \mapsto x_1, x_2]$ means $(V \setminus \{x\}) \cup \{x_1, x_2\}$ if $x \in V$; otherwise, V remains unchanged.

(IP:Rec) is the crux of the notion of inherently polynomial time. Observe that the stepping function e_1 must be constant-time in $|z|$ (i.e. the size of z's semantic value).

In (IP:Let-Base), we use a finer-grained notation where the typing context of e_1 is split into Σ_1 for arrow-type variables and Γ_1 for base-type variables. V_3 is determined by

$$V_3 := \begin{cases} \mathrm{dom}(\Gamma_1) \cup (V_2 \setminus \{x\}) & \text{if } x \in V_2; \\ V_1 \cup V_2 & \text{otherwise.} \end{cases}$$

If $x \in V_2$, it means that e_2 runs in polynomial time in $|x|$. In the worst case, not only the running time of e_1 but $|e_1|$ (i.e. the output size of e_1) is polynomial in the sizes of those variables in V_1. Hence, in the worst case, the overall running time of $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ is polynomial in $\mathrm{dom}(\Gamma_1)$, which contains all base-type variables appearing in e_1, and $V_2 \setminus \{x\}$. Note that (IP:Let-Base) considers the worst case – if we had information about the output size, we might be able to derive a more precise judgment.

Finally, the judgment (4.5) is defined by the following inference rules:

$$\frac{\Delta; x : b \vdash e \ \mathsf{inhpoly}(\emptyset)}{\Delta; \cdot \vdash \lambda(x : b).e \ \mathsf{const}} \ (\text{IP:Const}) \qquad \frac{\Delta; x : b \vdash e \ \mathsf{inhpoly}(\{x\})}{\Delta; \cdot \vdash \lambda(x : b).e \ \mathsf{poly}} \ (\text{IP:Poly})$$

In (IP:Const), because the conclusion indicates that the λ-abstraction's running time is constant in the input size, the premise states that the running time of the body e can only be polynomial in $\mathrm{dom}(\Gamma)$, which excludes x. By contrast, in the premise of (IP:Poly), the set of variables contains x.

[2] f's running time being polynomial does NOT mean that it is *strictly* polynomial – it can also be constant in the input size.

[3] Again, the running time of e may be constant as well as polynomial in the size of any $v \in V$.

$$\frac{}{\cdot; x : b \vdash x\ \mathsf{inhpoly}(\emptyset)}\ (\text{IP:}\textsc{Base}) \qquad \frac{\Delta = \{f\ \mathsf{time}\}}{\Delta; f : b_1 \to b_2 \vdash f\ \mathsf{time}}\ (\text{IP:}\textsc{Arrow})$$

$$\frac{\cdot; x : b \vdash x\ \mathsf{inhpoly}(\emptyset)}{\cdot; x : b \vdash \ell \cdot x\ \mathsf{inhpoly}(\emptyset)}\ (\text{IP:}\textsc{SumL}) \qquad \frac{\cdot; x : b \vdash x\ \mathsf{inhpoly}(\emptyset)}{\cdot; x : b \vdash r \cdot x\ \mathsf{inhpoly}(\emptyset)}\ (\text{IP:}\textsc{SumR})$$

$$\frac{\cdot; x_1 : b_1 \vdash x_1\ \mathsf{inhpoly}(\emptyset) \qquad \cdot; x_2 : b_2 \vdash x_2\ \mathsf{inhpoly}(\emptyset)}{\cdot; x_1 : b_1, x_2 : b_2 \vdash \langle x_1, x_2 \rangle\ \mathsf{inhpoly}(\emptyset)}\ (\text{IP:}\textsc{Pair})$$

$$\frac{}{\cdot; \cdot \vdash \langle\,\rangle\ \mathsf{inhpoly}(\emptyset)}\ (\text{IP:}\textsc{Unit}) \qquad \frac{\Delta = \{x_1\ \mathsf{const}\}}{\Delta; x_1 : b_1 \to b_2, x_2 : b_1 \vdash x_1\ x_2\ \mathsf{inhpoly}(\emptyset)}\ (\text{IP:}\textsc{App-Const})$$

$$\frac{}{\cdot; \cdot \vdash [\,]\ \mathsf{inhpoly}(\emptyset)}\ (\text{IP:}\textsc{Nil}) \qquad \frac{\Delta = \{x_1\ \mathsf{poly}\}}{\Delta; x_1 : b_1 \to b_2, x_2 : b_1 \vdash x_1\ x_2\ \mathsf{inhpoly}(\{x_2\})}\ (\text{IP:}\textsc{App-Poly})$$

$$\frac{\cdot; x_1 : b \vdash x_1\ \mathsf{inhpoly}(\emptyset) \qquad \cdot; x_2 : L(b) \vdash x_2\ \mathsf{inhpoly}(\emptyset)}{\cdot; x_1 : b, x_2 : L(b) \vdash x_1 :: x_2\ \mathsf{inhpoly}(\emptyset)}\ (\text{IP:}\textsc{Cons})$$

$$\frac{\Delta; \Gamma, y : b_1 \vdash e_\ell\ \mathsf{inhpoly}(V[x \mapsto y]) \qquad \Delta; \Gamma, y : b_2 \vdash e_r\ \mathsf{inhpoly}(V[x \mapsto y])}{\Delta; \Gamma, x : b_1 + b_2 \vdash \mathsf{case}\ x\ \{\ell \cdot y \hookrightarrow e_\ell \mid r \cdot y \hookrightarrow e_r\}\ \mathsf{inhpoly}(V)}\ (\text{IP:}\textsc{Case-Sum})$$

$$\frac{\Delta; \Gamma, x_1 : b_1, x_2 : b_2 \vdash e\ \mathsf{inhpoly}(V[x \mapsto x_1, x_2])}{\Delta; \Gamma, x : b_1 \times b_2 \vdash \mathsf{case}\ x\ \{\langle x_1, x_2 \rangle \hookrightarrow e\}\ \mathsf{inhpoly}(V)}\ (\text{IP:}\textsc{Case-Prod})$$

$$\frac{\Delta; \Gamma \vdash e_0\ \mathsf{inhpoly}(V \setminus \{x\}) \qquad \Delta; \Gamma, x_1 : b, x_2 : L(b) \vdash e_1\ \mathsf{inhpoly}(V[x \mapsto x_1, x_2])}{\Delta; \Gamma, x : L(b) \vdash \mathsf{case}\ x\ \{[\,] \hookrightarrow e_0 \mid (x_1 :: x_2) \hookrightarrow e_1\}\ \mathsf{inhpoly}(V)}\ (\text{IP:}\textsc{Case-List})$$

$$\frac{\Delta; \Gamma \vdash e_0\ \mathsf{inhpoly}(V) \qquad \cdot; y : b, ys : L(b), z : b_2 \vdash e_1\ \mathsf{inhpoly}(\{y, ys\})}{\Delta; \Gamma, x : L(b) \vdash \mathsf{rec}\ x\ \{[\,] \hookrightarrow e_0 \mid (y :: ys)\ \mathsf{with}\ z \hookrightarrow e_1\}\ \mathsf{inhpoly}(V \cup \{x\})}\ (\text{IP:}\textsc{Rec})$$

$$\frac{\Delta_1; \Sigma_1; \Gamma_1 \vdash e_1\ \mathsf{inhpoly}(V_1) \qquad \Delta_2; \Gamma_2, x : b \vdash e_2\ \mathsf{inhpoly}(V_2)}{\Delta_1 \cup \Delta_2; \Sigma_1 \cup \Gamma_1 \cup \Gamma_2 \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2\ \mathsf{inhpoly}(V_3)}\ (\text{IP:}\textsc{Let-Base})$$

$$\frac{\Delta; \Gamma, x_1 : b, x_2 : b \vdash e\ \mathsf{inhpoly}(V[x \mapsto x_1, x_2])}{\Delta; \Gamma, x : b \vdash \mathsf{share}\ x\ \mathsf{as}\ x_1, x_2\ \mathsf{in}\ e\ \mathsf{inhpoly}(V)}\ (\text{IP:}\textsc{Share-Base})$$

Figure 1 Key inference rules of inherently polynomial time.

5 Typable Fragment of Resource-Aware ML

It is nontrivial to prove that inherently polynomial time (Section 4.2) implies typability in multivariate AARA. The chief challenge is to come up with a suitable statement of a typability theorem (i) that we can prove by induction and (ii) that satisfies the following two requirements. Firstly, because a term e may later be used as an input to a function, it must be possible to type e such that a user-specified (i.e. arbitrary) amount of potential remains in e's output. Secondly, to type primitive recursion, we need to establish an invariant of resource annotations that is analogous to a loop invariant in Hoare logic. Specifically, given a primitive recursion $\mathsf{rec}\ x\ \{[\,] \hookrightarrow e_0 \mid (y :: ys)\ \mathsf{with}\ z \hookrightarrow e_1\}$, we must give an (almost) identical annotation to both z, which is the result of a recursive call, and e_1, which is a stepping function.

Typability Theorem

We have partially overcome this challenge, and this section presents the result that inherently polynomial time implies typability in multivariate AARA under some restrictions. Detailed proofs of Theorem 6 and Theorem 9 are available in Appendix E of [30].

▶ **Definition 4** (Variables with zero potential). *Let $\Gamma \cup \{v : b\}$ be a base-type typing context and P be its multivariate annotation. Variable v is said to contain zero potential in P if and only if $P(i, j) = 0$ for every $i \in \mathcal{I}(\Gamma)$ and $j \in \mathcal{I}(\{v : b\})$ such that $j \neq 0_b$. In other words, the potential represented by P is constant with respect to $|v|$.*

▶ **Assumption 5.** *Suppose we are given $\Delta; \Sigma; \Gamma \vdash e\ t$ for $t \in \{\mathsf{inhpoly}(V), \mathsf{const}, \mathsf{poly}\}$. For every sub-derivation $\Delta_s; \Sigma_s; \Gamma_s \vdash e_s\ \mathsf{inhpoly}(V_s)$ inside the derivation of $\Delta; \Sigma; \Gamma \vdash e\ t$, we assume the following:*

- *If $e_s \equiv \mathsf{share}\ v\ \mathsf{as}\ v_1, v_2\ \mathsf{in} \cdots$, then v must be in V_s;*
- *If $e_s \equiv \mathsf{case}\ x\ \{[\,] \hookrightarrow \cdots \mid (y :: ys) \hookrightarrow \cdots\}$, then the type of x is of the form $L(b)$ where $b \in \mathbb{B}$ does not contain a list type; that is, x cannot be a nested list.*

The next theorem establishes that inherently polynomial time implies typability in multivariate AARA under Assumption 5, which restricts variable sharing and pattern matching on nested lists.

▶ **Theorem 6** (Inherently polynomial time implies typability). *Suppose we are given a term $\Sigma; \Gamma \vdash e : b$ with base type $b \in \mathbb{B}$, where $\Delta; \Sigma; \Gamma \vdash e\ \mathsf{inhpoly}(V)$ holds for some $V \subseteq dom(\Gamma)$. Additionally, assume Assumption 5. There exist P and Q satisfying $\Sigma; \Gamma; P \vdash e : \langle b, Q\rangle$ such that each $v \in dom(\Gamma) \setminus V$ contains zero potential (Definition 4).*

Consider an arrow-type term $\Sigma; \cdot \vdash e : b_1 \to b_2$ and assume Assumption 5. There exist P and Q such that $\Sigma; \cdot; 1 \vdash e : \langle b_1, P\rangle \to \langle b_2, Q\rangle$. Additionally, if $\Delta; \cdot; \Gamma \vdash e\ \mathsf{const}$ is true, P contains constant potential; i.e. b_1 stores zero potential in P.

Given a base-type expression e, if $\Delta; \Sigma; \Gamma \vdash e\ \mathsf{inhpoly}(V)$ holds, the running time of e is constant in the size of any $v \in \mathrm{dom}(\Gamma) \setminus V$. In other words, such v does not contribute to the computational cost of e. Therefore, it intuitively makes sense that such v contains zero potential in Theorem 6.

However, Theorem 6 cannot be immediately proved by induction on $\mathsf{inhpoly}(V)$, since the statement of the theorem is not strong enough for an inductive proof to go through. Specifically, a problem arises in the inductive case for (IP:Let-Base). In a let-binding $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$, e_1 must carry sufficient potential to be transferred to e_2. However, Theorem 6 does not allow us to specify how much potential will remain available in the output of e.

Prior to remedying this issue, we first introduce the notion of *uniform resource annotations* for multivariate AARA.

▶ **Definition 7** (Uniform resource annotations for base types in multivariate AARA). *Given a base type $b \in \mathbb{B}$, let P be a multivariate resource annotation of b. P is said to be a* uniform multivariate annotation *with degree $d \in \mathbb{N}$ and number $n \in \mathbb{N}$ if and only if the following conditions hold*

1. *The maximum degree of P is at most d;*
2. *$P(i) = n$ for every $i \in \mathcal{I}(b)$ such that $\mathsf{deg}(i) = d$.*

In words, all coefficients of base polynomials with degree d (which should be the maximum degree) are equal to n. This will be denoted by a judgment $P\ \mathsf{uniform}(d, n)$.

▶ **Definition 8** (Uniform annotations for typing contexts in multivariate AARA)**.** *Consider a term $\Sigma; \Gamma \vdash e : b$ of base type. Suppose that $\Delta; \Sigma; \Gamma \vdash e$ $\mathsf{inhpoly}(V)$ holds. Let P be a multivariate annotation for the base-type typing context Γ. We say that P is* uniform *with respect to degree $d \in \mathbb{N}$, number $n \in \mathbb{N}$, and set V of variables if and only if the following conditions hold:*

1. *For any base-type variable $v \in dom(\Gamma) \setminus V$ of type b_v, we have*

$$\forall i \in \mathcal{I}(\{v : b_v\}), j \in \mathcal{I}(\Gamma \setminus \{v : b_v\}).\mathsf{deg}(i) > d \implies P(i, j) = 0.$$

   In words, for any base polynomial with a non-zero coefficient in P, its projection on v must have degree at most d.

2. *For any $v \in dom(\Gamma) \setminus V$ of base type b_v, we have*

$$\forall i \in \mathcal{I}(\{v : b_v\}), j \in \mathcal{I}(\Gamma \setminus \{v : b_v\}).(\mathsf{deg}(i) = d \wedge j \neq 0) \implies P(i, j) = 0.$$

   In words, if a base polynomial has a non-zero coefficient and its projection on v has degree d, then the base polynomial is not allowed to involve size variables of any other base-type variables from $dom(\Gamma)$.

3. *For any $v \in dom(\Gamma) \setminus V$ of base type b_v, we have*

$$\forall i \in \mathcal{I}(\{v : b_v\}).\mathsf{deg}(i) = d \implies P(i, 0) = n.$$

   That is, every base polynomial whose projection on v has degree d has coefficient n.

If these conditions hold, we denote P being a uniform annotation by a judgment P $\mathsf{uniform}(d, n, V)$.

Note that Definition 8 is a generalization of Definition 7. P $\mathsf{uniform}(d, n)$ in Definition 7 is equivalent to P $\mathsf{uniform}(d, n, \emptyset)$ in Definition 8.

Now that we have the notion of uniform annotations in place, we next present Theorem 9 that allows us to specify the amount of potential remaining in the output of a program. The major difficulty of the proof lies in establishing an invariant for primitive recursion as explained at the start of Section 5. We employ the notion of uniform annotations to characterize this invariant.

▶ **Theorem 9** (Existence of a multivariate annotation with arbitrary potential in the output)**.** *Given a term $\Sigma; \Gamma \vdash e : b$ with $b \in \mathbb{B}$, suppose that $\Delta; \Sigma; \Gamma \vdash e$ $\mathsf{inhpoly}(V)$ holds, where $V \subseteq dom(\Gamma)$. Also, assume Assumption 5. Fix a multivariate annotation Q for the base type b such that Q $\mathsf{uniform}(d, n)$. Then there exists a multivariate annotation P such that $\Sigma; \Gamma; P \vdash e : \langle b, Q \rangle$ under the cost-free metric. Furthermore, P $\mathsf{uniform}(d, n, V)$ holds.*

Consider an arrow-type term $\Sigma; \cdot \vdash e : b_1 \to b_2$ and assume Assumption 5. Fix a multivariate annotation Q for base type b_2 such that Q $\mathsf{uniform}(d, n)$. Then there exists P such that $\Sigma; \cdot; 0 \vdash e : \langle b_1, P \rangle \to \langle b_2, Q \rangle$ under the cost-free metric. Furthermore, if $\Delta; \Sigma; \cdot \vdash e$ const is true, P $\mathsf{uniform}(d, n)$ holds.

The cost-free metric in Theorem 9 refers to the cost metric in which all evaluation costs are zero. For instance, if $f : L(\mathbf{1}) \to L(\mathbf{1})$ is a function that doubles the size of an input list, it can be typed as $f : \langle L^2(\mathbf{1}), 0 \rangle \to \langle L^1(\mathbf{1}), 0 \rangle$ under the cost-free metric[4]. That is, the potential stored in each element is halved because the length of the list is doubled. If the cost

[4] For readability, I use univariate AARA instead of multivariate AARA to denote resource-annotated types, although Theorem 9 concerns multivariate AARA

metric is the running time, we instead have $f : \langle L^{2+c}(\mathbf{1}), 0\rangle \to \langle L^1(\mathbf{1}), 0\rangle$, where c is the cost of processing each list element. The type system of multivariate AARA under the cost-free metric is provided in Appendix D.2 of [30]. Theorem 9 uses the cost-free metric (as opposed to the running time) since Theorem 6 has already considers the cost of evaluating programs.

Theorem 9 assumes Assumption 5 as the proof of the theorem poses technical challenges in variable sharing and pattern matching on nested lists. We will now look at these challenges more closely.

Variable Sharing

Theorem 9 is false if we impose no restrictions on variable sharing. To illustrate this, consider e defined as

$$e := \mathsf{rec}\ x\ \{[\,] \hookrightarrow \langle \ell, \ell\rangle \mid (y :: \_)\ \mathsf{with}\ z \hookrightarrow e_1\}, \tag{5.1}$$

where the stepping function is $e_1 \equiv \mathsf{case}\ z\ \{\langle z_1, z_2\rangle \hookrightarrow \mathsf{share}\ z_1\ \mathsf{as}\ z_{1,1}, z_{1,2}\ \mathsf{in}\ \langle z_{1,1}, z_{1,2}\rangle\}$. The typing context of e in (5.1) is $\Gamma = \{x : L(\mathbf{1}), \ell : L(\mathbf{1})\}$. The stepping function satisfies $e_1\ \mathsf{inhpoly}(\{y, ys\})$. Hence, (5.1) is indeed inherently polynomial time. However, inside e_1, we have $\mathsf{share}\ z_1$, which Assumption 5 forbids.

Let $\langle \ell_1, \ell_2\rangle$ be the output of (5.1). Suppose that both ℓ_1 and ℓ_2 are to be annotated with $L^1(\mathbf{1})$. To type (5.1) under the cost-free metric such that $\ell_1, \ell_2 : L^1(\mathbf{1})$, the typing context Γ of e needs to be annotated with $2|\ell| + |x| \cdot |\ell|$, where $|\cdot|$ denotes the size of an input list. Observe that we need to use multivariate AARA rather than univariate AARA to type (5.1).

In the notation[5] of univariate AARA, the stepping function of (5.1) can be typed as

$$y : \mathbf{1}, ys : L^0(\mathbf{1}), z : L^2(\mathbf{1}) \times L^0(\mathbf{1}); 0 \vdash e_1 : \langle L^1(\mathbf{1}) \times L^1(\mathbf{1}), 0\rangle.$$

Here, the maximum degree is $d = 1$. It is impossible for both z and e_1 to have the same coefficient for all base polynomials of degree $d = 1$. Therefore, Theorem 9 is false for (5.1). To accommodate the multivariate annotation of (5.1), it is necessary to relax the notion of uniform resource annotations, but this will make the typability proof more challenging.

Nested Lists in Pattern Matching

Theorem 9 is false for pattern matching on nested lists. For example, consider e defined as

$$e := \mathsf{case}\ x\ \{[\,] \hookrightarrow \_ \mid (y :: ys) \hookrightarrow \langle y, ys\rangle\},$$

where the first branch is unimportant in the present discussion. The typing context of e is $\Gamma = \{x : L(L(\mathbf{1}))\}$. Assume that we consider multivariate annotations of degree up to $d = 2$. Let P denote a multivariate annotation of Γ. The multivariate annotation for context $\{y : L(\mathbf{1}), ys : L(L(\mathbf{1}))\}$ as a result of pattern matching on $x : L(L(\mathbf{1}))$ is given by the *additive shift* of P, denoted by $\lhd(P)$. It is defined as

$$\lhd(P)(i, j) := \begin{cases} P(0_{L(\mathbf{1})} :: j) + P(j) & \text{if } i = 0_{L(\mathbf{1})}; \\ P(i :: j) & \text{otherwise,} \end{cases} \tag{5.2}$$

where $i \in \mathcal{I}(\{y : L(\mathbf{1})\})$ and $j \in \mathcal{I}(\{ys : L(L(\mathbf{1}))\})$. The problem is that the base polynomial (i, j) on the left hand side of (5.2) has degree $\mathsf{deg}(i) + \mathsf{deg}(j)$, while $(i :: j)$ in the second branch

[5] Although we are concerned with multivariate AARA, I will use univariate AARA to denote the resource annotation of e_1 because it happens to be describable by univariate AARA and it is easier to read.

of the right hand side has degree $1+\mathsf{deg}(i)+\mathsf{deg}(j)$. As a consequence, if $1+\mathsf{deg}(i)+\mathsf{deg}(j)=2$, $P(i :: j)$ is required to be equal to n because Theorem 9 requires P $\mathsf{uniform}(d,n)$ to be true. This means $\triangleleft(P)(i,j)=n$ must hold as well. But $\triangleleft(P)(i,j)=n$ is not necessarily the case, since Theorem 9 imposes no requirements on the coefficients of lower-degree base polynomials.

6 Conclusion

In this work, we have shown that polynomial-time Turing machines can be embedded in a typable fragment of RaML in such a way that the semantics and worst-case cost bounds are preserved. Moreover, we have proved that if a first-order program P satisfies the following conditions, it is guaranteed to be typable in multivariate polynomial AARA:

1. P uses primitive recursion instead of general recursion;
2. P is (axiomatically) inherently polynomial-time;
3. No variable sharing is applied to variable v, where P's running time is (axiomatically) constant in v;
4. No pattern matching is applied to a nested list.

We have neither found a counterexample to the full typability theorem (i.e. Theorem 6 without Assumption 5) nor proved it. As future work, we are looking to investigate how to prove or disprove the full typability theorem. To lift the restriction on nested lists, we expect that it suffices to modify the statement of the theorem such that we can keep track of the largest coefficient. However, lifting the restriction on variable sharing will be more challenging because it certainly requires a drastically different inductive hypothesis.

References

1 Elvira Albert, Puri Arenas, Samir Genaim, Germán Puebla, and Damiano Zanardini. Cost Analysis of Java Bytecode. In *16th Euro. Symp. on Prog. (ESOP'07)*, 2007.

2 Martin Avanzini and Ugo Dal Lago. Automating sized-type inference for complexity analysis. *Proc. ACM Program. Lang.*, 1(ICFP), August 2017. doi:10.1145/3110287.

3 Martin Avanzini and Georg Moser. A Combination Framework for Complexity. In *24th International Conference on Rewriting Techniques and Applications (RTA'13)*, 2013.

4 Spephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the polytime functions. *computational complexity*, 2(2):97–110, June 1992. doi:10.1007/BF01201998.

5 Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl. Alternating Runtime and Size Complexity Analysis of Integer Programs. In *20th Int. Conf. on Tools and Alg. for the Constr. and Anal. of Systems (TACAS'14)*, 2014.

6 Ezgi Çiçek, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Jan Hoffmann. Relational cost analysis. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, page 316–329, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3009837.3009858.

7 Krishnendu Chatterjee, Hongfei Fu, and Aniket Murhekar. Automated Recurrence Analysis for Almost-Linear Expected-Runtime Bounds. In *Computer Aided Verification - 29th Int. Conf. (CAV'17)*, 2017.

8 Karl Crary and Stephnie Weirich. Resource bound certification. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '00, page 184–198, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/325694.325716.

9 Nils Anders Danielsson. Lightweight semiformal time complexity analysis for purely functional data structures. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, page 133–144, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1328438.1328457.

10 Florian Frohn, M. Naaf, Jera Hensel, Marc Brockschmidt, and Jürgen Giesl. Lower Runtime Bounds for Integer Programs. In *Automated Reasoning - 8th International Joint Conference (IJCAR'16)*, 2016.

11 Bernd Grobauer. Cost recurrences for dml programs. In *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming*, ICFP '01, page 253–264, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/507635.507666.

12 Martin A. T. Handley, Niki Vazou, and Graham Hutton. Liquidate your assets: Reasoning about resource usage in liquid haskell. *Proc. ACM Program. Lang.*, 4(POPL), December 2019. doi:10.1145/3371092.

13 Jan Hoffmann. *Types with potential: polynomial resource bounds via automatic amortized analysis*. PhD thesis, Ludwig Maximilians University Munich, 2011. URL: http://edoc.ub.uni-muenchen.de/13955/.

14 Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate amortized resource analysis. *ACM Trans. Program. Lang. Syst.*, 34(3), November 2012. doi:10.1145/2362389.2362393.

15 Jan Hoffmann, Ankush Das, and Shu-Chun Weng. Towards automatic resource bound analysis for ocaml. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, page 359–373, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3009837.3009842.

16 Jan Hoffmann and Martin Hofmann. Amortized resource analysis with polynomial potential. In Andrew D. Gordon, editor, *Programming Languages and Systems*, pages 287–306, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

17 Martin Hofmann. The strength of non-size increasing computation. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '02, pages 260–269, New York, NY, USA, 2002. ACM. doi:10.1145/503272.503297.

18 Martin Hofmann and Steffen Jost. Static prediction of heap space usage for first-order functional programs. In *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '03, page 185–197, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/604131.604148.

19 Martin Hofmann and Georg Moser. Amortised Resource Analysis and Typed Polynomial Interpretations. In *Rewriting and Typed Lambda Calculi (RTA-TLCA;14)*, 2014.

20 Steffen Jost, Kevin Hammond, Hans-Wolfgang Loidl, and Martin Hofmann. Static determination of quantitative resource usage for higher-order programs. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '10, page 223–236, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1706299.1706327.

21 Steffen Jost, Pedro B. Vasconcelos, Mário Florido, and Kevin Hammond. Type-based cost analysis for lazy functional languages. *Journal of Automated Reasoning*, 59:87–120, 2017. doi:10.1007/s10817-016-9398-9.

22 David M. Kahn and Jan Hoffmann. Exponential automatic amortized resource analysis. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures*, pages 359–380, Cham, 2020. Springer International Publishing.

23 G. A. Kavvos, Edward Morehouse, Daniel R. Licata, and Norman Danner. Recurrence extraction for functional programs through call-by-push-value. *Proc. ACM Program. Lang.*, 4(POPL), December 2019. doi:10.1145/3371083.

24 Zachary Kincaid, Jason Breck, Ashkan Forouhi Boroujeni, and Thomas Reps. Compositional recurrence analysis revisited. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, page 248–262, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3062341.3062373.

25 Zachary Kincaid, John Cyphert, Jason Breck, and Thomas Reps. Non-linear reasoning for invariant synthesis. *Proc. ACM Program. Lang.*, 2(POPL), December 2017. doi:10.1145/3158142.

26 Ugo Dal Lago and Marco Gaboardi. Linear Dependent Types and Relative Completeness. In *26th IEEE Symp. on Logic in Computer Science (LICS'11)*, 2011.

27 Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of poly-time. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, pages 274–288, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

28 V. C. Ngo, M. Dehesa-Azuara, M. Fredrikson, and J. Hoffmann. Verifying and synthesizing constant-resource implementations with types. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 710–728, 2017.

29 Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. Bounded expectations: Resource analysis for probabilistic programs. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2018, page 496–512, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3192366.3192394.

30 Long Pham and Jan Hoffmann. Typable fragments of polynomial automatic amortized resource analysis, 2020. arXiv:2010.16353.

31 Ivan Radiček, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Florian Zuleger. Monadic refinements for relational cost analysis. *Proc. ACM Program. Lang.*, 2(POPL), December 2017. doi:10.1145/3158124.

32 Robert E. Tarjan. Amortized computational complexity. *SIAM Journal on Matrix Analysis and Applications*, 6(2):306–13, April 1985. Copyright - Copyright] © 1985 © Society for Industrial and Applied Mathematics; Last updated - 2012-02-28. URL: https://search-proquest-com.proxy.library.cmu.edu/docview/923648267?accountid=9902.

33 Pedro B. Vasconcelos. *Space cost analysis using sized types.* PhD thesis, University of St Andrews, UK, 2008. URL: http://hdl.handle.net/10023/564.

34 Ben Wegbreit. Mechanical program analysis. *Commun. ACM*, 18(9):528–539, September 1975. doi:10.1145/361002.361016.

The Yoneda Reduction of Polymorphic Types

Paolo Pistone
DISI, University of Bologna, Italy
paolo.pistone2@unibo.it

Luca Tranchini
Wilhelm-Schickard-Institut, Universität Tübingen, Germany
luca.tranchini@gmail.com

Abstract

In this paper we explore a family of type isomorphisms in System F whose validity corresponds, semantically, to some form of the Yoneda isomorphism from category theory. These isomorphisms hold under theories of equivalence stronger than $\beta\eta$-equivalence, like those induced by parametricity and dinaturality. We show that the Yoneda type isomorphisms yield a rewriting over types, that we call Yoneda reduction, which can be used to eliminate quantifiers from a polymorphic type, replacing them with a combination of monomorphic type constructors. We establish some sufficient conditions under which quantifiers can be fully eliminated from a polymorphic type, and we show some application of these conditions to count the inhabitants of a type and to compute program equivalence in some fragments of System F.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Program semantics

Keywords and phrases System F, Type isomorphisms, Yoneda isomorphism, Program equivalence

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.35

Related Version A full version of the paper is available at `https://arxiv.org/abs/1907.03481`.

Funding *Paolo Pistone*: ERC CoG 818616 "DIAPASoN".
Luca Tranchini: DFG TR1112/4-1 "Falsity and Refutation. On the negative side of logic".

1 Introduction

The study of type isomorphisms is a fundamental one both in the theory of programming languages and in logic, through the well-known *proofs-as-programs* correspondence: type isomorphisms supply programmers with transformations allowing them to obtain simpler and more optimized code, and offer new insights to understand and refine the syntax of type- and proof-systems.

Roughly speaking, two types A, B are isomorphic when one can transform any call by a program to an object of type A into a call to an object of type B, without altering the behavior of the program. Thus, type isomorphisms are tightly related to theories of *program equivalence*, which describe what counts as the observable behavior of a program, so that programs with the same behavior can be considered equivalent.

The connection between type isomorphisms and program equivalence is of special importance for polymorphic type systems like System F (hereafter $\Lambda 2$). In fact, while standard $\beta\eta$-equivalence for $\Lambda 2$ and the related isomorphisms are well-understood [10, 12], stronger notions of equivalence (as those based on *parametricity* or *free theorems* [37, 19, 1]) are often more useful in practice but are generally intractable or difficult to compute, and little is known about the type isomorphisms holding under such theories.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 35; pp. 35:1–35:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$
\begin{aligned}
\forall X.X \Rightarrow X \Rightarrow A &\equiv A[X \mapsto 1+1] && (*)\\
\forall X.(A \Rightarrow X) \Rightarrow (B \Rightarrow X) \Rightarrow C &\equiv C[X \mapsto \mu X.A + B] && (**)\\
\forall X.(X \Rightarrow A) \Rightarrow (X \Rightarrow B) \Rightarrow D &\equiv D[X \mapsto \nu X.A \times B] && (***)
\end{aligned}
$$

Figure 1 Other examples of Yoneda type isomorphisms, where X only occurs positively in A, B, C and only occurs negatively in D.

$$
\begin{aligned}
&\forall X.X \Rightarrow X \Rightarrow \forall Y.(\forall Z.(Z \Rightarrow X) \Rightarrow (\forall W.(W \Rightarrow Z) \Rightarrow W \Rightarrow X) \Rightarrow Z \Rightarrow Y) \Rightarrow (X \Rightarrow Y) \Rightarrow Y\\
&\overset{(*)}{\equiv} \forall Y.(\forall Z.(Z \Rightarrow 1+1) \Rightarrow (\forall W.(W \Rightarrow Z) \Rightarrow W \Rightarrow 1+1) \Rightarrow Z \Rightarrow Y) \Rightarrow (1+1 \Rightarrow Y) \Rightarrow Y\\
&\overset{(\star)}{\equiv} \forall Y.(\forall Z.(Z \Rightarrow 1+1) \Rightarrow (Z \Rightarrow 1+1) \Rightarrow Z \Rightarrow Y) \Rightarrow (1+1 \Rightarrow Y) \Rightarrow Y\\
&\overset{(***)}{\equiv} \forall Y.((\nu Z.(1+1) \times (1+1)) \Rightarrow Y) \Rightarrow (1+1 \Rightarrow Y) \Rightarrow Y\\
&\overset{(**)}{\equiv} \mu Y.(\nu Z.(1+1) \times (1+1)) + (1+1) \equiv 1+1+1+1+1+1
\end{aligned}
$$

Figure 2 Short proof that a Λ2-type has 6 inhabitants, using type isomorphisms.

Type Isomorphisms with the Yoneda Lemma. Our starting point is the observation that the *Yoneda lemma*, a cornerstone of category theory, is sometimes invoked [5, 16, 7, 35, 17] to justify some type isomorphisms in Λ2 like e.g.

$$
\forall X.(A \Rightarrow X) \Rightarrow (B \Rightarrow X) \equiv B \Rightarrow A \qquad \forall X.(X \Rightarrow A) \Rightarrow (X \Rightarrow B) \equiv A \Rightarrow B \qquad (\star)
$$

which do not hold under $\beta\eta$-equivalence, but only under stronger equivalences. Such isomorphisms are usually justified by reference to the interpretation of polymorphic programs as *(di)natural transformations* [5], a well-known semantics of Λ2 related to both parametricity [28] and free-theorems [36], and yielding a not yet well-understood equational theory over the programs of Λ2 [15, 11, 23], that we call here the *ε-theory*. Other isomorphisms, like those in Fig. 1, can be justified in a similar way as soon as the language of Λ2 is enriched with other type constructors like $1, 0, +, \times, \Rightarrow$ and *least/greatest fixpoints* $\mu X.A, \nu X.A$.

All such type isomorphisms have the effect of *eliminating* a quantifier, replacing it with a combination of monomorphic type constructors, and can be used to test if a polymorphic type has a *finite* number of inhabitants (as illustrated in Fig. 2) or, as suggested in [7], to devise *decidable tests* for program equivalence.

In this paper we develop a formal study of the elimination of quantifiers from polymorphic types using a class of type isomorphisms, that we will call *Yoneda type isomorphisms*, which generalize the examples above, and we explore its application for counting type inhabitants as well as to establish properties of program equivalence in Λ2.

Eliminating Quantifiers with Yoneda Reduction. To give the reader a first glimpse of our approach, we compare the use of Yoneda type isomorphisms to count type inhabitants with some well-known *sufficient conditions* for a *simple* type A to have a unique or finitely many inhabitants [2, 9]: we show that whenever a simple type A satisfies either of these conditions, its universal closure $\forall \vec{X}.A$ can be converted (as in Fig. 2) to either 1 or $1 + \cdots + 1$ by applying Yoneda type isomorphisms and usual $\beta\eta$-isomorphisms.

We then turn to investigate the quantifier-eliminating rewriting over types arising from the left-to-right orientation of Yoneda type isomorphisms. A major obstacle here is that the rewriting must take into account possible applications of $\beta\eta$-isomorphisms, whose axiomatization is challenging in presence of the constructors $+, 0$ [13, 18] (as well as μ, ν). For this reason we introduce a family of rewrite rules, that we call *Yoneda reduction*, defined not directly over types but over a class of finite trees which represent the types of Λ2 (but crucially *not* those made with $0, +, \dots$) up to $\beta\eta$-isomorphism.

Using this rewriting we establish some sufficient conditions for eliminating quantifiers, based on elementary graph-theoretic properties of such trees, which in turn provide some *new* sufficient conditions for the finiteness of type inhabitants of polymorphic types. First, we prove quantifier-elimination for the types satisfying a certain *coherence* condition which can be seen as an instance of the 2-SAT problem. We then introduce a more refined condition by associating each polymorphic type A with a value $\kappa(A) \in \{0, 1, \infty\}$, that we call the *characteristic of* A, so that whenever $\kappa(A) \neq \infty$, A rewrites into a monomorphic type, and when furthermore $\kappa(A) = 0$, A converges to a finite type. In the last case our method provides an effective way to count the inhabitants of A. The computation of $\kappa(A)$ is somehow reminiscent of linear logic *proof-nets*, as it is obtained by inspecting the existence of cyclic paths in a graph obtained by adding some "axiom-links" to the tree-representation of A.

Program Equivalence in System F with Finite Characteristic. Computing program equivalence under the ε-theory can be a challenging task, as this theory involves global permutations of rules which are difficult to detect and apply [11, 23, 34, 27, 36]. Things are even worse at the semantic level, since computing with dinatural transformations can be rather cumbersome, due to the well-known fact that such transformations need not compose [5, 21].

Nevertheless, our approach to quantifier-elimination based on the notion of characteristic provides a way to compute program equivalence *without* the appeal to ε-rules, free theorems and parametricity, since all polymorphic programs having types of finite characteristic can be embedded inside well-known monomorphic systems. To demonstrate this fact, we introduce two fragments $\Lambda2^{\kappa\leqslant 0}$ and $\Lambda2^{\kappa\leqslant 1}$ of $\Lambda2$ in which types have a fixed finite characteristic, and we prove that these are equivalent, under the ε-theory, respectively, to the simply typed λ-calculus with finite products and co-products (or, equivalently, to the *free bicartesian closed category* $\mathbb{B}$), and to its extension with μ, ν-types (that is, to the *free cartesian closed* μ*-bicomplete category* $\mu\mathbb{B}$ [29, 6]). Using well-known facts about $\mathbb{B}$ and $\mu\mathbb{B}$ [31, 6, 22], we finally establish that the ε-theory is decidable in $\Lambda2^{\kappa\leqslant 0}$ and undecidable in $\Lambda2^{\kappa\leqslant 1}$.

Preliminaries and Notations

We will presuppose familiarity with the syntax of $\Lambda2$ (in the version à la Church) and its extensions $\Lambda2\mathsf{p}, \Lambda2\mathsf{p}_{\mu\nu}$ with sum and product types, as well as μ and ν-types. We indicate by $\Lambda, \Lambda\mathsf{p}, \Lambda\mathsf{p}_{\mu\nu}$ their respective quantifier-free fragments. The syntax of these systems is recalled in App. A. We let $\mathsf{V} = \{X, Y, Z, \dots\}$ indicate the countable set of *type variables.*

Let S indicate any of the type systems above. We let $\Gamma \vdash_{\mathsf{S}} t : A$ indicate that the judgement $\Gamma \vdash t : A$ is derivable in S. We indicate as $t[x]$ a term with a unique free variable x, and we let $t[x] : A \vdash^{\Gamma}_{\mathsf{S}} B$ be shorthand for $\Gamma, x : A \vdash_{\mathsf{S}} t : B$.

A *theory* of S is a class of equations over well-typed terms satisfying usual congruence rules. Standard theories of $\Lambda2, \Lambda2\mathsf{p}, \Lambda2\mathsf{p}_{\mu\nu}$ are those generated by $\beta\eta$-equivalence and by contextual equivalence, recalled in App. A. We will also consider a less standard theory, the ε-theory, described in App. B. For all theory T including $\beta\eta$-equivalence, we let $\mathbb{C}_{\mathsf{T}}(\mathsf{S})$ be the category whose objects are the types of S and whose arrows are the T-equivalence classes of terms $t[x] : A \vdash_{\mathsf{S}} B$. $\mathbb{C}_{\mathsf{T}}(\mathsf{S})$ is cartesian closed as soon as S contains products, meaning in particular that $\mathbb{C}_{\mathsf{T}}(\mathsf{S})(A \times B, C) \simeq \mathbb{C}_{\mathsf{T}}(\mathsf{S})(A, B \Rightarrow C)$.

By a T-*isomorphism*, indicated as $A \equiv_{\mathsf{T}} B$, we mean a pair of terms $t[x] : A \vdash_{\mathsf{S}} B$, $u[x] : B \vdash_{\mathsf{T}} A$ such that $t[u[x]] \simeq_{\mathsf{T}} x$ and $u[t[x]] \simeq_{\mathsf{T}} x$ (where $t[u[x]]$ is $t[x \mapsto u]$).

2 Yoneda Type Isomorphisms

In this section we introduce an axiomatization for a class of type isomorphisms that we call *Yoneda type isomorphisms*. For this we will rely on the well-known distinction between *positive* and *negative* occurrences of a variable X in a type A.

▶ **Notation 2.1.** *Throughout the text we indicate as $\textcolor{blue}{X}$ (resp. $\textcolor{red}{X}$) a positive (resp. negative) occurrence of X. When B occurs within a larger type A, we often note B as $B\langle\textcolor{blue}{X}\rangle$ to indicate that all occurrences of the variable X in B are positive occurrences* in A, *or as $B\langle\textcolor{red}{X}\rangle$ to indicate that all occurrences of the variable X in B are negative occurrences* in A. *So for instance, when B only contains positive occurrences of X, we write the type $A = X \Rightarrow B$ as $\textcolor{red}{X} \Rightarrow B\langle\textcolor{blue}{X}\rangle$ (since all positive occurrences of X in B are positive in A) and the type $A' = B \Rightarrow X$ as $B\langle\textcolor{red}{X}\rangle \Rightarrow \textcolor{blue}{X}$ (since all positive occurrences of X in B are negative in A).*

The focus on positive/negative occurrences highlights a connection with to the so-called *functorial semantics* of $\Lambda 2$ [5, 15], in which types are interpreted as functors and typed programs as (di)natural transformations between such functors. More precisely, any positive type $A\langle\textcolor{blue}{X}\rangle$ gives rise to a functor $\Phi^X_A : \mathbb{C}_{\mathsf{T}}(\mathsf{S}) \to \mathbb{C}_{\mathsf{T}}(\mathsf{S})$, any negative type $A\langle\textcolor{red}{X}\rangle$ gives rise to a functor $\Phi^X_A : \mathbb{C}_{\mathsf{T}}(\mathsf{S})^{\mathsf{op}} \to \mathbb{C}_{\mathsf{T}}(\mathsf{S})$ and, more generally, any type A gives rise to a functor $\Phi^X_A : \mathbb{C}_{\mathsf{T}}(\mathsf{S})^{\mathsf{op}} \times \mathbb{C}_{\mathsf{T}}(\mathsf{S}) \to \mathbb{C}_{\mathsf{T}}(\mathsf{S})$. In all such cases, the action of the functor on a type A is obtained by replacing positive/negative occurrences of X by A, and the action on programs can be defined inductively (we recall this construction in App. A, see also [11, 24]).

With types being interpreted as functors, a polymorphic term $t[x] : A \vdash_{\Lambda 2} B$ is interpreted as a transformation satisfying an appropriate naturality condition: when A and B have the same variance, $t[x]$ is interpreted as an ordinary natural transformation; instead, if A and B have mixed variances, then $t[x]$ is interpreted as a dinatural transformation.

Such (di)naturality conditions can be described syntactically through a class of equational rules over typed programs [11, 23] generating, along with the usual $\beta\eta$-equations, a theory of program equivalence that we call the *ε-theory*.[1] These equational rules are usually interpreted as parametricity conditions [28], or as instances of *free theorems* [36].

As mentioned in the introduction, our goal here is not that of investigating the ε-theory directly, but rather to explore a class of type isomorphisms that hold under this theory (that is, of isomorphisms in the syntactic categories $\mathbb{C}_\varepsilon(\mathsf{S})$, with $\mathsf{S} = \Lambda 2, \Lambda 2\mathsf{p}, \Lambda 2\mathsf{p}_{\mu\nu}$). For example, in functorial semantics a type of the form $\forall X.A\langle\textcolor{red}{X}\rangle \Rightarrow B\langle\textcolor{blue}{X}\rangle$ is interpreted as the set of natural transformations between the functors $A\langle\textcolor{blue}{X}\rangle$ and $B\langle\textcolor{blue}{X}\rangle$. Now, if $A\langle\textcolor{blue}{X}\rangle$ is of the form $A_0 \Rightarrow \textcolor{blue}{X}$ (i.e. it is a *representable* functor), using the ε-theory we can deduce (see App. B) a "Yoneda lemma" in the form of the quantifier-eliminating isomorphism below:

$$\forall X.(A_0 \Rightarrow \textcolor{red}{X}) \Rightarrow B\langle\textcolor{blue}{X}\rangle \;\equiv\; B\langle\textcolor{blue}{X} \mapsto A_0\rangle \tag{1}$$

Similarly, if $A\langle\textcolor{red}{X}\rangle, B\langle\textcolor{red}{X}\rangle$ are both negative and $A\langle\textcolor{red}{X}\rangle$ is of the form $\textcolor{red}{X} \Rightarrow A_0$ (i.e. it is a *co-representable* functor), we can deduce another quantifier-eliminating isomorphism:

$$\forall X.(\textcolor{blue}{X} \Rightarrow A_0) \Rightarrow B\langle\textcolor{red}{X}\rangle \;\equiv\; B\langle\textcolor{red}{X} \mapsto A_0\rangle \tag{2}$$

Observe that both isomorphisms $(\star)$ from the Introduction are instances of (1) or (2).

[1] We define this theory formally in App. B, but this is not necessary to understand this paper.

As we admit more type-constructors in the language, we can use the ε-theory to deduce stronger schemas for eliminating quantifiers. For instance, using *least* and *greatest fixed points* $\mu X.A\langle X\rangle, \nu X.A\langle X\rangle$ of positive types, we can deduce the stronger schemas [35] below.

$$\forall X.(A\langle X\rangle \Rightarrow X) \Rightarrow B\langle X\rangle \equiv B\langle X \mapsto \mu X.A\langle X\rangle\rangle \tag{3}$$

$$\forall X.(X \Rightarrow A\langle X\rangle) \Rightarrow C\langle X\rangle \equiv C\langle X \mapsto \nu X.A\langle X\rangle\rangle \tag{4}$$

Note that (1) and (2) can be deduced from (3) and (4) using the isomorphisms $\mu X.A \equiv_{\beta\eta} \nu X.A \equiv_{\beta\eta} A$, when X does not occur in A. Moreover, adding sum and product types enables the elimination of the quantifier $\forall X$ also from a type of the form $A = \forall X.(A_{11}\langle X\rangle \Rightarrow A_{12}\langle X\rangle \Rightarrow X) \Rightarrow (A_{21}\langle X\rangle \Rightarrow A_{22}\langle X\rangle \Rightarrow X) \Rightarrow B\langle X\rangle$ by using $\beta\eta$-isomorphisms:

$$A \equiv_{\beta\eta} \forall X.\left(\left(\textstyle\sum_{i=1,2}\prod_{j=1,2} A_{ij}\langle X\rangle\right) \Rightarrow X\right) \Rightarrow B\langle X\rangle \equiv B\left\langle X \mapsto \mu X.\left(\textstyle\sum_{i=1,2}\prod_{j=1,2} A_{ij}\langle X\rangle\right)\right\rangle$$

These considerations lead to introduce the following class of isomorphisms:

▶ **Definition 1.** *A* Yoneda type isomorphism *is any instance of the schemas $\equiv_X$, $\equiv_X$ below*

$$\forall X.\forall\vec{Y}.\left\langle \forall\vec{Z}_k.\langle A_{jk}\langle X\rangle\rangle_j \Rightarrow X\right\rangle_k \Rightarrow B\langle X\rangle \quad \equiv_X \quad \forall\vec{Y}.B\left\langle X \mapsto \{\mu X.\}\textstyle\sum_k\left(\exists\vec{Z}_k.\prod_j A_{jk}\langle X\rangle\right)\right\rangle$$

$$\forall X.\forall\vec{Y}.\left\langle \forall\vec{Z}_k.X \Rightarrow A_j\langle X\rangle\right\rangle_k \Rightarrow B\langle X\rangle \quad \equiv_X \quad \forall\vec{Y}.B\left\langle X \mapsto \{\nu X.\}\forall\vec{Z}_k.\textstyle\prod_j A_j\langle X\rangle\right\rangle$$

where, given a list $L = \langle i_1, \dots, i_k\rangle$, and an L-indexed list of types $(A_i)_{i\in L}$, $\langle A_i\rangle_{i\in L} \Rightarrow B$ is a shorthand for $A_{i_1} \Rightarrow \dots \Rightarrow A_{i_k} \Rightarrow B$, the expressions $\{\mu X.\}$, $\{\nu X.\}$ indicate that the binder $\mu X.$ (resp. $\nu X.$) is applied only if X (resp. X) actually occurs in some of the A_{jk} (resp. A_j), and $\exists\vec{Y}.A$ is a shorthand for $\forall Y'.(\forall\vec{Y}.A \Rightarrow Y') \Rightarrow Y'$).

For all types A, B of $\mathsf{\Lambda 2p}_{\mu\nu}$, we write $A \equiv_{\mathsf{Y}} B$ when A can be converted to B using $\equiv_X, \equiv_X$ and the partial[2] *axiomatization of $\beta\eta$-isomorphisms in Fig. 9-11 (App. A).*

Since $\equiv_X$ and $\equiv_X$ are ε-isomorphisms (see App. B), whenever $A \equiv_{\mathsf{Y}} B$, A and B are interpreted as isomorphic objects in all dinatural and parametric models of $\mathsf{\Lambda 2}$.

3 Counting Type Inhabitants with Yoneda Type Isomorphisms

A first natural application of Yoneda type isomorphisms is to *count* the inhabitants of a *simple* type: given such a type A, with free variables $\vec{X}$, if $\forall\vec{X}.A \equiv_{\mathsf{Y}} 0 + 1 + \dots + 1 = \sum_{i=1}^{k} 1$, then A has exactly k proofs (up to ε-equivalence). Let us start with a “warm-up” example.

▶ **Example 2.** In [9] it is proved that the type $A = ((((X \Rightarrow Y) \Rightarrow X \Rightarrow Z) \Rightarrow (Y \Rightarrow Z) \Rightarrow W) \Rightarrow (Y \Rightarrow Z) \Rightarrow W$ has a unique inhabitant. Here’s a quick proof of $\forall XYZW.A \equiv_{\mathsf{Y}} 1$:

$$\begin{aligned}
&\forall XYZW.((((X \Rightarrow Y) \Rightarrow X \Rightarrow Z) \Rightarrow (Y \Rightarrow Z) \Rightarrow W) \Rightarrow (Y \Rightarrow Z) \Rightarrow W\\
&\equiv_W \forall XYZ.(Y \Rightarrow Z) \Rightarrow \Big(((X \Rightarrow Y) \Rightarrow X \Rightarrow Z) \times (Y \Rightarrow Z)\Big)\\
&\equiv_Z \forall XY.((X \Rightarrow Y) \Rightarrow X \Rightarrow Y) \times (Y \Rightarrow Y)\\
&\equiv_{\beta\eta} \Big(\forall XY.((X \Rightarrow Y) \Rightarrow X \Rightarrow Y)\Big) \times \Big(\forall Y.Y \Rightarrow Y\Big)\\
&\equiv_X \Big(\forall Y.Y \Rightarrow Y\Big) \times \Big(\forall Y.Y \Rightarrow Y\Big) \equiv_Y 1 \times 1 \equiv_{\beta\eta} 1
\end{aligned}$$

[2] The axiomatization in Fig. 9-11 is complete for the $\beta\eta$-isomorphisms of $\mathsf{\Lambda 2}$ [12], but fails to be complete (already at the propositional level) in presence of sums and the empty type [13, 18].

The literature on counting simple type inhabitants is vast (e.g. [2, 9, 8, 32]) and includes both complete algorithms and simpler *sufficient conditions* for a given type to have a unique or finite number of inhabitants. The latter provide then an ideal starting point to test our axiomatic theory of type isomorphisms, as several of these conditions are based on properties like the number of positive/negative occurrences of variables.

We tested Yoneda type isomorphisms on two well-known sufficient conditions for unique inhabitation. A simple type A is *balanced* when any variable occurring free in A occurs exactly once as X and exactly once as X. A is *negatively non-duplicated* if no variable occurs twice in A as X. An inhabited simple type which is balanced or negatively duplicated has exactly one inhabitant [2]. The theory $\equiv_{\mathsf{Y}}$ subsumes these conditions in the following sense:

▶ **Proposition 3.** *Let $A[X_1, \dots, X_n]$ be an inhabited simple type with free variables $X_1, \dots, X_n$. If $A[X_1, \dots, X_n]$ is either balanced or negatively non-duplicated, then $\forall X_1 \dots \forall X_n.A \equiv_{\mathsf{Y}} 1$.*

Observe that, since the type in Example 2 is neither balanced nor negatively non-duplicated, type isomorphisms provide a stronger condition than the two above.

We tested another well-known property, dual to one of the previous ones: a simple type A is *positively non-duplicated* if no variable occurs twice in A as X. A positively non-duplicated simple type has a finite number of proofs [9]. We reproved this fact using type isomorphisms, but this time only in a restricted case. Let the *depth* $d(A)$ of a simple type A be defined by $d(X) = 0$, $d(A \Rightarrow B) = \max\{d(A) + 1, d(B)\}$.

▶ **Proposition 4.** *Let $A[X_1, \dots, X_n]$ be an inhabited simple type with free variables $X_1, \dots, X_n$. If A is positively non-duplicated and $d(A) \leqslant 2$, then $\forall X_1 \dots \forall X_n.A \equiv_{\mathsf{Y}} 0 + 1 + \dots + 1$.*

4 From Polymorphic Types to Polynomial Trees

Read from left to right, the schemas $\equiv_{X}, \equiv_{X}$ yield rewriting rules over $\Lambda 2\mathsf{p}_{\mu\nu}$-types which *eliminate* occurrences of polymorphic quantifiers. Yet, a major obstacle to study this rewriting is that the application of $\equiv_{X}, \equiv_{X}$ might depend on the former application of $\beta\eta$-isomorphisms (as we did for instance in the previous section). Already for the propositional fragment $\Lambda\mathsf{p}$, the $\beta\eta$-isomorphisms are not finitely axiomatizable and it is not yet clear if a decision algorithm exists at all (see [13, 18]). This implies in particular that a *complete* criterion for the conversion of a $\Lambda 2$-type to a monomorphic (or even finite) type can hardly be computable.

For this reason, we restrict our goal to establishing some efficiently recognizable (in fact, polytime) *sufficient conditions* for quantifier-elimination. Moreover, we will exploit the well-known fact that the constructors $0, 1, +, \times, \mu, \nu$ can be encoded inside $\Lambda 2$ to describe our rewriting entirely within (a suitable representation of) $\Lambda 2$-types, for which $\beta\eta$-isomorphisms are completely axiomatized by the rules in Fig. 9 (see [12]).

Even if one restricts to $\Lambda 2$-types, recognizing if one of the schemas $\equiv_{X}, \equiv_{X}$ applies to a $\Lambda 2$-type $\forall X.A$ might still require to first apply some $\beta\eta$-isomorphisms. For example, consider the $\Lambda 2$-type $A = \forall X.((Y \Rightarrow X) \Rightarrow Y) \Rightarrow (Y \Rightarrow X) \Rightarrow Y$. In order to eliminate the quantifier $\forall X$ using $\equiv_{X}$, we first need to apply the $\beta\eta$-isomorphism $A \Rightarrow (B \Rightarrow C) \equiv_{\beta\eta} B \Rightarrow (A \Rightarrow C)$, turning A into $\forall X.(Y \Rightarrow X) \Rightarrow ((Y \Rightarrow X) \Rightarrow Y) \Rightarrow Y$, which is now of the form $\forall X.(B\langle X\rangle \Rightarrow X) \Rightarrow C\langle X\rangle$, with $B\langle X\rangle = Y$ and $C\langle X\rangle = ((Y \Rightarrow X) \Rightarrow Y) \Rightarrow Y$. We can then apply $\equiv_{X}$, yielding $((Y \Rightarrow Y) \Rightarrow Y) \Rightarrow Y$.

To obviate this problem, we introduce below a representation of $\Lambda 2$-types as labeled trees so that $\beta\eta$-isomorphic types are represented by the same tree. In the next section we will reformulate the schemas $\equiv_{X}, \equiv_{X}$ as reduction rules over such trees. This approach drastically simplifies the study of this rewriting, and will allow us to establish conditions for quantifier-elimination based on elementary graph-theoretic properties.

A $\beta\eta$-invariant representation of $\Lambda 2$-types. We introduce a representations of $\Lambda 2$-types as rooted trees whose leaves are labeled by *colored* variables, with colors being any $c \in \mathsf{Colors} = \{{\color{blue}\text{blue}}, {\color{red}\text{red}}\}$. We indicate such variables as either X^c or simply as ${\color{blue}X}, {\color{red}X}$. Moreover, we indicate as $\bar{c}$ the unique color different from c.

By a rooted tree we indicate a finite connected acyclic graph with a chosen vertex, called its root. If $(\mathscr{G}_i)_{i\in I}$ is a finite family of rooted trees, we indicate as $\begin{array}{c}\{\mathscr{G}_i\}_{i\in I}\\ | \\ u\end{array}$ the tree with root u obtained by adding an edge from any of the roots of the trees $\mathscr{G}_i$ to u.

▶ **Definition 5.** *The sets* ${\color{blue}\mathcal{E}}$ *and* ${\color{red}\mathcal{E}}$ *of* positive *and* negative $\Lambda 2$-trees *are inductively defined by:*

$$ {\color{blue}\mathsf{E}} \quad := \quad \begin{array}{ccc}\{{\color{red}\mathsf{E}_i}\}_{i\in I} & & {\color{blue}X}\\ & \diagdown\ \diagup & \\ & \vec{Y} & \end{array} \qquad\qquad {\color{red}\mathsf{E}} \quad := \quad \begin{array}{ccc}\{{\color{blue}\mathsf{E}_i}\}_{i\in I} & & {\color{red}X}\\ & \diagdown\ \diagup & \\ & \vec{Y} & \end{array} $$

where X is a variable, $\vec{Y}$ indicates a finite set of variables, and the edge in ${\color{blue}\mathsf{E}}$ *(resp.* ${\color{red}\mathsf{E}}$*) with label* ${\color{blue}X}$ *(resp.* ${\color{red}X}$*) is called the* head *of* ${\color{blue}\mathsf{E}}$ *(resp. of* ${\color{red}\mathsf{E}}$*). The trees* $\begin{array}{ccc}\{\,\}_\varnothing & & {\color{blue}X}\\ & \diagdown\ \diagup & \\ & \varnothing & \end{array}$ *and* $\begin{array}{ccc}\{\,\}_\varnothing & & {\color{red}X}\\ & \diagdown\ \diagup & \\ & \varnothing & \end{array}$ *are indicated simply as* ${\color{blue}X}$ *and* ${\color{red}X}$*.*

Free and *bound* variables of a tree $\mathsf{E} = \begin{array}{ccc}\{\mathsf{E}_i\}_{i\in I} & & X\\ & \diagdown\ \diagup & \\ & \vec{Y} & \end{array}$ are defined by $\mathsf{f}\mathcal{V}(\mathsf{E}) = \bigcup_{i=1}^n \mathsf{f}\mathcal{V}(\mathsf{E}_i) \cup \{X\} - \vec{Y}$ and $\mathsf{b}\mathcal{V}(\mathsf{E}) = \bigcup_{i=1}^n \mathsf{b}\mathcal{V}(\mathsf{E}_i) \cup \vec{Y}$.

We can associate a positive and a negative $\Lambda 2$-tree to any $\Lambda 2$-type as follows. Let us say that a type A of $\Lambda 2$ is *in normal form* (shortly, *in NF*) if $A = \forall \vec{Y}.A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow X$ where each of the variables in $\vec{Y}$ occurs in $A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow X$ at least once, and the types $A_1, \ldots, A_n$ are in normal form. It can be checked that any $\Lambda 2$-type is $\beta\eta$-isomorphic to a type in NF, that we indicate as $\mathsf{NF}(A)$.

▶ **Definition 6.** *For all $A \in \Lambda 2$, with $\mathsf{NF}(A) = \forall \vec{Y}.A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow X$, let*

$$ {\color{blue}\mathsf{t}}(A) = \begin{array}{ccc}\{{\color{red}\mathsf{t}}(A_i)\}_{i=1,\ldots,n} & & {\color{blue}X}\\ & \diagdown\ \diagup & \\ & \vec{Y} & \end{array} \qquad\qquad {\color{red}\mathsf{t}}(A) = \begin{array}{ccc}\{{\color{blue}\mathsf{t}}(A_i)\}_{i=1,\ldots,n} & & {\color{red}X}\\ & \diagdown\ \diagup & \\ & \vec{Y} & \end{array} $$

The tree-representation of $\Lambda 2$-types captures $\beta\eta$-isomorphism classes, in the sense that $A \equiv_{\beta\eta} B$ iff ${\color{blue}\mathsf{t}}(A) = {\color{blue}\mathsf{t}}(B)$ (this is proved in detail in [26]). For instance, the two $\beta\eta$-isomorphic types $\forall XY.({\color{blue}X} \Rightarrow {\color{red}X}) \Rightarrow (\forall Z.{\color{blue}X} \Rightarrow {\color{red}Z}) \Rightarrow ({\color{blue}Y} \Rightarrow {\color{red}X}) \Rightarrow {\color{blue}Y}$ and $\forall X.({\color{blue}X} \Rightarrow {\color{red}X}) \Rightarrow \forall Y.({\color{blue}Y} \Rightarrow {\color{red}X}) \Rightarrow ({\color{blue}X} \Rightarrow \forall Z.{\color{red}Z}) \Rightarrow {\color{blue}Y}$ translate into the same $\Lambda 2$-tree, shown in Fig. 4b (where underlined node labels and dashed edges can be ignored, for now).

Polynomial Trees. To formulate the schemas $\equiv_{\color{blue}X}, \equiv_{\color{red}X}$ in the language of rooted trees we exploit an encoding of the types of the form $\mu X.\exists \vec{Y}.\sum_{k\in K}\prod_{j\in J_k} A_{jk}\langle{\color{blue}X}\rangle$ and $\nu X.\forall \vec{Y}.\prod_{j\in J} A_j\langle{\color{blue}X}\rangle$ as certain special trees employing two new constants $\bullet$ and $\blacktriangle$. This encoding is easily seen to be a small variant, in the language of finite trees, of the usual second-order encodings.

We first introduce a handy notation for "polynomial" types, i.e. types corresponding to a generalized sum of generalized products. Following [14], any such type A is completely determined by a diagram of finite sets $I \xleftarrow{f} J \xrightarrow{g} K$ and a I-indexed family of types $(A_i)_{i\in I}$, so that $A = \sum_{k\in K}\prod_{j\in J_k} A_{f(j)}$, where $J_k := g^{-1}(k)$. In the following, we will call the given of a finite diagram $I \xleftarrow{f} J \xrightarrow{g} K$, and an I-indexed family $(a_i)_{i\in I}$ a *polynomial family*, and indicate it simply as $(a_{jk})_{k\in K, j\in J_k}$ (in fact, we already implicitly used this notation in Def. 1).

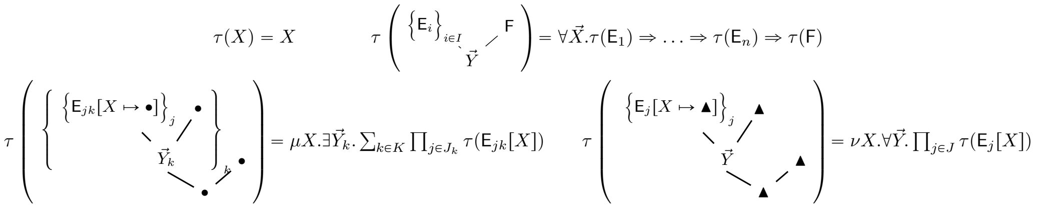
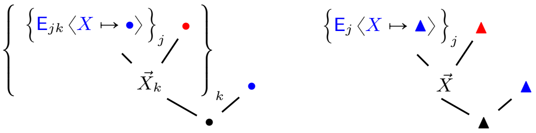

$\tau(X) = X$
τ
$\{\mathsf{E}_i\}_{i\in I}$
F
$\vec{Y}$
$= \forall\vec{X}.\tau(\mathsf{E}_1) \Rightarrow \dots \Rightarrow \tau(\mathsf{E}_n) \Rightarrow \tau(\mathsf{F})$
$\{\mathsf{E}_{jk}[X \mapsto \bullet]\}_j$
$\vec{Y}_k$
k
$= \mu X.\exists\vec{Y}_k.\sum_{k\in K}\prod_{j\in J_k}\tau(\mathsf{E}_{jk}[X])$
$\{\mathsf{E}_j[X \mapsto \blacktriangle]\}_j$
$\vec{Y}$
$= \nu X.\forall\vec{Y}.\prod_{j\in J}\tau(\mathsf{E}_j[X])$

Figure 3 Translation of simple polynomial trees into monomorphic types.

We now enrich the class of Λ2-trees as follows:

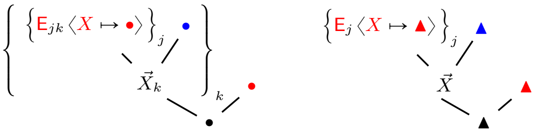
▶ **Definition 7** (Polynomial trees). *Let •, ▲ indicate two new constants.*

- *the set $\mathcal{P}$ of* positive polynomial trees *is defined by adding to the clauses defining positive Λ2-trees two new clauses:*


$\{\mathsf{E}_{jk}\langle X \mapsto \bullet\rangle\}_j$
$\vec{X}_k$
k
$\{\mathsf{E}_j\langle X \mapsto \blacktriangle\rangle\}_j$
$\vec{X}$

where X is some variable, $(\mathsf{E}_{jk}\langle X\rangle)_{k,j}$ (resp. $(\mathsf{E}_j\langle X\rangle)_j$) is a polynomial family (resp. a family) of positive polynomial trees with no occurrence of X, and $(\vec{X}_k)_{k\in K}$ is a K-indexed family of finite sets of variables.

- *the set $\mathcal{P}$ of* negative polynomial trees *is defined by adding to the clauses defining negative Λ2-trees two new clauses:*


$\{\mathsf{E}_{jk}\langle X \mapsto \bullet\rangle\}_j$
$\vec{X}_k$
k
$\{\mathsf{E}_j\langle X \mapsto \blacktriangle\rangle\}_j$
$\vec{X}$

where X is some variable, $(\mathsf{E}_{jk}\langle X\rangle)_{k,j}$ (resp. $(\mathsf{E}_j\langle X\rangle)_j$) is a polynomial family (resp. a family) of negative polynomial trees with no occurrence of X, and $(\vec{X}_k)_{k\in K}$ is a K-indexed family of finite sets of variables.

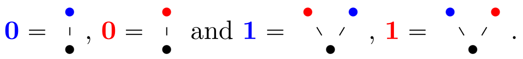
We indicate by $\mathcal{P}$ the set of all polynomial trees, and by $\mathcal{P}_0$ the set of all polynomial trees with no bound variables, which are called simple.

Any polynomial tree $\mathsf{E} \in \mathcal{P}$ can be converted into a type $\tau(\mathsf{E})$ of $\Lambda 2\mathsf{p}_{\mu\nu}$ as illustrated in Fig. 3. It is easily checked that, whenever E is simple, $\tau(\mathsf{E})$ has no quantifier. Moreover, one can check that for all Λ2-type, $\tau(\mathsf{t}(A)) = A$.

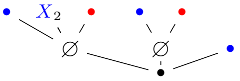
We conclude this section with some basic example of polynomial trees.

▶ **Example 8.**

- The constant types 0 and 1 are represented as positive/negative trees by

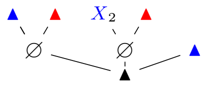


- The diagram $\{1,2\} \overset{1,3\mapsto 1;2\mapsto 2}{\longleftarrow} \{1,2,3\} \overset{1,2\mapsto 1;3\mapsto 2}{\longrightarrow} \{1,2\}$, along with the family $(X_i)_{i\in\{1,2\}}$, yields the polynomial family $(\mathsf{E}_{jk})_{k,j}$, with $\mathsf{E}_{11} = \mathsf{E}_{32} = X_1$ and $\mathsf{E}_{21} = X_2$, and yields the polynomial tree

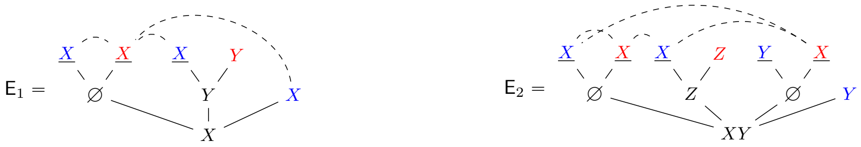


encoding the type $\mu X_1.(X_1 \times X_2) + X_1$.

- The diagram $\{1,2\} \overset{id}{\leftarrow} \{1,2\} \rightarrow \{1\}$ with the same family as above yields the polynomial tree



encoding the type $\nu X_1.X_1 \times X_2$.



(a) $\forall X.(X \Rightarrow X) \Rightarrow (\forall Y.X \Rightarrow Y) \Rightarrow X.$ **(b)** $\forall XY.(X \Rightarrow X) \Rightarrow (\forall Z.X \Rightarrow Z) \Rightarrow (Y \Rightarrow X) \Rightarrow Y.$

Figure 4 Polynomial trees with highlighted modular nodes and modular pairs.

5 Yoneda Reduction

In this section we introduce a family of rewriting rules $\rightsquigarrow_X, \rightsquigarrow_X$ over polynomial trees, that we call *Yoneda reduction*, which correspond to the left-to-right orientation of the isomorphisms $\equiv_X, \equiv_X$. We will adopt the following conventions:

▶ **Notation 5.1.** *We make the assumption that all bound variables of a polynomial tree* E *are distinct. More precisely, for any* $X \in \mathsf{b}\mathcal{V}(\mathsf{E})$*, we suppose there exist unique nodes* r_X *and* h_X *such that* $\mathsf{r}_X : \vec{X}$*, for some set of variable* $\vec{X}$ *such that* $X \in \vec{X}$*, and* h_X *is the head of the sub-tree whose root is* r_X*.*

We will call two distinct nodes *parallel* if they are immediate successors of the same node, and we let the distance $d(\alpha, \beta)$ between two nodes in a polynomial tree be the number of edges of the unique path from α to β.

Using polynomial trees we can identify when a quantifier can be eliminated from a type independently from $\beta\eta$-isomorphisms, by inspecting a simple condition on the tree-representation of the type based on the notion of *modular* node, introduced below.

▶ **Definition 9.** *For all* $X \in \mathsf{b}\mathcal{V}(\mathsf{E})$*, a terminal node* $\alpha : X^c$ *in* E *is said* modular *if* $\alpha \neq \mathsf{h}_X$*,* $1 \leqslant d(\alpha, \mathsf{r}_X) \leqslant 2$ *and* α *has no parallel node of label* X^c*. A pair of nodes of the form* $(\alpha : X, \beta : X)$ *is called a* X-pair*, and a* X*-pair is said* modular *if one of its nodes is modular.*

In the trees in Fig. 4 the modular nodes are underlined and the modular pairs are indicated as dashed edges.

▶ **Definition 10.** *A variable* $X \in \mathsf{b}\mathcal{V}(\mathsf{E})$ *is said* eliminable *when every* X*-pair of* E *is modular. For every color* c*, we furthermore call* X c-eliminable *if every node* $\alpha : X^{\overline{c}}$ *is modular.*

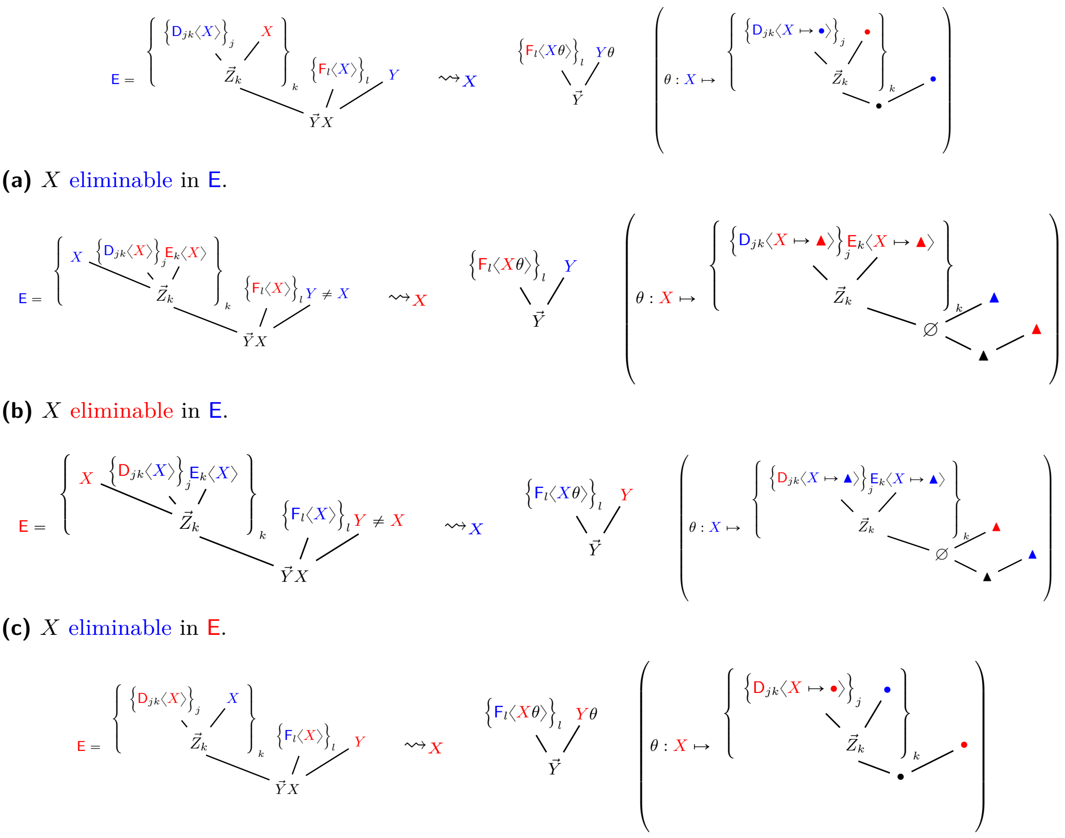
We let eliminable be a shorthand for “blue-eliminable” and eliminable be a shorthand for “red-eliminable”. These notions are related as follows:

▶ **Lemma 11.** *X is eliminable iff it is either* eliminable *or* eliminable*.*

Proof. If X is neither eliminable nor eliminable, then there exist non-modular nodes $\alpha : X, \beta : X$, whence the X-pair (α, β) is not modular. Conversely, suppose X is eliminable but not eliminable. Hence there is a non modular node $\alpha : X$. For all node $\beta : X$, since the X-pair (α, β) is modular, β is modular. We deduce that X is eliminable. ◀

▶ **Example 12.** The variable X is eliminable but not eliminable in the tree in Fig. 4a, and it is both eliminable and eliminable in the tree in Fig. 4b.

The proposition below shows that a variable X is eliminable (resp. eliminable) in the tree of a $\Lambda 2$-type $\forall X.A$ exactly when $\forall X.A$ matches, up to $\beta\eta$-isomorphisms, with the left-hand type of the schema $\equiv_X$ (resp. $\equiv_X$).


E =
$\{D_{jk}\langle X\rangle\}_j$
$\vec{Z}_k$
$\{F_l\langle X\rangle\}_l$
$\vec{Y}X$
$\rightsquigarrow_X$
$\{F_l\langle X\theta\rangle\}_l$
$\vec{Y}$
$\theta : X \mapsto$
$\{D_{jk}\langle X \mapsto \bullet\rangle\}_j$
$\{E_k\langle X\rangle\}$
$Y \neq X$
$\{D_{jk}\langle X \mapsto \blacktriangle\rangle\}_j$
$E_k\langle X \mapsto \blacktriangle\rangle$
$\varnothing$
$Y\theta$

(a) X eliminable in E.

(b) X eliminable in E.

(c) X eliminable in E.

(d) X eliminable in E.

Figure 5 Yoneda reduction of X-eliminable trees.

▶ **Proposition 13.**
- *X is eliminable in E iff E is as in Fig. 5a left, for some polynomial family $(D_{jk}\langle X\rangle)_{k\in K, j\in J_k}$, family $(F_l\langle X\rangle)_{l\in L}$ and blue variable Y (possibly X itself).*
- *X is eliminable in E iff E is as in Fig. 5b left, for some polynomial family $(D_{jk}\langle X\rangle)_{k\in K, j\in J_k}$, family $(E_k\langle X\rangle)_{k\in K}$, family $(F_l\langle X\rangle)_{l\in L}$ and blue variable $Y \neq X$.*
- *X is eliminable in E iff E is as in Fig. 5c left, for some polynomial family $(D_{jk}\langle X\rangle)_{k\in K, j\in J_k}$, family $(E_k\langle X\rangle)_{k\in K}$, family $(F_l\langle X\rangle)_{l\in L}$ and red variable $Y \neq X$.*
- *X is eliminable in E iff E is as in Fig. 5d left, for some polynomial family $(D_{jk}\langle X\rangle)_{k\in K, j\in J_k}$, family $(F_l\langle X\rangle)_{l\in L}$ and red variable Y (possibly X itself).*

For all four cases of Prop. 13 we define a rewriting rule which eliminates X.

▶ **Definition 14** (Yoneda reduction). *Let $\mathsf{F} \in \mathcal{P}$ and $X \in \mathsf{b}\mathcal{V}(\mathsf{F})$ be an eliminable variable. The rules $\mathsf{F} \rightsquigarrow_{X^c} \mathsf{F}'$ consist in replacing the subtree E of F rooted in r_X as illustrated in Fig. 5.*

▶ **Example 15.** The tree in Fig. 4b rewrites as illustrated in Fig. 6.

By inspecting the rules in Fig. 5 one can check that $\mathsf{E} \rightsquigarrow_{X^c} \mathsf{E}'$ implies $\tau(\mathsf{E}) \equiv_{\mathsf{Y}} \tau(\mathsf{E}')$. From this we can deduce by induction:

▶ **Lemma 16.** *For all $\Lambda 2$-type A, if $\mathsf{t}(A) \rightsquigarrow^* \mathsf{E} \in \mathcal{P}_0$, then $A \equiv_{\mathsf{Y}} \tau(\mathsf{E}) \in \Lambda\mathsf{p}_{\mu\nu}$.*

The lemma above suggests to study the elimination of quantifiers from $\Lambda 2$-types by studying the convergence of $\Lambda 2$-trees onto simple polynomial trees. This will be our next goal.

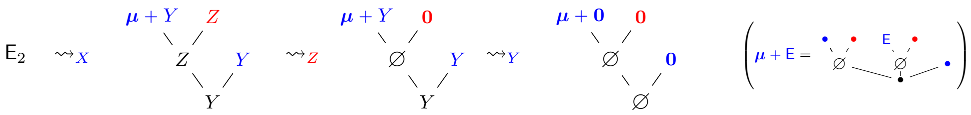


Figure 6 Yoneda reduction of the polynomial tree E_2 from Fig. 4b.

6 The Characteristic of a Polymorphic Type

In this section we exploit Yoneda reduction to establish two sufficient conditions to convert a $\Lambda2$-type A into a quantifier-free type A' such that $A \equiv_{\mathsf{Y}} A'$.

The Coherence Condition. When a reduction is applied to E, several sub-trees of E can be either erased or copied and moved elsewhere. Hence, the resulting tree E' might well have a greater size and even a larger number of bound variables than E. Nevertheless, sequences of Yoneda reductions always terminate, as one can define a measure which decreases at each step (see [26]).

▶ **Proposition 17.** *There is no infinite sequence of Yoneda reductions.*

Although sequences of reductions always terminate, they need not terminate on a simple polynomial tree, that is, on the encoding of a monomorphic type. This can be due to several reasons. Firstly, one bound variable might not be eliminable. Secondly, even if all variables are eliminable, this property need not be preserved by reduction. For example, take the type $A = \forall X.\forall Y.(X \Rightarrow Y \Rightarrow X) \Rightarrow ((Y \Rightarrow X) \Rightarrow W) \Rightarrow Z$: although X and Y are both eliminable (in the associated tree), if we apply a reduction to X, then Y ceases to be eliminable, and similarly if we reduce Y first. Such conflicts can be controlled by imposing a suitable *coherence* relation on variables.

▶ **Definition 18.** *Let* E *be a polynomial tree,* $X, Y \in \mathsf{b}\mathcal{V}(\mathsf{E})$ *and* $c, d \in \mathsf{Colors}$*.* X^c *and* Y^d *are said* coherent *if there exists no parallel modular nodes of the form* $\alpha : X^{\overline{c}}, \beta : Y^{\overline{d}}$ *in* E*.*

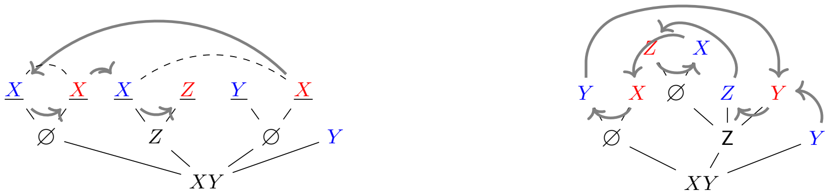
▶ **Example 19.** In the tree in Fig. 4b, Y and Z are coherent, while X and Y are not.

▶ **Definition 20** (coherence condition)**.** *Let* E *be a polynomial tree. A* valuation *of* E *is any map* $\phi : \mathsf{b}\mathcal{V}(\mathsf{E}) \to \mathsf{Colors}$*. For all valuation* ϕ *of* E*, we call* E ϕ-coherent *if for all* $X \in \mathsf{b}\mathcal{V}(\mathsf{E})$*,* X *is* $\phi(X)$*-eliminable, and moreover for all* $Y \neq X \in \mathsf{b}\mathcal{V}(\mathsf{E})$*,* $X^{\phi(X)}$ *is coherent with* $Y^{\phi(Y)}$*. We call* E coherent *if it is* ϕ*-coherent for some valuation* ϕ *of* E*.*

▶ Remark 21 (Coherence is an instance of 2-SAT)**.** The problem of checking if a polynomial tree E is coherent can be formulated as an instance of 2-SAT (a well-known polytime problem): consider n Boolean variables $x_1, \dots, x_n$ (one for each bound variable of E), and let a_i^c be x_i if $c =$ blue and $\neg x_i$ if $c =$ red. Consider then the 2-CNF $A \wedge B$, where A is the conjunction of all $a_i^c \vee a_i^c$ such that X_i is not $\overline{c}$-eliminable in E, and B is the conjunction of all $a_i^c \vee a_j^d$, for all incoherent $X_i^{\overline{c}}$ and $X_j^{\overline{d}}$. Then a coherent valuation of E is the same as a model of $A \wedge B$.

As observed before, a reduction $\mathsf{E} \rightsquigarrow_X \mathsf{E}'$ might copy or erase some bound variables of E. One can then define a map $g : \mathsf{b}\mathcal{V}(\mathsf{E}') \to \mathsf{b}\mathcal{V}(\mathsf{E})$ associating any variable in E' with the corresponding variable in E of which it is a copy. A sequence of reductions $\mathsf{E}_0 \rightsquigarrow_{X_1^{c_1}} \cdots \rightsquigarrow_{X_n^{c_n}} \mathsf{E}_n$ induces then, for $1 \leqslant i \leqslant n$, maps $g_i : \mathsf{b}\mathcal{V}(\mathsf{E}_i) \to \mathsf{b}\mathcal{V}(\mathsf{E}_{i-1})$, and we let $G_i : \mathsf{b}\mathcal{V}(\mathsf{E}_i) \to \mathsf{b}\mathcal{V}(\mathsf{E}_0)$ be $g_1 \circ g_2 \circ \cdots \circ g_i$.

The rewriting properties of coherent trees are captured by the following notion:



(a) Alternate path in $\forall XY.(X \Rightarrow X) \Rightarrow (\forall Z.X \Rightarrow Z) \Rightarrow (Y \Rightarrow X) \Rightarrow Y$.

(b) Cyclic alternate path in $\forall XY.(Y \Rightarrow X) \Rightarrow (\forall Z.(Z \Rightarrow X) \Rightarrow (Z \Rightarrow Y)) \Rightarrow Y$.

Figure 7 Examples of alternate paths.

▶ **Definition 22** (standard reduction). *A sequence of reductions* $\mathsf{E}_0 \rightsquigarrow_{X_1^{c_1}} \cdots \rightsquigarrow_{X_n^{c_n}} \mathsf{E}_n$ *is said* standard *if for all* $i, j = 1, \dots, n$, $G_i(X_i)^{c_i}$ *is coherent with* $G_j(X_j)^{c_j}$ *in* E_0. *We say that* E strongly converges under standard reduction *if all standard reductions starting from* E *terminate on a simple polynomial tree.*

Using the fact that coherence is stable under standard reduction (see [26]) we obtain:

▶ **Theorem 23.** E *is coherent iff* E *strongly converges under standard reduction.*

Using Lemma 16 we further deduce:

▶ **Corollary 24.** *Let* A *be a type of* $\Lambda 2$. *If* $\mathsf{t}(A)$ *is coherent, then there exists a* $\Lambda\mathsf{p}_{\mu\nu}$*-type* A' *such that* $A \equiv_{\mathsf{Y}} A'$.

The Characteristic. We now introduce a refined condition for coherent trees, which can be used to predict whether a type rewrites into a finite type (i.e. one made up from $0, 1, +, \times, \Rightarrow$ only) or into one using μ, ν-constructors.

An intuition from Section 2 is that, for a type of the form $\forall X.(A\langle X\rangle \Rightarrow X) \Rightarrow B\langle X\rangle$ (which rewrites into $B\langle X \mapsto \mu X.A\langle X\rangle\rangle$) to reduce to one without μ, ν-types, the variable X *must not occur in* A at all. However, the property "X does not occur in A" need not be preserved under reduction. Instead, we will define a stronger condition that is preserved by standard reduction by inspecting a class of *paths* in the tree of a type.

▶ **Definition 25.** *Let* E *be a polynomial* $\Lambda 2$*-tree and let* $\preceq$ *indicate the natural order on the nodes of* E *having the root of* E *as its minimum. A* down-move *in* E *is a pair* $\alpha \rhd \beta$, *such that, for some bound variable* $X \in \mathsf{b}\mathcal{V}(\mathsf{E})$, (α, β) *is an* X*-pair and* β *is modular. An* up-move *in* E *is a pair* $\alpha \rhd \beta$ *such that* $\alpha \neq \beta$, α *is a modular node with immediate predecessor* γ, $\beta : X$ *for some* $X \in \mathsf{b}\mathcal{V}(\mathsf{E})$, *and* $\gamma \preceq \beta$. *An* alternating path *in* E *is a sequence of nodes* $\alpha_0 \dots \alpha_{2n}$ *such that* $\alpha_{2i} \rhd \alpha_{2i+1}$ *is a down-move and* $\alpha_{2i+1} \rhd \alpha_{2i+2}$ *is an up-move.*

In Fig. 7b are illustrated some alternating paths. Observe that whenever X occurs in $A\langle X\rangle$, we can construct a *cyclic* alternate path in the tree of $\forall X.(A\langle X\rangle \Rightarrow X) \Rightarrow B\langle X\rangle$: down-move from an occurrence of X in A to the modular node labeled X, then up-move back to X. We deduce that if no cyclic alternate path exists, then any subtype of the form above must be such that X does not occur in A. This leads to introduce the following:

▶ **Definition 26.** *For any polynomial tree* E, *the* characteristic of E, $\kappa(\mathsf{E}) \in \{0, 1, \infty\}$ *is defined as follows: if* E *is coherent, then* $\kappa(\mathsf{E}) = 0$ *if it has no cyclic alternating path, and* $\kappa(\mathsf{E}) = 1$ *if it has a cyclic alternating path; if* E *is not coherent,* $\kappa(\mathsf{E}) = \infty$.

The characteristic is indeed stable under standard reduction (see [26]).

▶ **Lemma 27.** *For all* E, E'*, if* E *reduces to* E' *by standard reduction, then* $\kappa(\mathsf{E}') \leqslant \kappa(\mathsf{E})$*.*

Using the observation above and Lemma 27 we can easily prove:

▶ **Proposition 28.** *Suppose* $\kappa(\mathsf{E}) \in \{0,1\}$ *and* E *reduces to* $\mathsf{E}' \in \mathcal{P}_0$ *by standard reduction. If* $\kappa(\mathsf{E}) = 0$*, then* $\tau(\mathsf{E}') \in \Lambda\mathsf{p}$*, and if* $\kappa(\mathsf{E}) = 1$*, then* $\tau(\mathsf{E}') \in \Lambda\mathsf{p}_{\mu\nu}$*.*

For a $\Lambda 2$-type A, we can define its characteristic as $\kappa(A) = \kappa(\mathsf{t}(A))$. From Prop. 28 and Lemma 16 we deduce then a new criterion for finiteness:

▶ **Corollary 29.** *Let* A *be a closed* $\Lambda 2$*-type. If* $\kappa(A) = 0$*, then* $A \equiv_{\mathsf{Y}} 0 + 1 + \cdots + 1$*.*

The criterion based on the characteristic can be used to capture yet more finite $\Lambda 2$-types. In fact, whenever a type A reduces to a type with characteristic 0, we can deduce that $A \equiv_{\mathsf{Y}} 0 + 1 + \cdots + 1$. Note that such a type A need not even be coherent. For instance, the (tree of) the type $A = \forall XY.(\forall Z.((Z \Rightarrow Z) \Rightarrow X) \Rightarrow X) \Rightarrow Y \Rightarrow Y$, is not coherent (since the variable Z is not eliminable), but reduces, by eliminating X, to (the tree of) $\forall Y.Y \Rightarrow Y$, which has characteristic 0, and in fact we have that $A \equiv_{\mathsf{Y}} 1$.

As this example shows, a type A can reduce to a finite sum $0 + 1 + \cdots + 1$ even if some of its subtypes cannot be similarly reduced. In fact, while we can eliminate all quantifiers from the type A above, we cannot do this from its subtype $\forall Z.((Z \Rightarrow Z) \Rightarrow X) \Rightarrow X)$. By contrast, in the next section we show that the characteristic satisfies nice compositionality conditions that will allow us to define suitable fragments of $\Lambda 2$.

7 System F with Finite Characteristic

In this section we explore the use of Yoneda reduction to compute program equivalence in $\Lambda 2$. We introduce two fragments of $\Lambda 2$ in which types have a fixed finite characteristic, and we show that the ε-theory for such fragment can be computed by embedding polymorphic programs into well-known monomorphic systems.

First, we have to check that the types with a fixed finite characteristic do yield well-defined fragments of $\Lambda 2$. This requires to check two properties. First, the characteristic has to be *compositional*: a subtype of a type of characteristic k cannot have a higher characteristic, since every subtype of a type of the fragment must be in the fragment itself. Second, since a universally quantified variable can be instantiated with any other type of the fragment, the characteristic must be *closed by instantiation*: if $\forall X.A$ and B have characteristic k, then $A[B/X]$ must have characteristic (at most) k.

▶ **Lemma 30.** ***(compositionality)*** *If* A *is a sub-type of* B*, then* $\kappa(A) \leqslant \kappa(B)$*.*
(closure by instantiation) $\kappa(A[B/X]) \leqslant \max\{\kappa(\forall X.A), \kappa(B)\}$.

Thanks to Lemma 30 the following fragments can be seen to be well-defined.

▶ **Definition 31** (Systems $\Lambda 2^{\kappa \leqslant k}$)**.** *For* $k = 0, 1$*, let* $\Lambda 2^{\kappa \leqslant k}$ *be the subsystem of* $\Lambda 2$ *with same typing rules and types restricted to the types of* $\Lambda 2$ *of characteristic* k*.*

We recall that the *free bicartesian closed category* is the category $\mathbb{B} = \mathbb{C}_{\beta\eta}(\Lambda\mathsf{p})$ and the *free cartesian closed* μ*-bicomplete category* $\mu\mathbb{B}$ [29, 6] is the category $\mathbb{C}_{\beta\eta}(\Lambda\mathsf{p}_{\mu\nu})$. The β and η-rules for $\Lambda\mathsf{p}$ and $\Lambda\mathsf{p}_{\mu\nu}$ are recalled in App. A.

$\mathbb{B}$ and $\mu\mathbb{B}$ can be embedded in $\Lambda 2$ by the usual second order encoding (that we note $\sharp$ and recall in [26]). In fact, it is easily seen that any type of $\Lambda\mathsf{p}$ (resp. of $\Lambda\mathsf{p}_{\mu\nu}$) is encoded by a type of $\Lambda 2^{\kappa \leqslant 0}$ (resp. of $\Lambda 2^{\kappa \leqslant 1}$). Moreover, it is well-known (see [28, 16]) that the η-rules of

$\Lambda\mathsf{p}_{\mu\nu}$ are preserved in $\Lambda 2$ only up to dinaturality (i.e. up to the ε-theory). This embedding yields then a functor ${}^{\sharp} : \mu\mathbb{B} \to \mathbb{C}_{\varepsilon}(\Lambda 2^{\kappa\leqslant 1})$, which restricts to a functor from $\mathbb{B}$ to $\mathbb{C}_{\varepsilon}(\Lambda 2^{\kappa\leqslant 0})$.

Conversely, from Proposition 28, we already know that the types of $\Lambda 2^{\kappa\leqslant 0}$ (resp. $\Lambda 2^{\kappa\leqslant 1}$) are isomorphic, modulo the ε-theory, to types of $\Lambda\mathsf{p}$ (resp. $\Lambda\mathsf{p}_{\mu\nu}$).

▶ **Proposition 32.** *For all $\Lambda 2^{\kappa\leqslant 0}$-type (resp. $\Lambda 2^{\kappa\leqslant 1}$-type) A there exists a type $A^{\flat}$ of $\Lambda\mathsf{p}$ (resp. $\Lambda\mathsf{p}_{\mu\nu}$) such that $A \equiv_{\varepsilon} A^{\flat}$.*

Proof. Since all isomorphisms $\equiv_{\mathsf{Y}}$ are valid under the ε-theory, we can obtain $A^{\flat}$ from any standard reduction of the tree of A, using Prop. 28. For technical reasons we will consider a particular reduction, described in detail in [26]. ◀

In [26] we show that the embedding of types above scales to an embedding of terms: for all term t such that $\Gamma \vdash_{\Lambda 2^{\kappa\leqslant 1}} t : A$, we define a term $t^{\flat}$ such that $\Gamma^{\flat} \vdash_{\Lambda\mathsf{p}_{\mu\nu}} t^{\flat} : A^{\flat}$ holds (with the construction scaling well to $\Lambda 2^{\kappa\leqslant 0}$ and $\Lambda\mathsf{p}$). This yields then a functor ${}^{\flat} : \mathbb{C}_{\varepsilon}(\Lambda 2^{\kappa\leqslant 1}) \to \mu\mathbb{B}$. restricting to a functor from $\mathbb{C}_{\varepsilon}(\Lambda 2^{\kappa\leqslant 0})$ to $\mathbb{B}$.

The two functors ${}^{\sharp}$ and ${}^{\flat}$ preserve all the relevant structure (products, coproducts, exponentials, initial algebras, final coalgebras), but they are not *strictly* inverse: $(A^{\flat})^{\sharp}$ is not equal to A, but only ε-isomorphic to it (e.g. for $A = \forall X.((\forall Y.Y \Rightarrow Y) \Rightarrow X) \Rightarrow X$, we have $A^{\flat} = 1$ and $(A^{\flat})^{\sharp} = \forall X.X \Rightarrow X$). Nevertheless, the following equivalences of categories hold:

▶ **Theorem 33.** *$\mathbb{C}_{\varepsilon}(\Lambda 2^{\kappa\leqslant 0}) \simeq \mathbb{B}$, $\mathbb{C}_{\varepsilon}(\Lambda 2^{\kappa\leqslant 1}) \simeq \mu\mathbb{B}$.*

The proof of Theorem 33 (in [26]) is done by checking (by way of β-, η- and ε-*rules*, see App. B) that both $\Lambda 2^{\kappa\leqslant 1}$ and $\Lambda\mathsf{p}_{\mu\nu}$ embed *fully* in a suitable fragment of $\Lambda 2\mathsf{p}_{\mu\nu}$, using the lemma below (with $\mathbb{C} = \mathbb{C}_{\varepsilon}(\Lambda 2^{\kappa\leqslant 1})$, $\mathbb{D} = \mu\mathbb{B}$ and $f(A) = A^{\flat}$):

▶ **Lemma 34.** *Let $\mathbb{C}, \mathbb{D}$ be full subcategories of a category $\mathbb{E}$. Let $f : \mathsf{Ob}(\mathbb{C}) \to \mathsf{Ob}(\mathbb{D})$ be surjective and suppose there is a map u associating each object a of $\mathbb{C}$ with an isomorphism $u_a : a \to f(a)$ in $\mathbb{E}$. Then f extends to an equivalence of categories $F : \mathbb{C} \to \mathbb{D}$.*

Theorem 33 can be used to deduce properties of program equivalence in $\Lambda 2^{\kappa\leqslant 0}$ and $\Lambda 2^{\kappa\leqslant 1}$ from well-known properties of program equivalence for $\mathbb{B}$ and $\mu\mathbb{B}$. In fact, it is known that $\beta\eta$-equivalence in $\Lambda\mathsf{p}$ (i.e. arrow equivalence in $\mathbb{B}$) is decidable and coincides with contextual equivalence [31], while contextual equivalence for $\mu\mathbb{B}$ is undecidable [6]. Using Theorem 33 we can deduce similar facts for $\Lambda 2^{\kappa\leqslant 0}$ and $\Lambda 2^{\kappa\leqslant 1}$:

▶ **Theorem 35.** *The ε-theory for $\Lambda 2^{\kappa\leqslant 0}$ is decidable and coincides with contextual equivalence. Both the ε-theory and contextual equivalence for $\Lambda 2^{\kappa\leqslant 1}$ are undecidable.*

The first claim of Theorem 35 is proved by defining a new embedding $t \mapsto t^{\natural}$ of $\Lambda 2^{\kappa\leqslant 0}$ into $\Lambda\mathsf{p}$, exploiting the well-known fact that for the terms of the system $\Lambda 2\mathsf{p}$ one can obtain a normal form under β-reduction and *commutative conversions* [33][3]. Using the isomorphisms $\mathsf{d}_A[x], \mathsf{d}_A^{-1}[x]$ between A and $A^{\flat}$, a term t such that $\Gamma \vdash_{\Lambda 2^{\kappa\leqslant 0}} t : A$ holds is first translated into $u = \mathsf{d}_A[t[x_i \mapsto \mathsf{d}_{A_i}^{-1}[x_i]]]$ (where $\Gamma = x_1 : A_1, \dots, x_n : A_n$), and then $t^{\natural}$ is defined as the normal form of u. From the fact that $\Gamma^{\flat} \vdash_{\Lambda 2\mathsf{p}} t^{\natural} : A^{\flat}$ and that $t^{\natural}$ is in normal form, we can deduce that $\Gamma^{\flat} \vdash_{\Lambda\mathsf{p}} t^{\natural} : A^{\flat}$ holds, so the embedding is well-defined.

Using the embedding $t \mapsto t^{\natural}$ we establish the proposition below, from which the first claim of Theorem 35 descends (since $\simeq_{\beta\eta}$ and $\simeq_{\mathsf{ctx}}$ coincide and are decidable in $\Lambda\mathsf{p}$).

[3] Actually, [33] does not consider commuting conversions for 0, but these can be added without altering the existence of normal forms.

▶ **Proposition 36.** $\Gamma \vdash_{\Lambda 2^{\kappa \leqslant 0}} t \simeq_{\varepsilon} u : A$ *iff* $\Gamma \vdash_{\Lambda 2^{\kappa \leqslant 0}} t \simeq_{\mathsf{ctx}} u : A$ *iff* $\Gamma^{\flat} \vdash_{\Lambda \mathsf{p}} t^{\natural} \simeq_{\beta\eta} u^{\natural} : A^{\flat}$.

For the second claim of Theorem 35, the undecidability of contextual equivalence in $\Lambda 2^{\kappa \leqslant 1}$ immediately follows from its undecidability in $\mu\mathbb{B}$ by the encoding $^{\sharp}$. Instead, we do not know if the ε-theory and contextual equivalence coincide in this case, as it is not clear whether the embedding $t \mapsto t^{\natural}$ scales to $\Lambda 2^{\kappa \leqslant 1}$: this depends on the existence of normal forms for commutative conversions, which are not known to hold in presence of μ, ν-types (although this is conjectured in [20]).

The undecidability of the ε-theory is proved in [26] following a different strategy: we first observe that the ε-rules for $\Lambda 2^{\kappa \leqslant 1}$ imply some *uniqueness* rule for inductively/co-inductively defined functions. For instance, under the ε-theory, the usual type of natural numbers $\mathtt{int} = \forall X.(X \Rightarrow X) \Rightarrow (X \Rightarrow X)$ is associated with a uniqueness rule for functions defined by iteration of the following form: for any type C and functions $f : \mathtt{int} \Rightarrow C$ and $h : C \Rightarrow C$, if $f(x+1) \simeq h(f(x))$, then f is equivalent to the function defined by iterating h on $f(0)$, i.e. $f \simeq \lambda x.xCh(f(0))$. Similarly, the type of the usual recursor of $\Lambda 2$ is associated with a uniqueness rule for functions defined by recursion.

It is well-known that any equational theory over a system containing the simply typed λ-calculus, a type of natural numbers and a recursion operator, and satisfying a uniqueness rule for recursively defined functions as above, is undecidable [22]. It suffices then to check that $\Lambda 2^{\kappa \leqslant 1}$, under the ε-theory, provides such a system.

8 Conclusion

Related Work. The connection between parametricity, dinaturality and the Yoneda isomorphism is well-known [5, 28, 16]. The extension of this correspondence to initial algebras comes from [35]. [7] exploits this connection to define a schema to *test* the equivalence of two programs t, u of type $\forall X.(F\langle X \rangle \Rightarrow X) \Rightarrow (G\langle X \rangle \Rightarrow X') \Rightarrow H\langle X \rangle$ by first instantiating X as $\alpha = \mu X.F\langle X \rangle$ and then applying t, u to the canonical morphism $F\langle X \mapsto \alpha \rangle \Rightarrow \alpha$ (in fact, this is exactly how one side of the isomorphisms $\equiv_X$ are constructed). The possibility of expressing program equivalence through naturality conditions has recently attracted new attention due to [3], where these are investigated using ideas from homotopy type theory. Type isomorphisms in $\Lambda 2$ with the Yoneda lemma are also discussed in [17]. In [25] a similar restriction based on the Yoneda isomorphism is used by the first author to describe a fragment of *second order multiplicative linear logic* with a decidable program equivalence.

Future Work. The definition of the characteristic employs an acyclicity condition which is reminiscent of linear logic *proof-nets*. In particular, we would like to investigate whether the alternating paths can be related to the *cyclic proofs* for linear logic systems with μ, ν-types [4]. Moreover, the notion of characteristic seems likely to scale to second order *multiplicative-exponential* linear logic, an extension which might lead to better expose the intrinsic duality in the tree-shapes in Fig. 5.

The application of Yoneda isomorphisms to count type inhabitants suggests that these can be related to some canonical proof-search strategy, as already suggested in [30], that we would like to investigate further. Moreover, the appeal to least/greatest fixpoints suggests a connection with the proof-technique to count inhabitants by computing *fixpoints of polynomial equations* [38]. For example, given $A = (Y_1 \Rightarrow X) \Rightarrow (X \Rightarrow Y_2 \Rightarrow X) \Rightarrow X$, one can show by proof-theoretic reasoning that the number $|A|$ of inhabitants of A is a solution of the fixpoint equation $|A| = |A_{Y_1}| + (|A| \times |A_{Y_2}|)$, where $A_{Y_i} = (Y_1 \Rightarrow X) \Rightarrow (X \Rightarrow Y_2 \Rightarrow X) \Rightarrow Y_i$, which implies $|A| = 0$, since $|A_{Y_i}| = 0$. On the other hand, Yoneda type isomorphisms yield the strikingly similar computation $\forall \vec{Y} X.A \equiv_X \forall \vec{Y}.\mu X.Y_1 + (X \times Y_2) \equiv_{\vec{Y}} \mu X.0 + (X \times 0) \equiv 0$.

References

1 Amal Ahmed, Dustin Jamner, Jeremy G. Siek, and Philip Wadler. Theorems for free for free: parametricity, with and without types. In *Proceedings of the ACM on Programming Languages*, volume 1 of *ICFP*, page Article No. 39, New York, 2017.

2 T. Aoto. Uniqueness of normal proofs in implicational intuitionistic logic. *Journal of Logic, Language and Information*, 8:217–242, 1999.

3 Steve Awodey, Jonas Frey, and Sam Speight. Impredicative encodings of (higher) inductive types. In *LICS 2018*, 2018.

4 David Baelde, Amina Doumane, and Alexis Saurin. Infinitary proof theory: the multiplicative-additive case. In *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:17, Germany, 2016. Dagstuhl.

5 E.S. Bainbridge, Peter J. Freyd, Andre Scedrov, and Philip J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70:35–64, 1990.

6 Henning Basold and Helle Hvid Hansen. Well-definedness and observational equivalence for inductive-coinductive programs. *Journal of Logic and Computation*, exw091, 2016.

7 Jean-Philippe Bernardy, Patrik Jansson, and Koen Claessen. Testing Polymorphic Properties. In *ESOP 2010: Programming Languages and Systems*, volume 6012 of *Lecture Notes in Computer Science*, pages 125–144. Springer Berlin Heidelberg, 2010.

8 P. Bourreau and S. Salvati. Game semantics and uniqueness of type inhabitance in the simply-typed λ-calculus. In *TLCA 2011*, volume 6690 of *LNCS*, pages 61–75. Springer Berlin Heidelberg, 2011.

9 S. Broda and L. Damas. On long normal inhabitants of a type. *Journal of Logic and Computation*, 15:353–390, 2005.

10 Kim B. Bruce, Roberto Di Cosmo, and Giuseppe Longo. Provable isomorphisms of types. *Mathematical Structures in Computer Science*, 1:1–20, 1991.

11 Joachim de Lataillade. Dinatural terms in System F. In *Proceedings of the Twenty-Fourth Annual IEEE Symposium on Logic in Computer Science (LICS 2009)*, pages 267–276, Los Angeles, California, USA, 2009. IEEE Computer Society Press.

12 Roberto Di Cosmo. A short survey of isomorphisms of types. *Mathematical Structures in Computer Science*, 15:825–838, 2005.

13 Fiore, Roberto Di Cosmo, and Vincent Balat. Remarks on isomorphisms in typed lambda calculi with empty and sum types. *Annals of Pure and Applied Logic*, 141(1–2):35–50, 2006.

14 Nicola Gambino and Joachim Kock. Polynomial functors and polynomial monads. *Mathematical Structures in Computer Science*, 154(1):153–192, 2013.

15 Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Normal forms and cut-free proofs as natural transformations. In Y. Moschovakis, editor, *Logic from Computer Science*, volume 21, pages 217–241. Springer-Verlag, 1992.

16 Ryu Hasegawa. Categorical data types in parametric polymorphism. *Mathematical Structures in Computer Science*, 4(1):71–109, 1994.

17 Ralf Hinze and Daniel W.H. James. Reason isomorphically! In *WGP 2010*, pages 85–96, 2010.

18 Danko Ilik. Axioms and decidability for type isomorphism in the presence of sums. 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)*, 2014.

19 Patricia Johann. On proving the correctness of program transformation based on free theorems for higher-order polymorphic calculi. *Mathematical Structures in Computer Science*, 15:201–229, 2005.

20 Ralph Matthes. *Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types*. PhD thesis, Ludwig-Maximilians-Universität München, 1998.

21 Guy McCuscker and Alessio Santamaria. On compositionality of dinatural transformations. In *CSL 2018*, volume 119 of *LIPIcs*, pages 33:1–33:22, Dagstuhl, Germany, 2018. Schloss Daghstuhl–Leibinz-Zentrum fuer Informatik.

22 M. Okada and P.J. Scott. A note on rewriting theory for uniqueness of iteration. *Theory and Applications of Categories*, 6:47–64, 1999.

23 Paolo Pistone. On dinaturality, typability and $\beta\eta$-stable models. In Schloos Dagstul-Leibniz-Zentrum fuer Informatik, editor, *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:17, Dagstuhl, Germany, 2017.

24 Paolo Pistone. On completeness and parametricity in the realizability semantics of System F. *Logical Methods in Computer Science*, 15(4):6:1–6:54, 2019.

25 Paolo Pistone. Proof nets, coends and the Yoneda isomorphism. In Thomas Ehrhard, Maribel Fernández, Valeria de Paiva, and Lorenzo Tortora de Falco, editors, *Proceedings Joint International Workshop on* Linearity *&* Trends in Linear Logic and Applications *(Linearity-TLLA 2018)*, volume 292 of *EPTCS*, pages 148–167, 2019.

26 Paolo Pistone and Luca Tranchini. The Yoneda Reduction of Polymorphic Types, extended version, 2020. URL: https://arxiv.org/abs/1907.03481.

27 Paolo Pistone, Luca Tranchini, and Mattia Petrolo. The naturality of natural deduction (II). Some remarks on atomic polymorphism, 2020. URL: https://arxiv.org/abs/1908.11353.

28 Gordon Plotkin and Martin Abadi. A logic for parametric polymorphism. In *TLCA '93, International Conference on Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 361–375. Springer Berlin Heidelberg, 1993.

29 Luigi Santocanale. Free μ-lattices. *Journal of Pure and Applied Algebra*, 9:166–197, 2002.

30 Gabriel Scherer. *Which types have a unique inhabitant? Focusing on pure program equivalence.* PhD thesis, Université Paris-Diderot, 2016.

31 Gabriel Scherer. Deciding equivalence with sums and the empty type. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 374–386, New York, NY, USA, 2017. ACM.

32 Gabriel Scherer and Didier Rémy. Which simple types have a unique inhabitant? *SIGPLAN Not.*, 50(9):243–255, 2015.

33 Makoto Tatsuta. Second order permutative conversions with Prawitz's strong validity. *Progress in Informatics*, 2:41–56, 2005.

34 Luca Tranchini, Paolo Pistone, and Mattia Petrolo. The naturality of natural deduction. *Studia Logica*, 107(1):195–231, 2019.

35 Tarmo Uustalu and Varmo Vene. The Recursion Scheme from the Cofree Recursive Comonad. *Electronic Notes in Theoretical Computer Science*, 229(5):135–157, 2011.

36 Janis Voigtländer. Free theorems simply, via dinaturality. In *Declarative Programming and Knowledge Management*, pages 247–267, Cham, 2020. Springer International Publishing.

37 Philip Wadler. Theorems for free! In *Proceedings of the fourth international conference on functional programming languages and computer architecture - FPCA '89*, 1989.

38 Marek Zaoinc. Fixpoint technique for counting terms in typed lambda-calculus. Technical report, State University of New York, 1995.

A Type Systems, Type Isomorphisms and Equational Theories

The typing rules and β-, η-rules of $\mathsf{\Lambda 2}, \mathsf{\Lambda 2p}, \mathsf{\Lambda 2p}_{\mu\nu}$ are recalled in Fig. 8. The standard axiomatization of $\beta\eta$-isomorphisms for $\mathsf{\Lambda 2}$ and $\mathsf{\Lambda 2p}$ are recalled in Fig. 9 and 10, and a "minimalistic" axiomatization of $\beta\eta$-isomorphisms for μ, ν-types is illustrated in Fig. 11.

The *contextual equivalence* relation for $\mathsf{\Lambda 2p}_{\mu\nu}$ and $\mathsf{\Lambda p}$ is defined by

$$\Gamma \vdash t \simeq_{\mathsf{ctx}} u : A \text{ iff for all context } \mathtt{C} : (\Gamma \vdash A) \Rightarrow (\vdash 1 + 1),\ \mathtt{C}[t] \simeq_{\beta\eta} \mathtt{C}[u]$$

The contextual equivalence relation for $\mathsf{\Lambda 2}$ is defined by

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \Lambda X.t : \forall X.A}\; X \notin FV(\Gamma) \qquad \frac{\Gamma \vdash t : \forall X.A}{\Gamma \vdash tB : A[B/X]}$$

(a) Typing rules for Λ2.

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u\rangle : A \times B} \qquad \frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash \pi_i^{A_i} t : A_i} \qquad \frac{}{\Gamma \vdash \star : 1}$$

$$\frac{\Gamma \vdash t : A_i}{\Gamma \vdash \iota_i t : A_1 + A_2} \qquad \frac{\Gamma \vdash t : A_1 + A_2 \qquad (\Gamma, y : A_i \vdash u_i : C)_{i=1,2}}{\Gamma \vdash \delta_C(t, y.u_1, y.u_2) : C} \qquad \frac{\Gamma \vdash t : 0}{\Gamma \vdash \xi_A t : A}$$

$$\frac{\Gamma \vdash t : A\langle X \mapsto \mu X.A\langle X\rangle\rangle}{\Gamma \vdash \mathsf{in}_A t : \mu X.A\langle X\rangle} \qquad \frac{\Gamma \vdash t : A\langle X \mapsto B\rangle \Rightarrow B}{\Gamma \vdash \mathsf{fold}_A(t) : \mu X.A\langle X\rangle \Rightarrow B}$$

$$\frac{\Gamma \vdash t : \nu X.A\langle X\rangle}{\Gamma \vdash \mathsf{out}_A t : A\langle X \mapsto \nu X.A\langle X\rangle\rangle} \qquad \frac{\Gamma \vdash t : B \Rightarrow A\langle X \mapsto B\rangle}{\Gamma \vdash \mathsf{unfold}_A(t) : B \Rightarrow \nu X.A\langle X\rangle}$$

(b) Typing rules for $+, \times, 0, 1, \mu, \nu$.

$$(\lambda x.t)u \;\simeq_\beta\; t[u/x] \qquad (\Lambda X.t)B \;\simeq_\beta\; t[B/X]$$

$$\frac{\Gamma \vdash t : A \to B}{\Gamma \vdash t \simeq_\eta \lambda x.tx : A \to B} \qquad \frac{\Gamma \vdash t : \forall X.A}{\Gamma \vdash t \;\simeq_\eta\; \Lambda X.tX : \forall X.A}$$

(c) β and η-rules for Λ2.

$$\Big(\pi_i^{A_i}\langle t_1, t_2\rangle \;\simeq_\beta\; t_i\Big)_{i=1,2} \qquad \Big(\delta_C(\iota_i t, y.u_1, y.u_2) \;\simeq_\beta\; u_i[t/y]\Big)_{i=1,2}$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash t \simeq_\eta \langle \pi_1^A t, \pi_2^B t\rangle : A \times B} \qquad \frac{\Gamma \vdash t : 1}{\Gamma \vdash t \simeq_\eta \star : 1}$$

$$\frac{\Gamma \vdash t : A + B \qquad u[x] : A + B \vdash^\Gamma C}{\Gamma \vdash u[t] \simeq_\eta \delta_C(t, y.u[\iota_1 y], y.u[\iota_2 y]) : C} \qquad \frac{\Gamma \vdash t : 0 \qquad u[x] : 0 \vdash^\Gamma A}{\Gamma \vdash u[t] \simeq_\eta \xi_A t : A}$$

(d) β and η-rules for $+, \times, 0, 1$.

$$\mathsf{fold}_P(t)(\mathsf{in}_P u) \simeq_\beta t(\Phi_P^X(\mathsf{fold}_P(t)x)[x \mapsto u])$$

$$\mathsf{out}_P(\mathsf{unfold}_P(t)u) \simeq_\beta \Phi_P^X(\mathsf{unfold}_P(t)x)[x \mapsto tu]$$

$$\frac{\Gamma \vdash u : A\langle X \mapsto C\rangle \Rightarrow C \qquad t[x] : \mu X.A\langle X\rangle \vdash^\Gamma C \qquad t[\mathsf{in}_A x] \simeq u\Phi_A^X(t) : A\langle X \mapsto \mu X.A\langle X\rangle\rangle \vdash^\Gamma C}{t[x] \simeq_\eta \mathsf{fold}_A(u)x : \mu X.A\langle X\rangle \vdash^\Gamma C}$$

$$\frac{\Gamma \vdash u : C \Rightarrow A\langle X \mapsto C\rangle \qquad t[x] : C \vdash^\Gamma \nu X.A\langle X\rangle \qquad \Phi_A^X(t)[x \mapsto ux] \simeq \mathsf{out}_A t : C \vdash^\Gamma A\langle X \mapsto \nu XA\langle X\rangle\rangle}{t[x] \simeq_\eta \mathsf{unfold}_A(u)x : C \vdash^\Gamma \nu X.A\langle X\rangle}$$

(e) β and η-rules for μ, ν.

Figure 8 Typing rules and $\beta\eta$-rules.

$$A \Rightarrow (B \Rightarrow C) \equiv B \Rightarrow (A \Rightarrow C)$$

$$\forall X.\forall Y.A \equiv \forall Y.\forall X.A$$

$$A \equiv \forall X.A \quad (X \notin FV(A)) \qquad A \Rightarrow \forall X.B \equiv \forall X.A \Rightarrow B$$

Figure 9 Axiomatization of $\beta\eta$-isomorphisms for $\Lambda 2$.

$$A \times 1 \equiv A \qquad 1 \Rightarrow A \equiv A$$

$$A \times (B \times C) \equiv (A \times B) \times C \qquad A \times B \equiv B \times A$$

$$(A \times B) \Rightarrow C \equiv A \Rightarrow (B \Rightarrow C) \qquad \forall X.A \times B \equiv \forall X.A \times \forall X.B$$

$$A + 0 \equiv A \quad A \times 0 \equiv 0 \qquad 0 \Rightarrow A \equiv 1$$

$$A + (B + C) \equiv (A + B) + C \qquad A + B \equiv B + A$$

$$A \times (B + C) \equiv (A \times B) + (A \times C) \qquad (A + B) \Rightarrow C \equiv (A \Rightarrow C) \times (B \Rightarrow C)$$

Figure 10 Axiomatization of $\beta\eta$-isomorphisms for $\Lambda 2\mathsf{p}$.

$$\Gamma \vdash t \simeq_{\mathsf{ctx}} u : A \text{ iff for all context } \mathtt{C} : (\Gamma \vdash A) \Rightarrow (\vdash \forall X.X \Rightarrow X \Rightarrow X),\ \mathtt{C}[t] \simeq_{\beta\eta} \mathtt{C}[u]$$

It is a standard result that contextual equivalence (for either $\Lambda 2\mathsf{p}_{\mu\nu}$ or $\Lambda 2$) is closed under congruence rules and thus generates an equational theory ctx.

For all positive type $A\langle X\rangle$ and negative type $B\langle X\rangle$ the functors $\Phi^X_A : \mathbb{C}_{\beta\eta}(\Lambda 2) \to \mathbb{C}_{\beta\eta}(\Lambda 2)$ and $\Phi^X_B : \mathbb{C}_{\beta\eta}(\Lambda 2)^{\mathsf{op}} \to \mathbb{C}_{\beta\eta}(\Lambda 2)$ are defined by letting $\Phi^X_A(C) = A[C/X]$, $\Phi^X_B(C) = B[C/X]$ and for all $t[x] : C \vdash D$,

$$\Phi^X_X(t) = t \tag{5}$$

$$\Phi^X_Y(t) = x \tag{6}$$

$$\Phi^X_{(C \Rightarrow D)}(t) = \lambda y.\Phi^X_D(t[\Phi^X_C(y)]) \tag{7}$$

$$\Phi^X_{(\forall X.C)}(t) = \Lambda Y.\Phi^X_C(tY) \tag{8}$$

One can check that $\Phi^X_A(x) \simeq_\eta x$ and $\Phi^X_A(t[u[x]]) \simeq_\beta \Phi^X_A(t)\Big[x \mapsto \Phi^X_A(u)\Big]$.

The definition above can be extended to the types defined using all other constructors, yielding functors $\Phi^X_A : \mathbb{C}_{\beta\eta}(\Lambda 2\mathsf{p}_{\mu\nu}) \to \mathbb{C}_{\beta\eta}(\Lambda 2\mathsf{p}_{\mu\nu})$ and $\Phi^X_B : \mathbb{C}_{\beta\eta}(\Lambda 2\mathsf{p}_{\mu\nu})^{\mathsf{op}} \to \mathbb{C}_{\beta\eta}(\Lambda 2\mathsf{p}_{\mu\nu})$.

B The ε-Theory and the Yoneda Isomorphisms

In this section we describe the equational theory induced by the interpretation of polymorphic programs as dinatural transformations (for a suitable fragment of $\Lambda 2\mathsf{p}_{\mu\nu}$), that we call the ε-theory, and we show that the type isomorphisms $\equiv_X$ and $\equiv_X$ from Section 2 hold under this theory.

We will work for simplicity in an *ad-hoc* fragment $\Lambda 2\mathsf{p}^*_{\mu,\nu}$ of $\Lambda 2\mathsf{p}_{\mu\nu}$, in which we require

$$\mu X.A \equiv \nu X.A \equiv A \quad (X \notin FV(A))$$

$$\mu X.A\langle X\rangle \equiv A\langle X \mapsto \mu X.A\langle X\rangle\rangle \qquad \nu X.A\langle X\rangle \equiv A\langle X \mapsto \nu X.A\langle X\rangle\rangle$$

Figure 11 Axiomatization of $\beta\eta$-isomorphisms for μ,ν-types.

that all universal types are of one of the two forms below:

$$\forall X.\Big\langle \forall \vec{Y}_k.\langle A_{jk}\langle X\rangle\rangle_j \Rightarrow X\Big\rangle_k \Rightarrow B\langle X\rangle \qquad \forall X.\Big\langle \forall \vec{Y}_j.X \Rightarrow A_j\langle X\rangle\Big\rangle_j \Rightarrow B\langle X\rangle \quad (9)$$

As this set of types is stable by substitution, all type rules and equational rules of $\Lambda 2\mathsf{p}_{\mu\nu}$ scale well to $\Lambda 2\mathsf{p}^{*}_{\mu,\nu}$.

To each universal type $\forall X.C$ of $\Lambda 2\mathsf{p}^{*}_{\mu,\nu}$ as in Eq. (9) left (resp. Eq. (9) right) we associate the ε-*rule* illustrated in Fig. 12a (resp. Fig. 12b).[4] From the viewpoint of category theory, the two ε-rules express *strong dinaturality* conditions [35] for the transformations induced by polymorphic programs (illustrated in Fig. 12c and Fig. 12d).

▶ **Definition 37** (ε-theory)**.** *The* ε-theory *of* $\Lambda 2\mathsf{p}^{*}_{\mu,\nu}$ *is the smallest congruent equational theory closed under* β-*,* η-*equations as well as* ε-*rules.*

We will show that the isomorphism schema $\equiv_{X}$, that we recall below, holds under the ε-theory (a similar argument can be developed for the isomorphism schema $\equiv_{X}$, see [26]).

$$\forall X.\Big\langle \forall \vec{Y}_k.\langle A_{jk}\langle X\rangle\rangle_j \Rightarrow X\Big\rangle_k \Rightarrow B\langle X\rangle \equiv B\left\langle X \mapsto \{\mu X.\}\sum_k\left(\exists \vec{Y}_k.\prod_j A_{jk}\langle X\rangle\right)\right\rangle \quad (10)$$

▶ **Notation B.1.** *Let* $L = \langle i_1,\dots,i_k\rangle$ *be a list. If* $(t_k)_{i\in L}$ *is a* L*-indexed list of terms, we let for any term* u*,* $u\langle t_k\rangle_{i\in L}$ *be shorthand for* $ut_{i_1}\dots t_{i_k}$*; if* $\langle x_{i_1},\dots,x_{i_k}\rangle$ *is a* L*-indexed list of variables, we let for any term* u*,* $\lambda\langle x_k\rangle_{i\in L}.u$ *be shorthand for* $\lambda x_{i_1}\dots\lambda x_{i_k}.u$*.*

In the case of Eq. (10) we can construct terms

$$\begin{aligned}\mathsf{a}_k[\langle z_j\rangle_j] = \mathtt{in}_k^{\sharp K}(\mathsf{pack}[\vec{Y}_k](\mathtt{prod}_j^{\sharp J_k}\langle z_j\rangle_j)) \ &: \ \big\langle A_{jk}\langle X \mapsto \alpha\rangle\big\rangle_j \vdash T\langle X \mapsto \alpha\rangle\\ \hat{\mathsf{a}}_k = \Lambda\vec{Y}_k.\lambda\langle z_j\rangle_j.\mathtt{in}_T(\mathsf{a}_k[\langle z_j\rangle_j]) \ &: \ \forall\vec{Y}_k.\langle A_{jk}\langle X \mapsto \alpha\rangle\rangle_j \Rightarrow \alpha\end{aligned}$$

where $T\langle X\rangle = \sum_k \exists\vec{Y}_k.\prod_j A_{jk}\langle X\rangle$ and $\alpha = \mu X.T\langle X\rangle$, $\mathtt{in}_k^{\sharp I} : X_k \Rightarrow \sum_k^{\sharp I} X_k$ and $\mathtt{prod}_j$: $\langle X_j\rangle_j \Rightarrow \prod_j^{\sharp J_k} X_j$ are defined composing usual sum and product constructors, and pack is defined as follows:

$$\mathsf{pack}[B_1,\dots,B_k] = \lambda x.\Lambda Z.\lambda f.xB_1\dots B_k f : A[B_1/Y_1,\dots,B_k/Y_k] \Rightarrow \exists\vec{Y}.A$$

With such terms we can then construct a term

$$\mathtt{s}_{A_{jk},B}[x] = x\langle\hat{\mathsf{a}}_k\rangle_k : \Big\langle \forall\vec{Y}_k.\langle A_{jk}\langle X \mapsto \alpha\rangle\rangle_j \Rightarrow \alpha\Big\rangle_k \Rightarrow B\langle X \mapsto \alpha\rangle \quad \vdash \quad B\langle X \mapsto \alpha\rangle$$

Moreover, using sum and product destructors we can construct terms

$$\begin{aligned}\mathtt{b}[x,Z] \ &: \ T\langle X \mapsto Z\rangle \vdash^{\Delta} Z\\ \mathtt{t}_{A_{jk},B}[x,Z] \ &: \ B\langle X \mapsto \alpha\rangle \quad \vdash \quad \Big\langle \forall\vec{Y}_k.\langle A_{jk}\langle X \mapsto Z\rangle\rangle_j \Rightarrow Z\Big\rangle_k \Rightarrow B\langle X \mapsto Z\rangle\end{aligned}$$

[4] Observe that $\forall X.C$ might well be of *both* forms (9) left and right

$$\dfrac{\begin{array}{c} t : \forall X.C \\ \Big(e_k : \forall \vec{Y}_k.\langle A_{jk}\langle E\rangle\rangle_j \Rightarrow E\Big)_k \\ \Big(f_k : \forall \vec{Y}_k.\langle A_{jk}\langle F\rangle\rangle_j \Rightarrow F\Big)_k \\ \mathtt{v}[x] : E \vdash F \\ \mathtt{v}[e_k\vec{Y}_k\langle z_j\rangle_j] \simeq f_k\vec{Y}_k\langle \Phi^X_{A_{jk}}(\mathtt{v})[z_j]\rangle_j : \langle A_{jk}\langle X \mapsto E\rangle\rangle_j \vdash F \end{array}}{\Gamma \vdash \Phi^X_B(\mathtt{v})\Big[x \mapsto tE\langle e_k\rangle_k\Big] \simeq tF\langle f_k\rangle_k : B\langle X \mapsto F\rangle}$$

(a) ε-rule for the left-hand type in Eq. (9).

$$\dfrac{\begin{array}{c} t : \forall X.C \\ \Big(e_j : \forall \vec{Y}_k.E \Rightarrow A_j\langle E\rangle\Big)_j \\ \Big(f_j : \forall \vec{Y}_k.F \Rightarrow A_j\langle F\rangle\Big)_j \\ \mathtt{v}[x] : E \vdash F \\ \Phi^X_{A_j}(\mathtt{v}[x])[e_jx] \simeq f_j\mathtt{v}[x] : E \vdash A_j\langle X \mapsto F\rangle \end{array}}{\Gamma \vdash \Phi^X_B(\mathtt{v})\Big[x \mapsto tF\langle f_j\rangle_j\Big] \simeq tE\langle e_j\rangle_j : B\langle X \mapsto E\rangle}$$

(b) ε-rule for the left-hand type in Eq. (9).

$$\begin{tikzcd}
 & \big\langle \forall \vec{Y}_k.\langle A_{jk}\langle E\rangle\rangle_j \Rightarrow E\big\rangle_k \arrow[rr, "\langle y_k\rangle_k \mapsto tE\langle y_k\rangle_k"] \arrow[dr, "\langle \forall \vec{Y}_k.\langle A_{jk}\langle E\rangle\rangle_j \Rightarrow \mathtt{v}\rangle_k"] & & B\langle E\rangle \arrow[dd, "B(\mathtt{v})"] \\
1 \arrow[ur, "\langle e_k\rangle_k"] \arrow[dr, "\langle f_k\rangle_k"'] & & \langle \forall \vec{Y}_k.\langle A_{jk}\langle E\rangle\rangle_j \Rightarrow F\rangle_k & \\
 & \big\langle \forall \vec{Y}_k.\langle A_{jk}\langle F\rangle\rangle_j \Rightarrow F\big\rangle_k \arrow[rr, "\langle y_k\rangle_k \mapsto tF\langle y_k\rangle_k"'] \arrow[ur, "\langle \forall \vec{Y}_k.\langle A_{jk}\langle \mathtt{v}\rangle\rangle_j \Rightarrow F\rangle_k"'] & & B\langle F\rangle
\end{tikzcd}$$

(c) Strong dinaturality diagram for the ε-rule **(a)**.

$$\begin{tikzcd}
 & \langle \forall \vec{Y}_j.E \Rightarrow A_j\langle E\rangle\rangle_j \arrow[rr, "\langle y_j\rangle_j \mapsto tE\langle y_j\rangle_j"] \arrow[dr, "\forall \vec{Y}_j.\langle E \Rightarrow A_j\langle \mathtt{v}\rangle\rangle_j"] & & B\langle E\rangle \\
1 \arrow[ur, "\langle e_j\rangle_j"] \arrow[dr, "\langle f_j\rangle_j"'] & & \langle \forall \vec{Y}_j.E \Rightarrow A_j\langle F\rangle\rangle_j & \\
 & \langle \forall \vec{Y}_j.F \Rightarrow A_j\langle F\rangle\rangle_j \arrow[rr, "\langle y_j\rangle_j \mapsto tF\langle y_j\rangle_j"'] \arrow[ur, "\forall \vec{Y}_j.\langle \mathtt{v} \Rightarrow A_j\langle F\rangle\rangle_j"'] & & B\langle F\rangle \arrow[uu, "B(\mathtt{v})"']
\end{tikzcd}$$

(d) Strong dinaturality diagram for the ε-rule **(b)**.

Figure 12 ε-rules and their associated strong dinaturality diagrams.

where $\Delta = \{\langle f_k : \forall \vec{Y}_k.\langle A_{jk}\langle X \mapsto Z\rangle\rangle_j \Rightarrow Z\rangle_k\}$ and

$$\begin{aligned} \mathsf{b}[x, Z] &= \delta^{\sharp K}\Big(x, \Big\langle z.\mathsf{unpack}(z)\big(\Lambda\vec{Y}_k.\lambda y.f_k\vec{Y}_k\langle \pi^{\sharp J_k}_j(y)\rangle_j\big)\Big\rangle_k\Big) \\ \mathtt{t}_{A_{jk},B}[x, Z] &= \lambda\langle f_k\rangle_k.\Phi^X_B\big(\mathsf{fold}_T(\lambda x.\mathsf{b}[x, Z])x\big) \end{aligned}$$

where π^j_i and $\delta^{\sharp K}(t, \langle z.u_k\rangle_k)$ indicate suitable generalized product and sum destructors which can be defined inductively using product and sum destructors, and the term unpack is defined as follows:

$$\mathsf{unpack} = \Lambda Z.\lambda x.\lambda f.fZx : \forall Z.\left(\exists\vec{Y}.A\right) \Rightarrow (\forall\vec{Y}.A \Rightarrow Z) \Rightarrow Z$$

One can check that $\mathsf{b}[x,\alpha]\big[\langle f_k \mapsto \hat{\mathsf{a}}_k\rangle_k\big] : T\langle X \mapsto \alpha\rangle \vdash \alpha$ is $\beta\eta$-equivalent to $\mathsf{in}_T x$, from which we deduce

$$\mathsf{fold}_T\Big(\lambda x.\mathsf{b}[x,\alpha]\big[\langle f_k \mapsto \hat{\mathsf{a}}_k\rangle_k\big]\Big)x \simeq_{\beta\eta} \mathsf{fold}_T(\lambda x.\mathsf{in}_T x)x \simeq_\eta x$$

In this way the isomorphism $\equiv_X$ are realized in $\mathbb{C}_\varepsilon(\mathsf{\Lambda 2p}^*_{\mu,\nu})$ by the two terms below:[5]

$$\mathtt{s}[x] = \mathtt{s}_{A_{jk},B}[x\alpha] \qquad\qquad \mathtt{t}[x] = \Lambda X.\mathtt{t}_{A_{jk},B}[x,X]$$

We can compute then

$$\mathtt{s}[\mathtt{t}[x]] \simeq_\beta \Phi^X_B\Big(\mathsf{fold}_T\big(\lambda x.\mathsf{b}[x,\alpha]\big[\langle f_k \mapsto \langle\hat{\mathsf{a}}_k\rangle_k\big]\big)x\Big) \simeq_{\beta\eta} \Phi^X_B(x) \simeq_\eta x$$

and

$$\mathtt{t}[\mathtt{s}[x]] \simeq_\beta \Lambda X.\lambda\langle f_k\rangle_k.\Phi^X_B\Big(\mathsf{fold}_T(\lambda x.\mathsf{b}[x,X])x\Big)\Big[x \mapsto x\alpha\langle\hat{\mathsf{a}}_k\rangle_k\Big]$$

$$\simeq_\varepsilon \Lambda X.\lambda\langle f_k\rangle_k.xX\langle f_k\rangle_k \simeq_\eta x$$

where the central ε-equivalence is justified using the ε-rule in Fig. 12a with $E = \alpha$, $F = X$, $e_k = \hat{\mathsf{a}}_k$ and $\mathtt{v}[x] = \mathsf{fold}_T(\lambda x.\mathsf{b}[x,X])x$, with the last premise given by the computation below:

$$\Big(\mathsf{fold}_T(\lambda x.\mathsf{b}[x,X])x\Big)\Big[x \mapsto \hat{\mathsf{a}}_k\vec{Y}_k\langle z_j\rangle_j\Big] \simeq_\beta \Big(\mathsf{fold}_T(\lambda x.\mathsf{b}[x,X])\Big)\mathsf{in}_T(\mathsf{a}_k[\langle z_j\rangle_j])$$

$$\simeq_\beta \big(\lambda x.\mathsf{b}[x,X]\big)\Big(\big(\Phi^X_T(\mathsf{fold}_T(\lambda x.\mathsf{b}[x,X])x)\big)\Big[x \mapsto \mathsf{a}_k[\langle z_j\rangle_j]\Big]\Big)$$

$$\simeq_\beta f_k\vec{Y}_k\Big\langle \Phi^X_{A_{jk}}(\mathsf{fold}_T(\lambda x.\mathsf{b}[x,X])x)[x \mapsto z_j]\Big\rangle_j$$

where the last β-equation can be checked by unrolling the definition of Φ^X_T:

$$\Phi^X_T(t) = \delta^{\sharp K}\Big(x, \Big\langle z.\mathsf{unpack}(z)\big(\Lambda\vec{Y}_k.\lambda x.\mathtt{in}^{\sharp K}_k(\mathsf{pack}[\vec{Y}_k](\mathtt{prod}^{\sharp J_k}\langle\Phi^X_{A_{jk}}(t[x \mapsto \pi^{\sharp J_k}_j(x)])\rangle_j))\big)\Big\rangle_k\Big)$$

[5] We are here supposing that X does occur in at least some of the A_{jk} (so that $\mu X.$ actually occurs in the left-hand type of $\equiv_X$). If this is not the case, the construction can be done in a similar (and simpler) way.

Degrees of Ambiguity for Parity Tree Automata

Alexander Rabinovich
Tel Aviv University, Israel
https://www.cs.tau.ac.il/~rabinoa/
rabinoa@tauex.tau.ac.il

Doron Tiferet[1]
Tel Aviv University, Israel
sdoron5.t2@gmail.com

Abstract

An automaton is unambiguous if for every input it has at most one accepting computation. An automaton is finitely (respectively, countably) ambiguous if for every input it has at most finitely (respectively, countably) many accepting computations. An automaton is boundedly ambiguous if there is $k \in \mathbb{N}$, such that for every input it has at most k accepting computations. We consider Parity Tree Automata (PTA) and prove that the problem whether a PTA is not unambiguous (respectively, is not boundedly ambiguous, not finitely ambiguous) is co-NP complete, and the problem whether a PTA is not countably ambiguous is co-NP hard.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases automata on infinite trees, degree of ambiguity, omega word automata, parity automata

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.36

Funding Supported in part by Len Blavatnik and the Blavatnik Family foundation.

Acknowledgements We would like to thank the reviewers for their useful suggestions. The first author is grateful to Michał Skrzypczak for very fruitful discussions.

1 Introduction

Degrees of Ambiguity

The relationship between deterministic and nondeterministic machines plays a central role in computer science. An important topic is the comparison of expressiveness, succinctness and complexity of deterministic and nondeterministic models. Various restricted forms of nondeterminism were suggested and investigated (see [3, 4] for recent surveys).

Probably, the oldest restricted form of nondeterminism is unambiguity. An automaton is unambiguous if for every input there is at most one accepting run. For automata over finite words there is a rich and well-developed theory on the relationship between deterministic, unambiguous and nondeterministic automata [4]. All three models have the same expressive power. Unambiguous automata are exponentially more succinct than deterministic ones, and nondeterministic automata are exponentially more succinct than unambiguous ones [6, 7].

Some problems are easier for unambiguous than for nondeterministic automata. As shown by Stearns and Hunt [13], the equivalence and inclusion problems for unambiguous automata are in polynomial time, while these problems are PSPACE-complete for nondeterministic automata.

[1] corresponding author

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 36; pp. 36:1–36:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

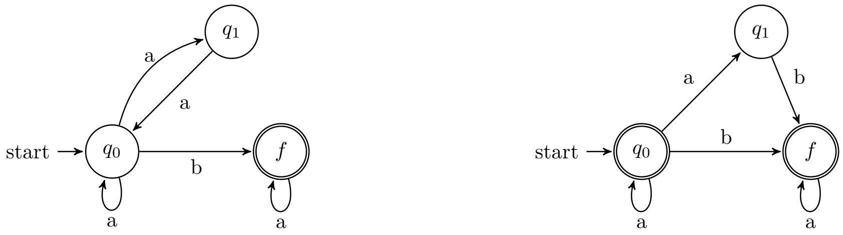


Figure 1 Finitely ambiguous and 2-ambiguous Büchi automata.

The complexity of basic regular operations on languages represented by unambiguous finite automata was investigated in [5], and tight upper bounds on state complexity of intersection, concatenation and many other operations on languages represented by unambiguous automata were established. It is well-known that the tight bound on the state complexity of the complementation of nondeterministic automata is 2^n. In [5], it was shown that the complement of the language accepted by an n-state unambiguous automaton is accepted by an unambiguous automaton with $2^{0.79n+\log n}$ states.

Many other notions of ambiguity were suggested and investigated. A recent paper [4] surveys works on the degree of ambiguity and on various nondeterminism measures for finite automata on words.

An automaton is *k-ambiguous* if on every input it has at most k accepting runs; it is *boundedly ambiguous* if it is k-ambiguous for some k; it is *finitely ambiguous* if on every input it has finitely many accepting runs.

It is clear that an unambiguous automaton is k-ambiguous for every $k > 0$, and a k-ambiguous automaton is finitely ambiguous. The reverse implications fail. For ϵ-free automata over words (and over finite trees), on every input there are at most finitely many accepting runs. Hence, every ϵ-free automaton on finite words and on finite trees is finitely ambiguous. However, over ω-words there are nondeterministic automata with uncountably many accepting runs. Over ω-words and over infinite trees, finitely ambiguous automata are a proper subclass of the class of countably ambiguous automata, which is a proper subclass of nondeterministic automata.

Weber and Seidl [15] investigated several classes of ambiguous automata on words, and obtained polynomial time algorithms for deciding the membership in each of these classes. Their algorithms were derived from structural characterizations of the classes.

In particular, they proved that the following Bounded Ambiguity Criterion (BA) characterizes whether there is a bound k such that a nondeterministic automaton on words has at most k accepting runs on each word.

Forbidden Pattern for Bounded Ambiguity: There are distinct useful[2] states $p, q \in Q$ such that for some word u, there are runs on u from p to p, from p to q and from q to q.

Weber and Seidl [15] proved that an NFA is not boundedly ambiguous iff it contains the forbidden pattern for bounded ambiguity. This pattern is testable in polynomial time; hence, it can be decided in polynomial time whether the degree of ambiguity of an NFA is bounded.

Seidl [12] provided a structural characterization of bounded ambiguity for automata on finite trees, and derived a polynomial algorithm to decide whether such an automaton is boundedly ambiguous.

[2] A state is useful if it is on an accepting run.

Löding and Pirogov [8] and Rabinovich [10] provided structural characterizations and polynomial algorithms for bounded, finite and countable ambiguity of Büchi automata on ω-words.

Rabinovich and Tiferet [11] provided a structural characterizations and polynomial time algorithms for bounded, finite and countable ambiguity of Büchi automata on infinite trees.

Over infinite trees, Büchi tree automata are less expressive than Monadic Second-Order Logic which is equivalent to parity tree automata. Our main result is:

▶ **Theorem 1** (Main).

1. *The problem whether a parity tree automaton is not unambiguous (respectively, not boundedly ambiguous, or not finitely ambiguous) is co-NP-complete.*
2. *The problem whether a parity tree automaton is not countably ambiguous is co-NP hard.*

The paper's organization. The next section contains standard definitions and notations about tree automata. The proof of Theorem 1 relies on the complexity of many-dimensional parity games. In Sect. 3, we first recall many-dimensional parity games, introduced by Chatterjee, Henzinger, and Piterman in [2]. We provide reductions between the emptiness problem for multidimensional parity tree automata and multidimensional parity games. Then, we show that the non-emptiness problem for intersection of PTA is polynomial time reducible to the non-emptiness problem of multi-dimensional PTA, and therefore is in co-NP.

In Sect. 4, we prove co-NP-hardness for the degrees of ambiguity of a PTA. This establishes the lower bounds stated in Theorem 1. The proof of the upper bounds of Theorem 1 is based on structural characterizations of degrees of ambiguity for PTA. In Sect. 5 we lift the structural characterizations of Büchi ω-word automata [8, 10] to structural characterizations of parity ω-word automata and provide a polynomial algorithm for the degrees of ambiguity of parity ω-word automata. In Sect. 6 we present characterizations for the finite and bounded ambiguity of PTA. Finally, in Sect. 7 we prove the co-NP upper bounds of the Main Theorem. The last section presents the conclusion.

2 Preliminaries

We recall here standard terminology and notations about trees and automata [14, 9].

Trees. We view the set $\{l, r\}^*$ of finite words over alphabet $\{l, r\}$ as the domain of a full-binary tree, where the empty word ϵ is the root of the tree, and for each node $v \in \{l, r\}^*$, we call $v \cdot l$ the left child of v, and $v \cdot r$ the right child of v.

We define a tree order "$\leq$" as a partial order such that $\forall u, v \in \{l, r\}^* : u \leq v$ iff u is a prefix of v. Nodes u and v are incomparable - denoted by $u \perp v$ - if neither $u \leq v$ nor $v \leq u$.

We say that an infinite sequence $\pi = v_0, v_1, \dots$ is a **tree branch** if $v_0 = \epsilon$ and $\forall i \in \mathbb{N} : v_{i+1} = v_i \cdot l$ or $v_{i+1} = v_i \cdot r$.

If Σ is a finite alphabet, then a Σ-labeled full-binary tree t is a labeling function $t : \{l, r\}^* \to \Sigma$. We denote by T^ω_Σ the set of all Σ-labeled full-binary trees. We often use "tree" for "labeled full-binary tree."

Given a Σ-labeled tree t and a node $v \in \{l, r\}^*$, the tree $t_{\geq v}$ (called the subtree of t, rooted at v) is defined by $t_{\geq v}(u) := t(v \cdot u)$ for each $u \in \{l, r\}^*$.

A tree language L over an alphabet Σ is a set of Σ-labeled trees.

Automata on ω-words and on infinite trees. An automaton on ω-words is a tuple $\mathcal{A} := (Q_{\mathcal{A}}, \Sigma, Q_I, \delta, Acc)$ where Σ is a finite alphabet, $Q_{\mathcal{A}}$ is a finite set of states, $Q_I \subseteq Q_{\mathcal{A}}$ is a set of initial states, $\delta \subseteq Q_{\mathcal{A}} \times \Sigma \times Q_{\mathcal{A}}$ is a transition relation, and Acc is an acceptance condition. A run of $\mathcal{A}$ on an ω-word $y = a_0a_1\dots$ is an infinite sequence $\rho = q_0, q_1, \dots$ such that $q_0 \in Q_I$, and for all $i \in \mathbb{N} : (q_i, a_i, q_{i+1}) \in \delta$.

The Büchi acceptance conditions are given by a set $F \subseteq Q_{\mathcal{A}}$. We say that a run ρ is accepting if there is a state $f \in F$ which appears infinitely often in ρ.

The parity acceptance conditions are given by a function $\mathbb{C} : Q_{\mathcal{A}} \to \mathbb{N}$. We say that a run ρ is accepting if the maximal number which appears infinitely often in $\mathbb{C}(q_0), \mathbb{C}(q_1), \dots$ is even.

We denote the set of all accepting runs of $\mathcal{A}$ on an ω-word y by $ACC(\mathcal{A}, y)$. The language of $\mathcal{A}$ is defined as $L(\mathcal{A}) := \{y \in \Sigma^\omega \mid ACC(\mathcal{A}, y) \neq \emptyset\}$.

An automaton on infinite trees is a tuple $\mathcal{A} := (Q_{\mathcal{A}}, \Sigma, Q_I, \delta, Acc)$ where $Q_{\mathcal{A}}$, Σ, Q_I are as in an automaton on ω-words, $\delta \subseteq Q_{\mathcal{A}} \times \Sigma \times Q_{\mathcal{A}} \times Q_{\mathcal{A}}$ is a transition relation, and Acc is an acceptance condition. A computation of $\mathcal{A}$ on a Σ-labeled tree t is a function $\phi : \{l, r\}^* \to Q_{\mathcal{A}}$ such that $\phi(\epsilon) \in Q_I$, and $\forall v \in \{l, r\}^* : (\phi(v), t(v), \phi(v \cdot l), \phi(v \cdot r)) \in \delta$.

The Büchi acceptance conditions are given by a set $F \subseteq Q_{\mathcal{A}}$. We say that ϕ is accepting if for each branch $\pi = v_0, v_1, \dots$ there is a state $f \in F$ such that the sequence $\phi(v_0), \phi(v_1), \dots$ contains infinitely many occurrences of f. The parity acceptance conditions are given by a function $\mathbb{C} : Q_{\mathcal{A}} \to \mathbb{N}$. We say that ϕ is accepting if for each branch $\pi = v_0, v_1, \dots$ the maximal number which appears infinitely often in $\mathbb{C}(\phi(v_0)), \mathbb{C}(\phi(v_1)), \dots$ is even.

We denote the set of all accepting computations of $\mathcal{A}$ on t by $ACC(\mathcal{A}, t)$. The language of $\mathcal{A}$ is defined as $L(\mathcal{A}) := \{t \mid ACC(\mathcal{A}, t) \neq \emptyset\}$.

A state $q \in Q_{\mathcal{A}}$ of $\mathcal{A}$ is *useful* if it appears on an accepting computation.

Given an automaton $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, Q_I, \delta, Acc)$ and a state $q \in Q_{\mathcal{A}}$, $\mathcal{A}_q$ is defined as $\mathcal{A}_q := (Q_{\mathcal{A}}, \Sigma, \{q\}, \delta, Acc)$, by replacing the set of initial states of $\mathcal{A}$ by $\{q\}$.

We will use PTA for Parity Tree Automata, BTA for Büchi Tree Automata, PWA for Parity Word Automata, and BWA for Büchi Word Automata.

Degree of Ambiguity of an Automaton. We denote by $|X|$ the cardinality of a set X. An automaton $\mathcal{A}$ is k-ambiguous if $|ACC(\mathcal{A}, t)| \leq k$ for all $t \in L(\mathcal{A})$. $\mathcal{A}$ is unambiguous if it is 1-ambiguous. $\mathcal{A}$ is boundedly ambiguous if there is $k \in \mathbb{N}$ such that $\mathcal{A}$ is k-ambiguous, $\mathcal{A}$ is finitely ambiguous if $ACC(\mathcal{A}, t)$ is finite for all t, $\mathcal{A}$ is countably ambiguous if $ACC(\mathcal{A}, t)$ is countable for all t. The set $ACC(\mathcal{A}, t)$ is definable in the Monadic Second-Order logic (MSO). The results of Bárány et al. in [1] imply that every uncountable MSO-definable (in the full-binary tree) set has cardinality $2^{\aleph_0}$.

The degree of ambiguity of $\mathcal{A}$ (notation $da(\mathcal{A})$) is defined by $da(\mathcal{A}) := k$ if $\mathcal{A}$ is k-ambiguous and either $k = 1$ or $\mathcal{A}$ is not $k - 1$ ambiguous, $da(\mathcal{A}) := \mathit{finite}$ if $\mathcal{A}$ is finitely ambiguous and not boundedly ambiguous, $da(\mathcal{A}) := \aleph_0$ if $\mathcal{A}$ is countably ambiguous and not finitely ambiguous, and $da(\mathcal{A}) := 2^{\aleph_0}$ if $\mathcal{A}$ is not countably ambiguous.

3 Non-Emptiness Problem for Intersection of PTA

In this section we will prove that deciding the non-emptiness of the intersection of k PTA is in co-NP. Our proof relies on a reduction from k-dimensional parity tree automata to k-dimensional parity games. We first recall k-dimensional parity games [2] and introduce k-dimensional parity automata.

▶ **Definition 2.** *A k-dimensional parity game is a tuple $G = (S_1, S_2, E, \mathbb{P})$, where (S_1, S_2, E) is a directed bipartite graph: S_i the set of states of Player i, E a set of edges such that each state $s \in S_1 \cup S_2$ has at least one outgoing edge $(s, s') \in E$; and $\mathbb{P} : S_1 \cup S_2 \to \mathbb{N}^k$ a (priority) function. The game starts at some state $s \in S$. The players construct an infinite sequence of states (called a play) as follows: Let s be the last state in the sequence. If $s \in S_i$, then Player i chooses an edge $(s, s') \in E$ and the state s' is added to the sequence. Since each state has at least one successor, the sequence can always be continued.*

Let $s_1, s_2, \dots$ be the play which is constructed by the selections of the two players, and let $\mathbb{P}(s_1), \mathbb{P}(s_1), \dots$ be a sequence of priorities in $(\mathbb{N}^k)^\omega$. We say Player 1 wins the play if for every $i \leq k$ the maximal value which is seen infinitely often in the i-th coordinates is even. Otherwise, we say that Player 2 wins the play.

A strategy for Player i specifies for each sequence $s_1, \dots s_m$ where $s_m \in S_i$, the next state s' such that $(s_m, s') \in E$. A play is consistent with a strategy of Player i if each move of Player i in the play is according to the strategy.

The winning region of Player 1 is a subset $S' \subseteq S_1 \cup S_2$, for which there exists a strategy of Player 1 such that each play from $s' \in S'$ which is consistent with it is winning for Player 1. The winning region of Player 2 is defined similarly.

Notice that each play $s_1, s_2, \dots$ could equivalently be represented by a sequence of edges $e_1, e_2, \dots$ such that $e_i = (s_i, s_{i+1})$.

▶ **Definition 3** (k-dimensional PTA)**.** *A k-dimensional PTA is a tuple $\mathcal{A} = (Q, \Sigma, Q_I, \delta, \mathbb{C})$ where Q, Σ, Q_I, and δ are as in PTA, and $\mathbb{C} : Q \to \mathbb{N}^k$ is a function which assigns a k-dimensional color vector to each state in Q. A computation ϕ of $\mathcal{A}$ on t is accepting if for each branch $\pi = v_0, v_1, \dots$ and each coordinate $i \leq k$, the maximal color which occurs infinitely often in the i-th coordinate of $\mathbb{C}(\phi(v_0)), \mathbb{C}(\phi(v_1)), \dots$ is even.*

▶ **Theorem 4** (Chatterjee, Henzinger and Piterman [2])**.** *Let $G = (S_1, S_2, E, \mathbb{P})$ be a k-dimensional parity game, for $k > 1$. The problem of deciding whether a node $s \in S_1$ is in the winning region of Player 1 is co-NP-complete.*

We use Theorem 4 to obtain the following result regarding the non-emptiness problem of k-dimensional parity tree automata:

▶ **Proposition 5.** *(1) The non-emptiness problem for k-dimensional PTA is in co-NP. (2) The non-emptiness problem for deterministic 2-dimensional PTA is co-NP-hard.*

Proof. (1) We use the standard reduction from the emptiness problem for automata to games; it works also for k-dimensional parity conditions. Given a k-dimensional PTA $\mathcal{A} = (Q, \Sigma, Q_I, \delta, \mathbb{C})$, we define a k-dimensional parity game $G(\mathcal{A}) = (S_1, S_2, E, \mathbb{P})$ as follows:

- $S_1 = Q$
- $S_2 = \delta$
- $(q, (p, a, p_1, p_2)) \in E$ iff $(p, a, p_1, p_2) \in \delta$ and $p = q$
- $((p, a, p_1, p_2), q) \in E$ iff $q = p_1$ or $q = p_2$
- $\forall s \in S_1 : \mathbb{P}(s) := \mathbb{C}(s)$, and $\forall s \in S_2 : \mathbb{P}(s) := \underbrace{(0, \dots, 0)}_{k \text{ times}}$

Recall that for $s \in Q$ an automaton $\mathcal{A}_s$ is defined by replacing the set of initial states of $\mathcal{A}$ by $\{s\}$. For every state $s \in Q$, $L(\mathcal{A}_s)$ is non-empty iff Player 1 has a winning strategy from s. Hence, by Theorem 4 we conclude that deciding the non-emptiness of $\mathcal{A}$ is in co-NP.

(2) We prove this using a reduction from the problem of deciding whether Player 1 has a winning strategy from state $s \in S_1$ in a 2-dimensional parity game. Since this problem is co-NP-hard (by Theorem 4), the result will follow.

Let $G = (S_1, S_2, E, \mathbb{P})$ be a 2-dimensional parity game with priority function $\mathbb{P} : S_1 \cup S_2 \to \mathbb{N}^2$. We will assume without restriction that:

- The out degree of each node in G is 2.
- $\forall s \in S_2 : \mathbb{P}(s) = (0, 0)$.

For each node $s \in S_1$, we denote one of its successors by $E_a(s)$ and the other by $E_b(s)$, and for each node $s \in S_2$ we denote one of its successors by $E_l(s)$ and the other by $E_r(s)$. We refer to the selection of edge $(s', E_h(s')) \in E$ for $h \in \{a, b, l, r\}$ by Player i as the h move of Player i. Notice that a sequence of states in the game could equivalently be represented using a sequence of h moves.

We construct a deterministic 2-dimensional parity automaton $\mathcal{A} := \mathcal{A}(G) = (Q, \Sigma, Q_I, \delta, \mathbb{C})$ as follows:

- $Q := S_1$
- $\Sigma := \{a, b\}$
- $Q_I := S_1$
- $\forall c \in \Sigma, \forall q \in Q : \delta(q, c) = (E_l(E_c(q)), E_r(E_c(q)))$
- $\mathbb{C} := \mathbb{P}|_{S_1}$ (the restriction of $\mathbb{P}$ onto S_1)

We will prove that Player 1 has a winning strategy from a state $s \in S_1$ iff $\mathcal{A}_s$ is non-empty. This implies the co-NP-hardness of deciding whether $\mathcal{A}_s$ is non-empty.

$\Rightarrow$: A strategy of Player 1 from a state s can be represented as a function from a sequence of moves of Player 2 to a move of Player 1. Assume Player 1 has a winning strategy $\mathit{strategy}_1 : \{l, r\}^* \to \{a, b\}$ from a state s. We will use $\mathit{strategy}_1$ to construct a tree $t \in L(\mathcal{A})$ and an accepting computation $\phi \in \mathit{ACC}(\mathcal{A}_s, t)$.

Define $t(d_1 \dots d_n) := \mathit{stategy}_1(d_1, \dots, d_n)$. Define a computation ϕ on t as $\phi(\epsilon) := s$ and $\phi(v \cdot d_{n+1}) := E_{d_{n+1}}(E_{t(v)}(\phi(v))$ for $v \in \{l, r\}^n$ and $d_{n+1} \in \{l, r\}$. It is easy to verify that ϕ is a computation of $\mathcal{A}_s$ on t (this holds even for the non-winning strategies of Player 1).

A proof that ϕ is an accepting computation of $\mathcal{A}_s$ on t is also simple. Let $\pi = v_1, v_2, \dots$ be a tree branch such that $v_1 = \epsilon$ and $\forall i \in \mathbb{N} : v_{i+1} = v_i \cdot d_i$ where $d_i \in \{l, r\}$. By definition of ϕ, $\phi(\pi)$ corresponds to the states of Player 1 in a play which is consistent with winning strategy $\mathit{strategy}_1$, and with Player 2 choosing move d_i on his i-th turn. This play is winning for Player 1; hence, it satisfies the 2-dimensional parity condition; therefore, $\phi(\pi)$ satisfies the acceptance conditions. Since π was an arbitrary branch, this implies that $\phi \in \mathit{ACC}(\mathcal{A}_s, t)$, and $\mathcal{A}_s$ is non-empty.

$\Leftarrow$: Assume that $\mathcal{A}_s$ is non-empty. Therefore, there exists $\phi \in \mathit{ACC}(\mathcal{A}_s, t)$. We use t to define a strategy $\mathit{strategy}_1 : \{l, r\}^* \to \{a, b\}$ for Player 1, by $\mathit{strategy}_1(d_1, \dots, d_n) := t(d_1 \dots d_n)$. We leave to the reader the verification that $\mathit{strategy}_1$ is winning for Player 1 from s. ◀

We will now proceed to show the connection between k-dimensional automata and the intersection of parity automata.

▶ **Definition 6** (Product of PTA). *Given two PTA $\mathcal{A} = (Q_\mathcal{A}, \Sigma, Q_I^\mathcal{A}, \delta_\mathcal{A}, \mathbb{C}_\mathcal{A})$ and $\mathcal{B} = (Q_\mathcal{B}, \Sigma, Q_I^\mathcal{B}, \delta_\mathcal{B}, \mathbb{C}_\mathcal{B})$ we define the 2-dimensional PTA $\mathcal{A} \times \mathcal{B} := (Q_\times, \Sigma, Q_I^\times, \delta_\times, \mathbb{C}_\times)$ where:*

- $Q_\times := Q_\mathcal{A} \times Q_\mathcal{B}$
- $Q_I^\times := Q_I^\mathcal{A} \times Q_I^\mathcal{B}$
- $((p, q), a, (p_l, q_l), (p_r, q_r)) \in \delta_\times$ *iff* $(q, a, q_l, q_r) \in \delta_\mathcal{A}$ *and* $(p, a, p_l, p_r) \in \delta_\mathcal{B}$
- $\forall (q, p) \in Q_\mathcal{A} \times Q_\mathcal{B} : \mathbb{C}_\times(q, p) = (\mathbb{C}_\mathcal{A}(q), \mathbb{C}_\mathcal{B}(p))$

The product automata of k PTA is defined similarly, as a k-dimensional PTA.

▶ **Lemma 7.** *Let $\mathcal{A}_1, \dots, \mathcal{A}_k$ be parity tree automata, and define $\mathcal{A}_\times$ as the k-dimensional product automaton $\mathcal{A}_1 \times \cdots \times \mathcal{A}_k$. Then $L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k) \neq \emptyset$ iff $L(\mathcal{A}_\times) \neq \emptyset$.*

Proof. $\Rightarrow$: Assume that $L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k) \neq \emptyset$. Therefore, there is an infinite tree $t \in L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k)$ and accepting computations $\phi_i \in ACC(\mathcal{A}_i, t)$ for all $1 \leq i \leq k$. Define a computation $\phi_\times$ such that $\phi_\times(u) := (\phi_1(u), \ldots, \phi_k(u))$ for all $u \in \{l, r\}^*$. By definition of $\mathcal{A}_\times$ we conclude that $\phi_\times \in ACC(\mathcal{A}_\times, t)$, and therefore $t \in L(\mathcal{A}_\times)$ and $L(\mathcal{A}_\times) \neq \emptyset$

$\Leftarrow$: Assume that $L(\mathcal{A}_\times) \neq \emptyset$. Therefore, there is a tree $t \in L(\mathcal{A}_\times)$ and an accepting computation $\phi_\times \in ACC(\mathcal{A}_\times, t)$. For each $u \in \{l, r\}^*$ there are k states $q_1^u, \ldots, q_k^u$ such that $\phi_\times(u) = (q_1^u, \ldots, q_k^u)$. Define a computation ϕ_i by $\phi_i(u) := q_i^u$. By definition of $\mathcal{A}_\times$ it follows that $\phi_i \in ACC(\mathcal{A}_i, t)$. We conclude that for each automaton $\mathcal{A}_i$ there is an accepting computation ϕ_i of $\mathcal{A}_i$ on t, and therefore $t \in L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k)$ and we obtain $L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k) \neq \emptyset$. ◀

▶ **Lemma 8.** *For every $k \in \mathbb{N}$, the problem of deciding whether the intersection of k PTA is non-empty is in co-NP.*

Proof. Given k PTA $\mathcal{A}_1, \ldots, \mathcal{A}_k$, the product automaton $\mathcal{A}_1 \times \cdots \times \mathcal{A}_k$ can be computed in polynomial time in the size of $\mathcal{A}_1, \ldots, \mathcal{A}_k$. By Lemma 7 we conclude that deciding the non-emptiness of $L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k)$ is equivalent to deciding the non-emptiness of $L(\mathcal{A}_1 \times \cdots \times \mathcal{A}_k)$. $\mathcal{A}_1 \times \cdots \times \mathcal{A}_k$ is a k-dimensional PTA and therefore, by Proposition 5(1), this problem is in co-NP. ◀

4 co-NP-hardness of Deciding the Degree of Ambiguity of a PTA

In this section we will use the results of Sect. 3 and prove the co-NP hardness lower bounds stated in Theorem 1. We will first prove the co-NP-hardness of deciding whether a PTA is ambiguous, and then use a polynomial time reduction to show that co-NP-hardness holds for other ambiguity degree problems (Proposition 10).

▶ **Lemma 9.** *The problem of deciding whether a PTA is ambiguous is co-NP-hard.*

Proof. We will use a reduction from the problem of deciding the non-emptiness of deterministic 2-dimensional PTA (which is co-NP-hard, by Proposition 5(2)), to the problem of deciding whether a PTA is ambiguous.

Let $\mathcal{D} = (Q, \Sigma, q_{init}, \delta, \mathbb{C})$ be a deterministic 2-dimensional PTA. Define $\mathcal{D}_i := (Q, \Sigma, q_{init}, \delta, \mathbb{C}_i)$ for $i = 1, 2$ as the deterministic PTA obtained from $\mathcal{D}$ by defining a coloring function $\mathbb{C}_i$ such that $\mathbb{C}_i(q)$ returns the i-th coordinate of $\mathbb{C}(q)$ for each $q \in Q$.

Let $\mathcal{D}'_2$ be an automaton which is isomorphic to $\mathcal{D}_2$, and does not share common states with $\mathcal{D}_1$ (this could be achieved by renaming the states of $\mathcal{D}_2$). Since $\mathcal{D}_1$ and $\mathcal{D}'_2$ are deterministic, it easily follows that $L(\mathcal{D}) \neq \emptyset$ iff $L(\mathcal{D}_1) \cap L(\mathcal{D}'_2) \neq \emptyset$.

Given two PTA $\mathcal{A} = (Q_\mathcal{A}, \Sigma, Q_I^\mathcal{A}, \delta_\mathcal{A}, \mathbb{C}_\mathcal{A})$ and $\mathcal{B} = (Q_\mathcal{B}, \Sigma, Q_I^\mathcal{B}, \delta_\mathcal{B}, \mathbb{C}_\mathcal{B})$ such that $Q_\mathcal{A} \cap Q_\mathcal{B} = \emptyset$, we define the parity automaton $\mathcal{A} \cup \mathcal{B} := (Q_\mathcal{A} \cup Q_\mathcal{B}, \Sigma, Q_I^\mathcal{A} \cup Q_I^\mathcal{B}, \delta_\mathcal{A} \cup \delta_\mathcal{B}, \mathbb{C}_\mathcal{A} \cup \mathbb{C}_\mathcal{B})$. Notice that $L(\mathcal{A} \cup \mathcal{B}) = L(\mathcal{A}) \cup L(\mathcal{B})$.

We will prove that $da(\mathcal{D}_1 \cup \mathcal{D}'_2) > 1$ iff $L(\mathcal{D}_1) \cap L(\mathcal{D}'_2)$ is non-empty:

$\Rightarrow$: Assume that $da(\mathcal{D}_1 \cup \mathcal{D}'_2) > 1$. Then, there exist two accepting computations $\phi_1, \phi_2 \in ACC(\mathcal{D}_1 \cup \mathcal{D}'_2, t)$. Since $\mathcal{D}_1$ and $\mathcal{D}'_2$ are deterministic, each of them has at most one accepting computation, and therefore, if $\phi_1 \in ACC(\mathcal{D}_1, t)$ then $\phi_2 \in ACC(\mathcal{D}'_2, t)$. Hence, $t \in L(\mathcal{D}_1) \cap L(\mathcal{D}'_2)$.

$\Leftarrow$: Assume $t \in L(\mathcal{D}_1) \cap L(\mathcal{D}'_2)$. Therefore, there are computations $\phi_1 \in ACC(\mathcal{D}_1, t)$, $\phi_2 \in ACC(\mathcal{D}'_2, t)$. By definition of $\mathcal{D}_1 \cup \mathcal{D}'_2$ we have $\phi_1, \phi_2 \in ACC(\mathcal{D}_1 \cup \mathcal{D}'_2, t)$. Since $\mathcal{D}_1$ and $\mathcal{D}'_2$ have no common states, we have $\phi_1 \neq \phi_2$ and therefore $da(\mathcal{D}_1 \cup \mathcal{D}'_2) > 1$.

We conclude that $L(\mathcal{D}) \neq \emptyset$ iff $da(\mathcal{D}_1 \cup \mathcal{D}'_2) > 1$, and therefore deciding whether a PTA is ambiguous is co-NP-hard. ◀

▶ **Proposition 10.** *The problem of deciding whether a PTA is not countably ambiguous (respectively, is not finitely ambiguous, or is not boundedly ambiguous) is co-NP-hard.*

Proof. We will prove it using a reduction from the problem of deciding whether a parity tree automaton is ambiguous, which is co-NP-hard by Lemma 9.

Let $\mathcal{A} = (Q, \Sigma, Q_I, \delta, \mathbb{C})$ be a PTA, and let $a \in \Sigma$. Construct an automaton $\mathcal{B} := (Q_{\mathcal{B}}, \Sigma, Q_I^{\mathcal{B}}, \delta_{\mathcal{B}}, \mathbb{C}_{\mathcal{B}})$, where:

- $Q_{\mathcal{B}} := Q \cup \{f\}$ and $f \notin Q$ is a new state,
- $Q_I^{\mathcal{B}} := \{f\}$,
- $\delta_{\mathcal{B}} = \delta \cup \{(f, a, f, q) \mid q \in Q_I\}$,
- $\mathbb{C}_{\mathcal{B}}(f) := 0$ and for all $q \in Q : \mathbb{C}_{\mathcal{B}}(q) := \mathbb{C}(q)$.

▷ Claim 11. **1.** If $\mathcal{A}$ is ambiguous, then $\mathcal{B}$ is not countably ambiguous.
2. If $\mathcal{B}$ is ambiguous, then $\mathcal{A}$ is ambiguous.

Proof. (1) Assume that $\mathcal{A}$ is ambiguous. Therefore, there is a tree $t \in L(\mathcal{A})$ and two computations $\phi_1, \phi_2 \in ACC(\mathcal{A}, t)$ such that $\phi_1 \neq \phi_2$. Let t' be a tree such that $\forall v \in l^* : t'(v) = a$ and $t'_{\geq v \cdot r} = t$. For each $S \subseteq \mathbb{N}$, define a computation ϕ_S by

$$\phi_S(u) := \begin{cases} f & u \in l^* \\ \phi_1(v) & u = l^i \cdot r \cdot v \text{ for } i \in S \\ \phi_2(v) & u = l^i \cdot r \cdot v \text{ for } i \notin S \end{cases}$$

It is easy to see that $\forall S \subseteq \mathbb{N} : \phi_S \in ACC(\mathcal{B}, t')$, and that $\forall S_1, S_2 \subseteq \mathbb{N} : S_1 \neq S_2 \rightarrow \phi_{S_1} \neq \phi_{S_2}$. Therefore, $ACC(\mathcal{B}, t')$ is not countable, and $\mathcal{B}$ is not countably ambiguous.

(2) Assume that $\mathcal{B}$ is ambiguous. Therefore, there is a tree $t \in L(\mathcal{B})$ and computations $\phi', \phi'' \in ACC(\mathcal{B}, t)$ such that $\phi' \neq \phi''$. Let $v \in \{l, r\}^*$ such that $\phi'(v) \neq \phi''(v)$. By definition of $\mathcal{B}$ we have $\forall u \in l^* : \phi'(u) = \phi''(u) = f$, and therefore, $v \notin l^*$. Hence, there is $w \in l^* \cdot r$ such that $v \geq w$. Let $\phi'_{\geq w}$ and $\phi''_{\geq w}$ be the restrictions of ϕ' and ϕ'', respectively, on $t_{\geq w}$. It is clear that $\phi'_{\geq w} \neq \phi''_{\geq w}$. By definition of $\mathcal{B}$ we obtain $\phi'_{\geq w}, \phi''_{\geq w} \in ACC(\mathcal{A}, t_{\geq w})$, and therefore $\mathcal{A}$ is ambiguous. ◁

The parity tree automaton $\mathcal{B}$ can be constructed in a polynomial time in the size of $\mathcal{A}$. By Claim 11 we obtain that the following conditions are equivalent:

- $\mathcal{A}$ is not unambiguous
- $\mathcal{B}$ is not countably ambiguous
- $\mathcal{B}$ is not finitely ambiguous
- $\mathcal{B}$ is not boundedly ambiguous

Therefore, there are polynomial time reductions from the problem of deciding whether a PTA is not unambiguous (which is co-NP-hard, by Lemma 9), to the problems of deciding whether a PTA is not countably ambiguous/not finitely ambiguous/not boundedly ambiguous. Hence, those problems are co-NP-hard. ◀

5 Degree of Ambiguity for Parity Automata on ω-words

In this section we will present structural characterizations and polynomial algorithms for deciding the degree of ambiguity of parity automata on ω-word (PWA). These characterizations and algorithms are derived from similar characterizations for Büchi automata on ω-words (BWA) given in [8, 10]. Throughout this section we will assume all states of the automata are useful (it is computable in polynomial time whether a state of a PWA/BWA is useful).

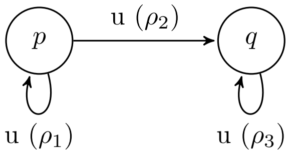


Figure 2 Forbidden Pattern for Finite Ambiguity of PWA. The runs ρ_1 and ρ_2 are distinct, $\mathbb{C}(q)$ is even, and $\mathbb{C}(q)$ is maximal among the colors which $\mathbb{C}$ assigns to the states in ρ_3.

The next definition and theorem are taken from [8, 10]. They provide a forbidden pattern characterization of the degrees of ambiguity of BWA.

▶ **Definition 12** (Forbidden pattern for BWA). *Let $\mathcal{B}$ be a BWA such that all its states are useful.*

- *$\mathcal{B}$ contains a* forbidden pattern for countable ambiguity, *if there is a final state f and there are two distinct runs of $\mathcal{B}_f$ on the same word u from f to f.*
- *$\mathcal{B}$ contains a* forbidden pattern for finite ambiguity, *if it contains the forbidden pattern for countable ambiguity or there is a final state f, a state $q \neq f$, and a word u such that there are runs of $\mathcal{B}_q$ on u from q to q and from q to f, and a run of $\mathcal{B}_f$ on u from f to f.*
- *$\mathcal{B}$ contains a* forbidden pattern for bounded ambiguity, *if there are distinct states p, q such that for a (finite) word u, there are runs of $\mathcal{B}_p$ on u from p to p and from p to q, and there is a run of $\mathcal{B}_q$ on u from q to q.*

▶ **Theorem 13** (Structural characterization). *Let $\mathcal{B}$ be a BWA.*

1. *$\mathcal{B}$ has uncountably many accepting runs on some ω-word iff $\mathcal{B}$ contains the forbidden pattern for countable ambiguity.*
2. *$\mathcal{B}$ has infinitely many accepting runs on some ω-word iff $\mathcal{B}$ contains the forbidden pattern for finite ambiguity.*
3. *$\mathcal{B}$ is not boundedly ambiguous iff it contains the forbidden pattern for bounded ambiguity.*

Now, we define forbidden patterns for PWA. Then, Proposition 16 proves their correctness.

▶ **Definition 14** (Forbidden pattern for PWA). *Let $\mathcal{A}$ be a PWA such that all its states are useful.*

- *$\mathcal{A}$ contains a* forbidden pattern for countable ambiguity, *if there is a state q such that $\mathbb{C}(q)$ is even, and two distinct runs ρ_1 and ρ_2 of $\mathcal{A}_q$ from q to q on the same finite word y such that $\forall p \in \rho_1 : \mathbb{C}(q) \geq \mathbb{C}(p)$ and $\forall p \in \rho_2 : \mathbb{C}(q) \geq \mathbb{C}(p)$.*
- *$\mathcal{A}$ contains a* forbidden pattern for finite ambiguity, *if there is a state q such that $\mathbb{C}(q)$ is even, a state p (which can be equal to q), and a word y such that there is a run ρ_1 of $\mathcal{A}_p$ on y from p to p, a run ρ_2 of $\mathcal{A}_p$ on y from p to q, and a run ρ_3 of $\mathcal{A}_q$ on y from q to q such that $\max\{\mathbb{C}(p') \mid p' \in \rho_3\}$ is even, and $\rho_1 \neq \rho_2$.*
- *$\mathcal{A}$ contains a* forbidden pattern for bounded ambiguity, *if there are two states $p \neq q$ and a finite word y such that there are runs of $\mathcal{A}_p$ on y from p to p and from p to q, and a run of $\mathcal{A}_q$ on y from q to q.*

First, we prove that the forbidden patterns provide sufficient conditions for the corresponding ambiguity.

▶ **Lemma 15** (Sufficient conditions for degrees of ambiguity). *Let $\mathcal{A}$ be a PWA.*

1. *If $\mathcal{A}$ contains the forbidden pattern for countable ambiguity, then it is not countably ambiguous.*
2. *If $\mathcal{A}$ contains the forbidden pattern for finite ambiguity, then it is not finitely ambiguous.*
3. *If $\mathcal{A}$ contains the forbidden pattern for bounded ambiguity, then it is not boundedly ambiguous.*

Proof. (1) Let y be a finite word, and let q be a state of $\mathcal{A}$ such that $\mathbb{C}(q)$ is even and there are two distinct runs ρ_1 and ρ_2 of $\mathcal{A}_q$ from q to q on y where $\forall p \in \rho_1 : \mathbb{C}(q) \geq \mathbb{C}(p)$ and $\forall p \in \rho_2 : \mathbb{C}(q) \geq \mathbb{C}(p)$.

We will show that there are infinitely many accepting runs of $\mathcal{A}_q$ on y^ω. Denote by ρ'_1 and ρ'_2 the runs ρ_1 and ρ_2, respectively, with the last state removed. For each $B \subseteq \mathbb{N}$, let $\rho^B := \rho^B_0 \cdot \rho^B_1 \cdots \in Q^\omega$, where $\rho^B_i := \rho'_1$ if $i \in B$, and $\rho^B_i := \rho'_2$ otherwise. It is clear that ρ^B is a computation of $\mathcal{A}_q$ on y^ω. Notice that q occurs infinitely often in ρ^B, and by the definition of ρ_1 and ρ_2 we conclude that $\mathbb{C}(q)$ is the maximal color which is assigned to a state in ρ^B.

Let $B_1, B_2 \subseteq \mathbb{N}$ such that $\exists b \in B_1 \setminus B_2$. We obtain $\rho^{B_1}_b = \rho'_1 \neq \rho'_2 = \rho^{B_2}_b$, and therefore $\rho^{B_1} \neq \rho^{B_2}$. There are uncountably many subsets of $\mathbb{N}$, and therefore there are uncountably many accepting computation of $\mathcal{A}_q$ on y^ω. Since q is a useful state we conclude that $\mathcal{A}$ is uncountably ambiguous.

(2) Let y be a finite word, q a state such that $\mathbb{C}(q)$ is even, and p a state such that there are three runs on y: a run ρ_1 of $\mathcal{A}_p$ from p to p, a run ρ_2 of $\mathcal{A}_p$ from p to q, and a run ρ_3 of $\mathcal{A}_q$ from q to q such that $\max\{\mathbb{C}(p') \mid p' \in \rho\}$ is even and $\rho_1 \neq \rho_2$.

We will show that there are infinitely many accepting runs of $\mathcal{A}_p$ on y^ω. Denote by ρ'_1, ρ'_2 and ρ'_3 the runs ρ_1, ρ_2 and ρ_3, respectively, with the last state removed. For each $k \in \mathbb{N}$, define a computation $\rho^k := \rho^k_0 \cdot \rho^k_1 \ldots$, where $\rho^k_i := \begin{cases} \rho'_1 & i < k \\ \rho'_2 & i = k \\ \rho'_3 & i > k \end{cases}$

It is easy to verify that ρ^k is a run of $\mathcal{A}_p$ on y^ω. Notice that q occurs infinitely often in ρ^k. Since each state which occurs infinitely often in ρ^k is in ρ'_3, we conclude that $\mathbb{C}(q)$ is the maximal color among the colors which are assigned to states which occurs infinitely often in ρ^k; hence, ρ^k is accepting. Let $k_1 < k_2 \in \mathbb{N}$. We obtain $\rho^{k_1}_{k_1} = \rho'_2 \neq \rho'_1 = \rho^{k_2}_{k_1}$, and therefore, $\rho^{k_1} \neq \rho^{k_2}$ and we conclude that $\mathcal{A}$ is not finitely ambiguous.

(3) Let y be a finite word such that there are two states $p \neq q$, and three runs: a run ρ_1 of $\mathcal{A}_p$ on y from p to p, a run ρ_2 of $\mathcal{A}_p$ from p to q, and a run ρ_3 of $\mathcal{A}_q$ from q to q.

Since q is a useful state, there is an ω-word $\widehat{y} \in L(\mathcal{A}_q)$, and an accepting run $\widehat{\rho}$ of $\mathcal{A}_q$ on $\widehat{y}$. We will prove that for each $k \in \mathbb{N}$, there are at least k accepting runs of $\mathcal{A}_p$ on $y^k \cdot \widehat{y}$.

Denote by ρ'_1, ρ'_2 and ρ'_3 the runs ρ_1, ρ_2 and ρ_3, respectively, with the last state removed. For each $0 \leq j < k$ we define a run $\rho^j := \rho^j_0 \cdot \rho^j_1 \ldots \rho^j_k$, where

$$\rho^j_i := \begin{cases} \rho'_1 & i < j \\ \rho'_2 & i = j \\ \rho'_3 & i > j \end{cases}$$

It is easy to verify that ρ^j is a run of $\mathcal{A}_p$ on y^k from p to q, and therefore, $\rho^j \cdot \widehat{\rho}$ is an accepting run of $\mathcal{A}_p$ on $y^k \cdot \widehat{y}$. Moreover, for each $j_1 < j_2 < k$ we have $\rho^{j_1}_{j_1} = \rho'_2 \neq \rho'_1 = \rho^{j_2}_{j_1}$, and therefore, $\rho^{j_1} \neq \rho^{j_2}$. We conclude that for each $k \in \mathbb{N}$ there are at least k different accepting computation of $\mathcal{A}_p$ on $y^k \cdot \widehat{y}$, and therefore $\mathcal{A}_p$ is not boundedly ambiguous. Since p is useful, we conclude that $\mathcal{A}$ is not boundedly ambiguous. ◀

▶ **Proposition 16** (Structural characterization). *Let $\mathcal{A}$ be a PWA.*

1. *$\mathcal{A}$ has uncountably many accepting runs on some ω-word iff $\mathcal{A}$ contains the forbidden pattern for countable ambiguity.*
2. *$\mathcal{A}$ has infinitely many accepting runs on some ω-word iff $\mathcal{A}$ contains the forbidden pattern for finite ambiguity.*

3. *$\mathcal{A}$ is not boundedly ambiguous iff it contains the forbidden pattern for bounded ambiguity.* The $\Leftarrow$ direction of the proposition was proved in Lemma 15. To prove the $\Rightarrow$ direction of Proposition 16 we will use the standard reduction of PWA to BWA.

Let $\mathcal{A} = (Q, \Sigma, Q_I, \delta, \mathbb{C})$ be a PWA, and let $\mathbb{C}_{even} := \{\mathbb{C}(q) \mid q \in Q \text{ and } \mathbb{C}(q) \text{ is even}\}$.

Define a BWA $\mathcal{B} := (Q', \Sigma, Q'_I, \delta', F)$, where:

- $Q' := Q \cup \{(c,p) \in \mathbb{C}_{even} \times Q \mid c \geq \mathbb{C}(p)\}$
- $Q'_I := Q_I \cup \{(c,p) \in Q' \mid p \in Q_I\}$
- δ' is the union of the following sets:
  - δ
  - $\{(p, a, (c, p')) \mid (p, a, p') \in \delta, \mathbb{C}(p) > c \text{ and } (c, p') \in Q'\}$
  - $\{((c,p), a, (c, p')) \mid (p, a, p') \in \delta \text{ and } (c, p') \in Q'\}$
- $F := \{(c,q) \mid \mathbb{C}(q) = c\}$

Proposition 16 immediately follows from Theorem 13, Lemma 17 and Lemma 18.

▶ **Lemma 17.** $|ACC(\mathcal{A}, y)| \leq |ACC(\mathcal{B}, y)|$

Proof. By the definition of PWA, for each accepting run $\rho := p_0, \dots p_i, \dots$ of $\mathcal{A}$ on y there is $i \in \mathbb{N}$ such that p_i occurs infinitely often in ρ, $c := \mathbb{C}(p_i)$ is even and $\forall j > i : \mathbb{C}(p_i) \geq \mathbb{C}(p_j)$.

If $\mathbb{C}(p_i) \geq \mathbb{C}(p)$ for all $p \in \rho$ then define $g(\rho) := (c, p_0), (c, p_1), \dots$. Otherwise, let k be such that $\mathbb{C}(p_k) > c$ and $\forall j > k : c \geq \mathbb{C}(p_j)$, and define $g(\rho) := p_0, \dots p_k, (c, p_{k+1}), (c, p_{k+2}), \dots$. By the definition of $\mathcal{B}$ we conclude that $g(\rho)$ is an accepting computation of $\mathcal{B}$ on y, since (c, p_i) occurs infinitely often in $g(\rho)$. It is easy to verify that g is injective, and therefore, we obtain $|ACC(\mathcal{A}, y)| \leq |ACC(\mathcal{B}, y)|$. ◀

▶ **Lemma 18.** **1.** *If $\mathcal{B}$ contains the forbidden pattern for countable ambiguity (of BWA) then $\mathcal{A}$ contains the forbidden pattern for countable ambiguity (of PWA).*

2. *If $\mathcal{B}$ contains the forbidden pattern for finite ambiguity (of BWA) then $\mathcal{A}$ contains the forbidden pattern for finite ambiguity (of PWA).*

3. *If $\mathcal{B}$ contains the forbidden pattern for bounded ambiguity (of BWA) then $\mathcal{A}$ contains the forbidden pattern for bounded ambiguity (of PWA).*

Proof. The lemma is proved by inspecting the transition relations of $\mathcal{A}$ and the corresponding Büchi automaton $\mathcal{B}$.

1. By definition of the forbidden pattern of countable ambiguity we conclude that there is a final state f and two distinct runs ρ_1 and ρ_2 of $\mathcal{B}_f$ on the same finite word y from f to f. By the definition of $\mathcal{B}$ there is a state $q \in Q$ with even color $c := \mathbb{C}(q)$ such that $f = (c, q)$, the run ρ_1 is of the form $(c, p_0), \dots, (c, p_n)$ and the run ρ_2 is of the form $(c, q_0), \dots, (c, q_n)$, where $p_0 = p_n = q_0 = q_n = q$.
The runs ρ_1 and ρ_2 are distinct, and therefore there is $0 < i < n$ such that $p_i \neq q_i$. We conclude that $q_0, \dots, q_n$ and $p_0, \dots, p_n$ are two distinct runs of $\mathcal{A}$ on y from q to q such that $\mathbb{C}(q)$ is even, $\forall p \in \rho_1 : \mathbb{C}(q) \geq \mathbb{C}(p)$ and $\forall p \in \rho_2 : \mathbb{C}(q) \geq \mathbb{C}(p)$. Therefore, $\mathcal{A}$ has a forbidden pattern for countable ambiguity.

2. If $\mathcal{B}$ contains the forbidden pattern for countable ambiguity, then by (1) we conclude that $\mathcal{A}$ contains the forbidden pattern for countable ambiguity. Hence, there is a state q such that $\mathbb{C}(q)$ is even, and there are two distinct runs ρ_1 and ρ_2 of $\mathcal{A}_q$ from q to q on the same finite word y, and for the run $\rho_3 := \rho_1$ we have $\forall p \in \rho_3 : \mathbb{C}(q) \geq \mathbb{C}(p)$. That is, $\mathcal{A}$ contains the forbidden pattern for finite ambiguity.
Otherwise, $\mathcal{B}$ has a final state f, a state $q \neq f$, and a finite word y such that there are runs ρ_1 of $\mathcal{B}_q$ on y from q to q, ρ_2 of $\mathcal{B}_q$ on y from q to f, and ρ_3 of $\mathcal{B}_f$ on y from f to f.

Since $f \in F$, we conclude that there is a state $q' \in Q$ with even color $c := \mathbb{C}(q')$ such that $f = (c, q')$. The run ρ_3 is therefore of the form $(c, p_0), \ldots, (c, p_n)$ where $p_0 = p_n = q'$. Hence, $p_0, \ldots, p_n$ is a run of $\mathcal{A}$ on y from q' to q' such that $\mathbb{C}(q')$ is even and $\forall i \leq n : c \geq \mathbb{C}(p_i)$.

Case 1: $q \in Q' \setminus Q$. By the definition of Q' we have $q = (c', p')$ for $c' \in \mathbb{C}_{even}$ and $p' \in Q$. Notice that the run ρ_2 is from (c', p') to (c, q'). By the definition of δ' we conclude that $c = c'$, and since $q \neq f$ we obtain $p' \neq q'$. By the definition of ρ_1 and ρ_2 we conclude that there are runs of $\mathcal{A}_{p'}$ on y from p' to p' and from p' to q', where $p' \neq q'$. Along with the run $p_0, \ldots, p_n$ from q' to q', we conclude that $\mathcal{A}$ contains the forbidden pattern for finite ambiguity.

Case 2: $q \in Q$. Notice that the run ρ_2 of $\mathcal{B}$ is from q to (c, q'). Therefore, by the definition of δ', there is a run $\rho_{\mathcal{A}}$ of $\mathcal{A}$ on y from q to q' which passes through a state with priority greater than c. If $q = q'$, then we conclude that there are two distinct runs of $\mathcal{A}_q$ on y from q to q: The run $\rho'_1 := \rho_{\mathcal{A}}$, which visits a state with a color greater than c, and the run $\rho'_2 := p_0, \ldots, p_n$ which only visits states of color at most c. Taking $\rho'_3 := \rho_2$, we conclude that $\mathcal{A}$ has the forbidden pattern for finite ambiguity. Otherwise, $q \neq q'$ and by the definition of ρ_1 and ρ_2 we conclude that there are runs of $\mathcal{A}_q$ from q to q and from q to q', and together with the run $p_0, \ldots, p_n$ from q' to q', we conclude that $\mathcal{A}$ contains the forbidden pattern for finite ambiguity.

3. $\mathcal{B}$ contains the forbidden pattern for bounded ambiguity. Therefore, there are distinct states $p \neq q$ and a finite word y such that there are runs of $\mathcal{B}_p$ on y from p to p and from p to q, and there is a run of $\mathcal{B}_q$ on y from q to q.

Case 1: $q \in Q$. Notice that there is a run of $\mathcal{B}$ from p to q, and by the definition of δ' we obtain $p \in Q$. Therefore, there are runs of $\mathcal{A}_p$ on y from p to p and from p to q, and a run of $\mathcal{A}_q$ on y from q to q. Hence, $\mathcal{A}$ contains a forbidden pattern for bounded ambiguity.

Case 2: $q \in Q' \setminus Q$. By the definition of Q', there are $c \in \mathbb{C}_{even}$ and $q' \in Q$ such that $q = (c, q')$. If $p \in Q' \setminus Q$ then there are $c' \in \mathbb{C}_{even}$ and $p' \in Q$ such that $p = (c', p')$. Notice that there is a run from (c', p') to (c, q') and by the definition of δ' we have $c = c'$. Since $p \neq q$ we conclude that $p' \neq q'$. Therefore, there are runs of $\mathcal{A}_{p'}$ on y from p' to p' and from p' to q', and a run of $\mathcal{A}_{q'}$ on y from q' to q'. We conclude that $\mathcal{A}$ contains the forbidden pattern for bounded ambiguity.

If $p \in Q$ and $p \neq q'$, then there are runs of $\mathcal{A}_p$ on y from p to p and from p to q', and a run of $\mathcal{A}_{q'}$ on y from q' to q'. We conclude that $\mathcal{A}$ contains the forbidden pattern for bounded ambiguity. Otherwise, we have $p = q'$. The run ρ_2 is from a state in Q to a state in $Q' \setminus Q$. By the definition of δ' we conclude that there is a run ρ' of $\mathcal{A}_p$ on y from p to p which passes though a state with color greater than $\mathbb{C}(q)$. The run ρ_3 is from a state in $Q' \setminus Q$ to a state in $Q' \setminus Q$, and by the definition of δ' we conclude that there is a run ρ'' of $\mathcal{A}_p$ on y from p to p which only visits states with color not greater than $\mathbb{C}(q)$. Let $\rho' = p_0, \ldots, p_n$ and $\rho'' = q_0, \ldots, q_n$. We conclude that $p_0 = p_n = q_0 = q_n = p$, and $\rho' \neq \rho''$. Therefore, there is $0 < i < n$ such that $p_i \neq q_i$. Let y_1, y_2 be two finite words such that y_1 is the prefix of y of length i, and $y_1 \cdot y_2 = y$. We conclude that there are runs of $\mathcal{A}_{p_i}$ on $y_2 \cdot y_1$ from p_i to p_i and from q_i to q_i, and there is a run of $\mathcal{A}_{q_i}$ on $y_2 \cdot y_1$ from q_i to q_i. Therefore, $\mathcal{A}$ contains the forbidden pattern for bounded ambiguity, as requested. ◀

We will now show that the problem of deciding the degree of ambiguity of PWA is in PTIME (in fact, this problem is even in NL).

▶ **Definition 19.** *Given a PWA $\mathcal{A} = (Q, \Sigma, Q_I, \delta, \mathbb{C})$ and $k > 0$, we define a graph $G^k_{\mathcal{A}}$ where the set of nodes is Q^k, and there is an edge from $(q_1, \ldots, q_k)$ to $(p_1, \ldots, p_k)$ iff there is $a \in \Sigma$ such that $(q_i, a, p_i) \in \delta$ for all $1 \leq i \leq k$.*

The following lemma supplies equivalent conditions to the forbidden patterns presented in Definition 14:

▶ **Lemma 20.**

- *$\mathcal{A}$ contains a forbidden pattern for countable ambiguity iff there is a state q such that $\mathbb{C}(q)$ is even, and $G^2_{\mathcal{A}}$ contains a path from (q, q) to itself which passes through a node (p, p') where $p \neq p'$, and for each node (p_1, p_2) in the path, $\mathbb{C}(q) \geq \mathbb{C}(p_1)$ and $\mathbb{C}(q) \geq \mathbb{C}(p_2)$.*
- *$\mathcal{A}$ contains a forbidden pattern for finite ambiguity iff there are states q and p such that $\mathbb{C}(q)$ is even, and $G^3_{\mathcal{A}}$ contains a path from (p, p, q) to (p, q, q) such that $\mathbb{C}(q)$ is the maximal color which is assigned to a state in the third coordinate of each node in the path, and the path contains a node (p_1, p_2, p_3) where $p_1 \neq p_2$.*
- *$\mathcal{A}$ contains a forbidden pattern for bounded ambiguity, if there are two states $p \neq q$ such that $G^3_{\mathcal{A}}$ contains a path from (p, p, q) to (p, q, q).*

▶ **Proposition 21.** *There is a polynomial time algorithm for deciding the degree of ambiguity of PWA.*

Proof. Let $\mathcal{A} = (Q, \Sigma, Q_I, \delta, \mathbb{C})$ be a PWA. By Lemma 20, it is sufficient to show that the equivalent conditions on $G^2_{\mathcal{A}}$ and $G^3_{\mathcal{A}}$ are decidable in polynomial time.

Indeed, each of the conditions in Lemma 20 can be reduced to polynomially many reachability problems in $G^2_{\mathcal{A}}$ and $G^3_{\mathcal{A}}$. Constructing $G^2_{\mathcal{A}}$ and $G^3_{\mathcal{A}}$ can be done in polynomial time in the size of $\mathcal{A}$, and the proposition follows. ◀

6 Finite Ambiguity and Bounded Ambiguity of PTA

In this section we will provide characterizations for finite ambiguity and bounded ambiguity of PTA. These characterizations are similar to the characterizations for finite ambiguity and bounded ambiguity of BTA (see Propositions 17 and 18 in [11]).

▶ **Definition 22** (Projection of a computation on a branch). *Let $\phi \in ACC(\mathcal{A}, t)$ and let $\pi := v_0, v_1, \ldots$ be a tree branch. We say that $\phi(\pi) := \phi(v_0), \phi(v_1), \dots \in Q^{\omega}_{\mathcal{A}}$ is the projection of ϕ on π, and define $ACC(\mathcal{A}, t, \pi) := \{\phi(\pi) \mid \phi \in ACC(\mathcal{A}, t)\}$.*

▶ **Definition 23** (Branch ambiguity). *$\mathcal{A}$ is at most n branch-ambiguous if $|ACC(\mathcal{A}, t, \pi)| \leq n$ for every t and branch π. $\mathcal{A}$ is bounded branch ambiguous if it is at most n branch ambiguous for some n. $\mathcal{A}$ is finitely (respectively, countably) branch ambiguous if $|ACC(\mathcal{A}, t, \pi)|$ is finite (respectively, countable) for every t and π.*

Let $\mathcal{A}$ be a PTA. We define a PWA $\mathcal{A}_B$ which has the same ambiguity as the branch ambiguity of $\mathcal{A}$:

▶ **Definition 24** (Branch automaton). *For a PTA $\mathcal{A} = (Q, \Sigma, Q_I, \delta, \mathbb{C})$, the corresponding branch automaton is a PWA $\mathcal{A}_B := (Q, \Sigma_B, Q_I, \delta_B, \mathbb{C})$, where*

1. *$\Sigma_B := \Sigma \times \Sigma_d \times \Sigma_{cons}$ with*
   a. *$\Sigma_d := \{l, r\}$ - directions alphabet (left/right).*
   b. *$\Sigma_{cons} := \{S \subseteq Q \mid \bigcap_{q \in S} L(\mathcal{A}_q) \neq \emptyset\}$ - sets of states, which we consider "consistent."*
2. *$(q, a, q') \in \delta_B$ iff $a = (\sigma, l, S)$ and $\exists p \in S : (q, \sigma, (q', p)) \in \delta$ or $a = (\sigma, r, S)$ and $\exists p \in S : (q, \sigma, (p, q')) \in \delta$.*

The following lemma reduces the branch ambiguity to the ambiguity of branch automaton.

▶ **Lemma 25.** *The branch ambiguity of a PTA $\mathcal{A}$ is bounded (respectively, finite, countable) iff the ambiguity of the corresponding branch ω-automaton $\mathcal{A}_B$ is bounded (respectively, finite, countable).*

The proof of the lemma, which appears in the appendix, is a minor modification of the proof of Lemma 11 in [11] which reduces the branch ambiguity of BTA to the ambiguity of the corresponding branch automaton.

▶ **Definition 26** (Ambiguous Transition Pattern). *Let $\mathcal{A} = (Q, \Sigma, Q_I, \delta, \mathbb{C})$ be a PTA with corresponding branch automaton $\mathcal{A}_B = (Q, \Sigma_B, Q_I, \delta_B, \mathbb{C})$. $\mathcal{A}$ has a* ***q-ambiguous transition pattern*** *if $q \in Q$ and there are $p_1, p_2 \in Q$ and $y_1 \in \Sigma_B^*$, $y_2 \in \Sigma_B^+$ with runs of $\mathcal{A}_B$ from q to p_1 on y_1 and from p_2 to q on y_2 such that at least one of the following holds:*

1. *There are two transitions $(p_1, (a, d, \{q_1\}), p_2), (p_1, (a, d, \{q_2\}), p_2) \in \delta_B$ with $q_1 \neq q_2$ and $L(\mathcal{A}_{q_1}) \cap L(\mathcal{A}_{q_2}) \neq \emptyset$, or*
2. *There is a transition $(p_1, (a, d, \{q_1\}), p_2) \in \delta_B$ with $da(\mathcal{A}_{q_1}) > 1$.*

A q-ambiguous transition pattern is said to be ***fine*** *if $\mathbb{C}(q)$ is even, and $\mathbb{C}(q) \geq \mathbb{C}(p)$ for each state p in the runs of $\mathcal{A}_B$ from q to p_1 on y_1 and from p_2 to q on y_2.*

$\mathcal{A}$ is said to have an ***ambiguous transition pattern*** *if there is $q \in Q$ such that $\mathcal{A}$ has a q-ambiguous transition pattern. $\mathcal{A}$ is said to have a* ***fine ambiguous transition pattern*** *if there is $q \in Q$ such that $\mathcal{A}$ has a fine q-ambiguous transition pattern.*

We are now ready to provide the characterization of finite and bounded ambiguity of parity tree automata.

▶ **Proposition 27** (Bounded ambiguity of parity automata). *The following are equivalent:*

1. *A PTA $\mathcal{A}$ is not boundedly ambiguous.*
2. *At least one of the following conditions holds:*
   - **a.** *$\mathcal{A}$ is not bounded branch ambiguous.*
   - **b.** *$\mathcal{A}$ has an ambiguous transition pattern.*

▶ **Proposition 28** (Finite ambiguity of parity automata). *The following are equivalent:*

1. *A PTA $\mathcal{A}$ is not finitely ambiguous.*
2. *At least one of the following conditions holds:*
   - **a.** *$\mathcal{A}$ is not finitely branch ambiguous.*
   - **b.** *$\mathcal{A}$ has a fine ambiguous transition pattern.*

The proofs of Propositions 28 and 27 are simple variations of the proofs of Propositions 17 and 18 of [11], which deal with the characterization of bounded and finite ambiguity of Büchi tree automata. See the appendix for the proof of Prop. 28.

In Section 7 we use Propositions 27 and 28 to show that the problems of deciding whether a PTA is not boundedly ambiguous/not finitely ambiguous are in co-NP.

7 co-NP Upper Bound of Theorem 1

Let $\mathcal{A} = (Q, \Sigma, Q_I, \delta, \mathbb{C})$ be a PTA. Deciding whether a state $q \in Q$ of a PTA is useful is reducible to the emptiness problem of PTA, and can be tested in NP$\cap$ co-NP as follows: Let $Q_{non-empty} := \{p \mid L(\mathcal{A}_p) \neq \emptyset\}$. If $q \notin Q_{non-empty}$, then q is not useful. Otherwise, let $\mathcal{B}$ be the restriction of the branch automaton $\mathcal{A}_B$ to the transitions over one state letters in $\Sigma \times \Sigma_d \times \{\{p\} \mid p \in Q_{non-empty}\}$. Now, q is reachable from an initial state of $\mathcal{B}$ iff it is useful.

Therefore, we will assume in the rest of the proof that all states in $\mathcal{A}$ are useful.

It is easy to verify that $\mathcal{A}$ is ambiguous iff there exist two states $p, q \in Q_I$ such that $L(\mathcal{A}_p) \cap L(\mathcal{A}_q) \neq \emptyset$ or there exist two transitions $(q, a, q_1, q_2), (q, a, q'_1, q'_2) \in \delta$ from a state q such that $L(\mathcal{A}_{q_1}) \cap L(\mathcal{A}_{q'_1}) \neq \emptyset$ and $L(\mathcal{A}_{q_2}) \cap L(\mathcal{A}_{q'_2}) \neq \emptyset$.

Since deciding whether $L(\mathcal{A}_p) \cap L(\mathcal{A}_q) \neq \emptyset$ is in co-NP (by Lemma 8), and the number of pairs $p, q \in Q$ is polynomial in $|\mathcal{A}|$, we conclude:

▶ **Lemma 29.** *Deciding whether a PTA is ambiguous is in co-NP.*

The following Lemma easily follows from Definition 24 of the branch automaton.

▶ **Lemma 30.** *Let $\mathcal{A}_B$ be the branch automaton of $\mathcal{A}$. Assume that $r_i \in Q^{l+1}$ for $i = 1, \dots, k$ are runs of $\mathcal{A}_B$ on $u = (\sigma_1, d_1, S_1) \dots (\sigma_l, d_l, S_l) \in \Sigma_B^*$. Then, for $i = 1, \dots, l$ there are $S'_i \subseteq S_i$ such that $|S'_i| \leq k$ and r_i for $i = 1, \dots, k$ are runs of $\mathcal{A}_B$ on $u = (\sigma_1, d_1, S'_1) \dots (\sigma_l, d_l, S'_l)$.*

A letter $(\sigma, d, S) \in \Sigma_B$ is called a k-state letter if S has at most k states. If $\mathcal{A}$ has n states, then the alphabet Σ_B of the branch automaton $\mathcal{A}_B$ might be of size $2|\Sigma| \times 2^n$, yet the number of k-state letters is bounded by $2|\Sigma| \times \sum_{i=1}^{k} \binom{n}{i} \leq 2|\Sigma| n^k$. To test whether a k-state letter (σ, d, S) is in Σ_B, we can check whether the intersection of the tree languages $L(\mathcal{A}_q)$ for $q \in S$ is non-empty. By Lemma 8, this is in co-NP for every fixed $k \in \mathbb{N}$. We denote by $\mathcal{A}_B^{(k)}$ the restriction of the branch automaton $\mathcal{A}_B$ to k-state letters.

▶ **Lemma 31** (Computability of branch ambiguity)**.** *The problem whether the branch ambiguity of $\mathcal{A}$ is not bounded (respectively, not finite, not countable) is in co-NP.*

Proof. By Lemma 25 and Proposition 16, deciding whether $\mathcal{A}_B$ is not finitely/not boundedly ambiguous is equivalent to deciding whether $\mathcal{A}_B$ has a forbidden pattern for finite/bounded ambiguity. The forbidden patterns involve conditions on at most three runs on the same word. By Lemma 30, we conclude that $\mathcal{A}_B$ has a forbidden pattern for finite/bounded ambiguity iff $\mathcal{A}_B^{(3)}$ has a forbidden pattern for finite/bounded ambiguity. Finding $\mathcal{A}_B^{(3)}$ requires deciding the non-emptiness of $L(\mathcal{A}_{q_1}) \cap L(\mathcal{A}_{q_2}) \cap L(\mathcal{A}_{q_3})$ for all triplets $q_1, q_2, q_3 \in Q$. This problem is in co-NP by Lemma 8; hence, $\mathcal{A}_B^{(3)}$ can be constructed in co-NP and its size is polynomial in the size of $\mathcal{A}$. The problem of deciding if $\mathcal{A}_B^{(3)}$ has a forbidden pattern for bounded/finite/countable ambiguity is in PTIME in the size of $\mathcal{A}_B^{(3)}$ (by Lemma 21). All these imply the co-NP bound of Lemma 31. ◀

▶ **Lemma 32** (Computability of q-ambiguous pattern)**.** *Deciding whether $\mathcal{A}$ has a q-ambiguous (respectively, fine q-ambiguous) transition for a state $q \in Q$ is in co-NP.*

Proof. Deciding if item (1) or (2) of Definition 26 holds for a fixed pair of states requires testing the non-emptiness of PTA intersection, which is in co-NP by Lemma 8. $\mathcal{A}$ has a q-ambiguous transition pattern iff there is a path in $\mathcal{A}_B$ from q to p_1 and from p_1 to q, such that items (1) or (2) hold for (p_1, p_2). If, additionally, $\mathbb{C}(q)$ is even and all states in the paths have a color which is not greater than $\mathbb{C}(q)$, then $\mathcal{A}$ has a fine q-ambiguous transition. Both these cases are reducible to the reachability problem in $\mathcal{A}_B^{(1)}$. Assuming all states are useful, finding $\mathcal{A}_B^{(1)}$ can be done in polynomial time. ◀

Lemmas 25, 31 and 32 imply that deciding whether condition 2(a) and 2(b) of Propositions 28 and 27 are in co-NP. Therefore, the problem whether a PTA is not boundedly (respectively, finitely) ambiguous is in co-NP. This together with Lemma 29 prove the upper bounds of Theorem 1.

8 Conclusion

We investigated the complexity of deciding the degree of ambiguity for PTA. The co-NP hardness lower bound was obtained by reductions from multi-dimensional parity games [2]. The co-NP upper bound was obtained by structural characterizations of degrees of ambiguity for PTA which is similar to the corresponding characterizations for BTA [11]. Unfortunately, we have not succeeded to find a characterization and an upper bound for countable ambiguity. It is also interesting to find natural problems for PTA/BTA which are easier for PTA/BTA with small degrees of ambiguity than for arbitrary PTA/BTA.

References

1 Vince Bárány, Łukasz Kaiser, and Alex Rabinovich. Expressing cardinality quantifiers in monadic second-order logic over trees. *Fundamenta Informaticae*, 100(1-4):1–17, 2010.

2 Krishnendu Chatterjee, Thomas A Henzinger, and Nir Piterman. Generalized parity games. In *International Conference on Foundations of Software Science and Computational Structures*, pages 153–167. Springer, 2007.

3 Thomas Colcombet. Unambiguity in automata theory. In *International Workshop on Descriptional Complexity of Formal Systems*, pages 3–18. Springer, 2015.

4 Yo-Sub Han, Arto Salomaa, and Kai Salomaa. Ambiguity, nondeterminism and state complexity of finite automata. *Acta Cybernetica*, 23(1):141–157, 2017.

5 Jozef Jirásek, Galina Jirásková, and Juraj Šebej. Operations on unambiguous finite automata. In *International Conference on Developments in Language Theory*, pages 243–255. Springer, 2016.

6 Ernst Leiss. Succinct representation of regular languages by boolean automata. *Theoretical computer science*, 13(3):323–330, 1981.

7 Hing Leung. Descriptional complexity of nfa of different ambiguity. *International Journal of Foundations of Computer Science*, 16(05):975–984, 2005.

8 Christof Löding and Anton Pirogov. On finitely ambiguous büchi automata. In Mizuho Hoshi and Shinnosuke Seki, editors, *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, volume 11088 of *Lecture Notes in Computer Science*, pages 503–515. Springer, 2018. doi:10.1007/978-3-319-98654-8_41.

9 Dominique Perrin and Jean-Éric Pin. *Infinite words: automata, semigroups, logic and games*, volume 141. Academic Press, 2004.

10 Alexander Rabinovich. Complementation of finitely ambiguous Büchi automata. In *International Conference on Developments in Language Theory*, pages 541–552. Springer, 2018.

11 Alexander Rabinovich and Doron Tiferet. Degrees of ambiguity of büchi tree automata. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 50:1–50:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.50.

12 Helmut Seidl. On the finite degree of ambiguity of finite tree automata. *Acta Informatica*, 26(6):527–542, 1989.

13 Richard Edwin Stearns and Harry B Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal on Computing*, 14(3):598–611, 1985.

14 Wolfgang Thomas. Automata on infinite objects. In *Formal Models and Semantics*, pages 133–191. Elsevier, 1990.

15 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991.

A Selected Proofs

A.1 Proof of Lemma 25

▶ **Lemma 25.** *The branch ambiguity of a PTA $\mathcal{A}$ is bounded (respectively, finite, countable) iff the ambiguity of the corresponding branch ω-automaton $\mathcal{A}_B$ is bounded (respectively, finite, countable).*

The proof will make use the following two lemmas, which deals with the connection between computations of $\mathcal{A}$ and runs of $\mathcal{A}_B$:

▶ **Lemma 33.** *Let $t \in L(\mathcal{A})$, and let $\pi = v_0, v_1, \ldots$ be a tree branch. Then there exists $y \in L(\mathcal{A}_B)$ such that $ACC(\mathcal{A}, t, \pi) \subseteq ACC(\mathcal{A}_B, y)$.*

Proof. Let $y := (a_1, d_1, S_1) \ldots (a_i, d_i, S_i) \ldots$ be an ω-word over alphabet Σ_B, such that:

- $d_i \in \{l, r\}$ and $d_i := l$ iff v_i is the left child of v_{i-1}
- $a_i := t(v_{i-1})$
- $S_i := \{\phi(v_i') \mid \phi \in ACC(\mathcal{A}, t)\}$ where v_i' is the child of v_{i-1} which is not v_i

Let $\phi \in ACC(\mathcal{A}, t)$. We will prove that $\rho := \phi(\pi)$ is a run of $\mathcal{A}_B$ on y. Assume that $\rho = p_0, p_1, \ldots$. For each $1 \leq i \leq n$ we have $p_{i-1} = \phi(v_{i-1})$ and $p_i = \phi(v_i)$. If v_i is the left child of v_{i-1} then we obtain $(\phi(v_{i-1}), t(v_{i-1}), \phi(v_i), \phi(v_{i-1}')) \in \delta$, and otherwise $(\phi(v_{i-1}), t(v_{i-1}), \phi(v_{i-1}'), \phi(v_i)) \in \delta$. By definition of S_i we obtain $\phi(v_{i-1}') \in S_i$. Notice that $a_i = t(v_{i-1})$, and $d_i = l$ iff v_i is the left child of v_{i-1}. Therefore, by definition of $\mathcal{A}_B$, we conclude that $(\phi(v_{i-1}), (a_i, d_i, S_i), \phi(v_i)) = (p_{i-1}, (a_i, d_i, S_i), p_i) \in \delta_B$, and the lemma follows. ◀

▶ **Lemma 34.** *Let $\mathcal{A}$ be a PTA with corresponding branch automaton $\mathcal{A}_B$. If $y = (a_1, d_1, S_1) \ldots (a_i, d_i, S_i) \ldots$ is an ω-word over alphabet Σ_B such that $y \in L(\mathcal{A}_B)$, then there exist a tree $t \in L(\mathcal{A})$ and a tree branch $\pi = v_0, v_1, \ldots$ such that:*

- *$t(v_i) = a_{i+1}$*
- *v_{i+1} is the left child of v_i iff $d_i = l$*
- *For each run $\rho \in ACC(\mathcal{A}_B, y)$ there is a computation $\phi \in ACC(\mathcal{A}, t)$ such that $\phi(\pi) = \rho$.*

Proof. For each S_i, let $t_i \in \bigcap_{q \in S_i} L(\mathcal{A}_q)$ (there is such t_i, since $S_i \in \Sigma_{cons}$).

Let $\pi := v_0, v_1, \ldots$ where $v_0 := \epsilon$ and $\forall i \in \mathbb{N} : v_{i+1} := v_i \cdot d_i$; and denote by v_i' be the child of v_i which is not v_{i+1}.

We define a Σ-labeled full-binary tree t by $t(u) := \begin{cases} a_{i+1} & \exists i : u = v_i \\ t_{i+1}(w) & \exists i : u = v_i' \cdot w \end{cases}$

Let $\rho := p_0, p_1, \ldots$ be an accepting run of $\mathcal{A}_B$ on y. By definition of $\mathcal{A}_B$, for each $i \in \mathbb{N}$ there is a state $q_i \in Q$ such that $(p_i, a_i, p_{i+1}, q_i) \in \delta$ if $d_i = l$ or $(p_i, a_i, q_i, p_{i+1}) \in \delta$ if $d_i = r$. Recall that $t_i \in L(\mathcal{A}_{q_i})$, and therefore there is a computation $\phi_i \in ACC(\mathcal{A}_{q_i}, t_i)$. We use ρ and ϕ_i to define a computation ϕ of $\mathcal{A}$ on t, as follows:

$$\phi(u) := \begin{cases} p_i & \exists i : u = v_i \\ \phi_{i+1}(w) & \exists i : u = v_i' \cdot w \end{cases}$$

It is easy to see that ϕ is a computation of $\mathcal{A}$ on t. We will show that ϕ is accepting. For each tree branch π', if $\pi' = \pi$ then $\phi(\pi') = \phi(\pi) = \rho$ and since $\rho \in ACC(\mathcal{A}_B, y)$ we conclude that the maximal color which is assigned infinitely often to a state in $\phi(\pi')$ is even. Otherwise, by definition of t, there is $i \in \mathbb{N}$ such that $v_i' \in \pi'$. By the definition of ϕ we obtain $\phi(u) = \phi_i(w)$ for all nodes $u = v_i' \cdot w$, and since ϕ_i is accepting we conclude that the maximal color which is assigned infinitely often to a state in $\phi(\pi')$ is also even. Hence, $\phi \in ACC(\mathcal{A}, t)$ as requested. ◀

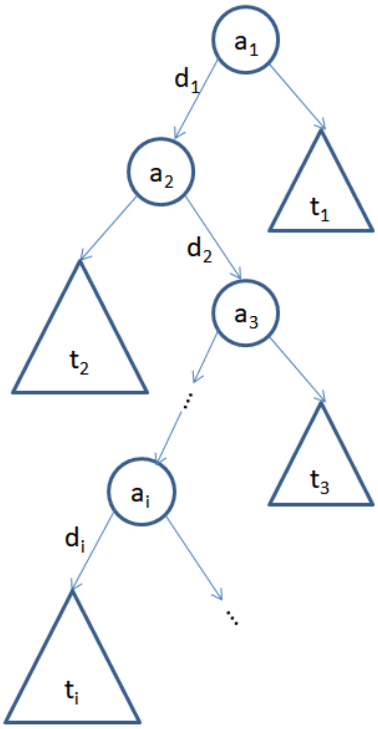


Figure 3 The tree t.

We are now ready to prove Lemma 25.

$\Rightarrow$: By Lemma 33, for each tree $t \in L(\mathcal{A})$ and a tree branch π there is an ω-word $y \in L(\mathcal{A}_B)$ such that $ACC(\mathcal{A}, t, \pi) \subseteq ACC(\mathcal{A}_B, y)$. Therefore, if $\mathcal{A}$ is not boundedly (respectively, finitely, countably) branch ambiguous then $\mathcal{A}_B$ is not boundedly (respectively, finitely, countably) ambiguous.

$\Leftarrow$: By Lemma 34, for each $y \in L(\mathcal{A}_B)$ there is a tree $t \in L(\mathcal{A})$ and a tree branch π such that $ACC(\mathcal{A}_B, y) \subseteq ACC(\mathcal{A}, t, \pi)$. Therefore, if $\mathcal{A}_B$ is not boundedly (respectively, finitely, countably) ambiguous then $\mathcal{A}$ is not boundedly (respectively, finitely, countably) branch ambiguous. ◀

A.2 Proof of Proposition 28

▶ **Proposition 28** (Finite ambiguity of parity automata). *The following are equivalent:*

1. *A PTA $\mathcal{A}$ is not finitely ambiguous.*
2. *At least one of the following conditions holds:*
   a. *$\mathcal{A}$ is not finitely branch ambiguous.*
   b. *$\mathcal{A}$ has a fine ambiguous transition pattern.*

We will first prove the following auxiliary lemma:

▶ **Lemma 35.** *If a PTA $\mathcal{A}$ has a fine ambiguous transition pattern then its ambiguity degree is not countable.*

Proof. Let $\mathcal{A}_B = (Q, \Sigma_B, Q_I, \delta_B, F)$ be the corresponding branch automaton of $\mathcal{A}$, and let q be a state such that $\mathcal{A}$ has a fine q-ambiguous transition pattern. Therefore, there exist $p'_1, p'_2 \in Q$ and $z_1 \in \Sigma_B^*$, $z_2 \in \Sigma_B^+$ such that there is a run ρ_1 of $(\mathcal{A}_B)_q$ on z_1 from q to p'_1, and a run ρ_2 of $(\mathcal{A}_B)_{p'_2}$ on z_2 from p'_2 to q; $\mathbb{C}(q)$ is even, and $\mathbb{C}(q) > \mathbb{C}(p)$ for each state p in the runs ρ_1 and ρ_2.

We choose $z' \in \Sigma_B$ as follows:

- If there are transitions $(p'_1, (a', d', \{q_1\}), p'_2), (p'_1, (a', d', \{q_2\}), p'_2) \in \delta_B$ with $L(\mathcal{A}_{q_1}) \cap L(\mathcal{A}_{q_2}) \neq \emptyset$, then by definition of $\mathcal{A}_B$ there exists a transition $(a', d', \{q_1, q_2\}) \in \delta_B$. Let $z' := (a', d', \{q_1, q_2\})$.
- Otherwise, by definition of fine q-ambiguous transition, there exists a transition: $(p'_1, (a', d', \{q_1\}), p'_2) \in \delta_B$ with $da(\mathcal{A}_{q_1}) > 1$. In this case, let $z' := (a', d', \{q_1\})$.

Define a word $y := z_1 \cdot z' \cdot z_2$ over alphabet Σ_B, and let $\rho := \rho_1 \cdot \rho_2$. Notice that ρ is a run of $\mathcal{A}_B$ on y from q to q.

y^ω is an ω-word in Σ_B^ω. Denote by ρ' the run ρ without the last state. By definition of ρ we conclude that $(\rho')^\omega$ is a run of $(\mathcal{A}_B)_f$ on y^ω. Notice that ρ' contains a final state, and therefore $(\rho')^\omega$ is an accepting run, and $y^\omega \in L((\mathcal{A}_B)_f)$.

y^ω is of the form $(a_1, d_1, S_1) \dots (a_i, d_i, S_i) \dots$ where $a_i \in \Sigma$, $d_i \in \{l, r\}$ and $S_i \subseteq Q$. Assume $z' = (a_z, d_z, S_z)$, and let $t_z \in \bigcap_{q' \in S_z} L(\mathcal{A}_{q'})$ such that there are two accepting computations ϕ_1 and ϕ_2 on t_z, where $\phi_1(\epsilon), \phi_2(\epsilon) \in S_z$ (there is such t_z by definition of z').

By Lemma 34, there is a tree $t \in L(\mathcal{A}_q)$, a computation $\phi \in ACC(\mathcal{A}_q, t)$ and a tree branch $\pi = v_0 v_1 \dots$ such that $\phi(\pi) = (\rho')^\omega$; and for each $i \in \mathbb{N}$ we have $t(v_i) = a_i$, and v_{i+1} is the left child of v_i iff $d_i = l$.

Let $J := \{i \mid \text{the } i\text{-th transition of } (\rho')^\omega \text{ is from } p'_1 \text{ to } p'_2 \text{ over } z'\}$. By definition of ρ we conclude that J is an infinite subset of $\mathbb{N}$.

Define a tree t' by $t'(u) = \begin{cases} t_z(w) & \exists i : u = v'_i \cdot w \\ t(u) & \text{otherwise} \end{cases}$

For each $B \subseteq A$, we define a computation ϕ_B by:

$$\phi_B(u) = \begin{cases} \phi_1(w) & \exists i : u = v'_i \cdot w \text{ and } v'_i \in A \setminus B \\ \phi_2(w) & \exists i : u = v'_i \cdot w \text{ and } v'_i \in B \\ \phi(u) & \text{otherwise} \end{cases}$$

It is routine to verify that ϕ_B is an accepting computation of $\mathcal{A}_q$ on t', and that $B_1 \neq B_2 \to \phi_{B_1} \neq \phi_{B_2}$. Since the number of subsets of A is uncountable, and each subset B yields a unique accepting computations ϕ_B of $\mathcal{A}_q$ on t', it follows that $ACC(\mathcal{A}_q, t')$ is not countable, and since q is useful, we conclude that $\mathcal{A}$ is not countably ambiguous. ◀

We are now ready to proceed with the proof of Prop. 28. The (2) ⇒ (1) direction follows from Lemma 25 and Lemma 35. Below we prove the (1) ⇒ (2) direction.

Let t be a tree such that $ACC(\mathcal{A}, t)$ is not finite. We define a branch $\pi := v_0, \dots, v_i, \dots$ in t and an ω-sequence of states $q_0 \dots q_i \dots$ such that for every i:

1. From q_i there are infinitely many accepting computations of $\mathcal{A}_{q_i}$ on the subtree $t_{\geq v_i}$.
2. There is an accepting computation ϕ_i on t such that $\phi_i(v_j) = q_j$ for every $j \leq i$.

Define v_0 as the root of t and q_0 as an initial state from which there are infinitely many accepting computations.

Assume that v_i and q_i were defined. Since there are infinitely many accepting computations from the state q_i on the subtree $t_{\geq v_i}$, infinitely many of them take the same first transition from q_i to $\langle q_l, q_r \rangle$ and either there are infinitely many accepting computations from state q_l on the subtree rooted at the left child of v_i, or from state q_r on the subtree rooted at the right child of v_i. Define v_{i+1} and q_{i+1} according to these cases.

If $|ACC(\mathcal{A}, t, \pi)|$ is infinite, then by the definition of branch ambiguity we have that $\mathcal{A}$ is not finitely branch ambiguous, and 2(a) holds. Otherwise, there exist $\phi_1, \dots, \phi_k \in ACC(\mathcal{A}, t)$ such that $ACC(\mathcal{A}, t, \pi) = \{\phi_i(\pi) \mid 1 \leq i \leq k\}$. Choose n such that for all $1 \leq i < j \leq k$: $\phi_i(v_0) \dots \phi_i(v_n) \neq \phi_j(v_0) \dots \phi_j(v_n)$.

There is $1 \leq j \leq k$ such that $\phi_j(v_0) \dots \phi_j(v_n) = q_0 \dots q_n$. Notice that by definition of n, each computation $\phi \in ACC(\mathcal{A}, t)$ which assigns $q_0, \dots, q_n$ to the nodes $v_0, \dots, v_n$ must also agree with ϕ_j on each node v_i for $i \in \mathbb{N}$. Therefore, again by the definition of π and $q_0, \dots q_i, \dots$, we conclude that $\phi_j(\pi) = q_0, q_1, \dots$.

Let q be a state which occurs infinitely often in $\phi_j(\pi)$ such that $\mathbb{C}(q)$ is even, and $\mathbb{C}(q)$ is maximal among the colors which $\mathbb{C}$ assigns to states which occurs infinitely often in $\phi_j(\pi)$.

Choose $N > n$ such that $\phi_j(v_N) = q_N = q$, and each state q_i where $i \geq N$ occurs infinitely often in $\phi_j(\pi)$. By selection of q_N, there are infinitely many accepting computations of $\mathcal{A}_q$ on $t_{\geq v_N}$. Take two different accepting computations $\phi', \phi'' \in ACC(\mathcal{A}_q, t_{\geq v_N})$. Note that $\forall i \geq N : \phi_j(v_i) = \phi'(v_i) = \phi''(v_i) = q_i$. Therefore, ϕ' and ϕ'' differ at some node $w \notin \pi$, and there exists $M > N$ such that $\phi_j(v_M) = q = \phi'(v_M) = \phi''(v_M)$ and $v_M \perp w$.

Let u be the node of maximal depth on the path from v_N to v_M such that $w > u$. Let u', u'' be the children of u such that $w \geq u'$. Assume w.l.o.g. that u' is the left child of u.

Look at the transitions $(\phi'(u), t(u), \phi'(u'), \phi'(u'')), (\phi''(u), t(u), \phi''(u'), \phi''(u'')) \in \delta$. Since $u'' \in \pi$, we have $\phi'(u'') = \phi''(u'')$. If $\phi'(u') = \phi''(u')$ then the restriction of ϕ' and ϕ'' on $t_{\geq u'}$ are two different computations in $ACC(\mathcal{A}_{\phi(u')}, t_{\geq u'})$ and therefore $da(\mathcal{A}_{\phi(u')}) > 1$ and condition 2 of q-ambiguous transition pattern definition applies. Otherwise, we have $\phi'(u') \neq \phi''(u')$ and $t_{\geq u'} \in L(\mathcal{A}_{\phi(u')}) \cap L(\mathcal{A}_{\phi'(u')})$ and therefore condition 1 of q-ambiguous transition pattern definition applies. By selection of q we conclude that $\mathcal{A}$ has a fine q-ambiguous transition pattern, and condition 2(b) of Prop. 28 holds. ◀

On Flat Lossy Channel Machines

Philippe Schnoebelen
LSV, CNRS, ENS Paris-Saclay, Université Paris-Saclay
http://www.lsv.fr/~phs
phs@lsv.fr

Abstract

We show that reachability, repeated reachability, nontermination and unboundedness are NP-complete for Lossy Channel Machines that are *flat*, i.e., with no nested cycles in the control graph. The upper complexity bound relies on a fine analysis of iterations of lossy channel actions and uses compressed word techniques for efficiently reasoning with paths of exponential lengths. The lower bounds already apply to acyclic or single-path machines.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Infinite state systems, Automated verification, Flat systems, Lossy channels

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.37

Funding Supported by ANR project BRAVAS (grant ANR-17-CE40-0028).

Acknowledgements We thank A. Finkel who raised the issue of flatness in lossy channel systems. We also thank J. Leroux and S. Halfon for useful comments that helped improve this paper.

1 Introduction

Lossy channel machines. Lossy channel machines, aka LCMs, are FIFO automata, i.e., finite-state machines operating on buffers with FIFO read/write discipline, where the buffers are *unreliable*, or *lossy*, in the sense that letters (or "messages") in a buffer can be lost nondeterministically at any time.

LCMs were first introduced as a model for communication protocols designed to work properly in unreliable environments. They immediately attracted interest because, unlike FIFO automata with reliable buffers, they have decidable safety and termination problems [21, 4, 11, 2]. It was later found that LCMs are a relevant computational model *per se*, useful for verifying timed automata [3, 34], modal logics [26], etc., and connected to other problems in computer science [31, 14, 39].

Flat LCMs. In this paper we consider the case of *flat LCMs*, i.e., LCMs where the control graph has no nested cycles. In the area of infinite-state systems verification, flat systems were first considered in [25, 15] for counter systems[1]. In addition, some earlier "loop acceleration" results, e.g. [6], where one can compute reachability sets along a cycle, can often be generalised to flat systems. Positive results on flat counter systems can be found in [36, 9, 18, 8, 35, 17], and in [27] for counter systems with recursive calls. Regarding flat FIFO automata, verification was shown decidable by Bouajjani and Habermehl [7] who improved on earlier results by Boigelot [5], and the main verification problems were only recently proven to be NP-complete [20, 22]. These results have applications beyond flat systems in the context of *bounded verification* techniques, where one analyses a bounded subset of the runs of a general system [20].

[1] Flatness remains relevant with *finite-state systems*, see e.g., [33]. This is especially true when one is considering the verification of properties expressed in a rich logic as in, e.g., [16]. In language theory, flat finite-state automata correspond to regular languages of polynomial density, sometimes called *sparse languages*, or also *bounded languages*.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 37; pp. 37:1–37:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Flat LCMs have not been explicitly considered in the literature. They are implicit in forward analysis methods based on loop acceleration, starting with [2], but these works do not address the overall complexity of the verification problem, only the complexity of elementary operations.

It is not clear whether one should expect flat LCMs to be simpler than flat FIFO automata (on account of unrestricted LCMs being simpler than the Turing powerful, unrestricted FIFO automata), or if they could be more complex since message losses introduce some nondeterminism that does not occur when one follows a fixed cycle in a FIFO system. Indeed, message losses can be seen as hidden implicit loops that disrupt the apparent flatness of the LCM.

Our contribution. We analyse the behaviour of the backward-reachability algorithm on cycles of lossy channel actions and establish a bilinear upper bound on its complexity. As a consequence, reachability along runs of the form $\rho_1\sigma_1^*\rho_2\sigma_2^*\ldots\rho_m\sigma_m^*$ where the ρ_i, σ_i are sequences of channel actions, can be decided in time $n^{O(m)}$, where n is the size of the instance. While shortest reachability witnesses can be exponentially long when the number m of cycles is not bounded, techniques based on SLP-compressed words allow handling and checking these witnesses in polynomial time, leading to an NP algorithm for flat LCMs. This easily translates into NP algorithms for nontermination, repeated reachability, and unboundedness, and in fact all four problems are NP-complete. Thus the restriction to flat systems really brings some simplification when compared to the very high complexity – sometimes undecidability as is the case for unboundedness – of verification for unrestricted LCMs [13, 40, 39].

▶ Remark 1.1 (Lossy channel machines vs. lossy channel systems). In line with most works on loop acceleration and verification of flat systems, we consider lossy channel “machines” instead of the more usual lossy channel “systems”, i.e., systems where several independent concurrent machines communicate via shared channels. This is because a combination of individually flat machines does not lead to a “flat” system. Additionally, finite-state concurrent systems typically have PSPACE-hard verification problems already when they have no channels and run synchronously, or when they only synchronise via bounded channels that can hold at most one message [19]. ⌟

Outline. After some technical preliminaries (Section 2), we present our main technical contribution (Section 3): we analyse the computation of predecessors (of some given configuration) through a cycle iterated arbitrarily many times. In particular we show that the backward-reachability analysis of a single cycle reaches its fixpoint after a bilinear number of iterations. This leads to an effective bound on the length of the shortest runs between two configurations. In Section 4 we show how the previous analysis can be turned into a nondeterministic polynomial-time algorithmic via the use of SLP-compressed words for efficiently computing intermediary channel contents along a run. In Section 5 we show how our main results also apply to termination, repeated reachability, and boundedness. Finally Appendix C presents reductions showing how the problems we considered are NP-hard, even for acyclic LCMs or single-path LCMs.

Related work. After we circulated our draft proof, we became aware that a related NP-membership result will be found in [23]. There the authors adapt the powerful technique from [20] and encode front-lossy channel systems into multi-head pushdown automata, from which an NP-algorithm for control-state reachability in flat machines ensue. Our approach is lower level, providing a tight bilinear bound on the number of times a cycle must be visited in the backward-reachability algorithm. Once these bounds are established, our NP algorithm only needs to guess the number of times each cycle is visited and perform the polynomial-time verification described in Section 4.

2 Preliminaries

We consider words $u, v, w, x, y, z, \dots$ over a finite alphabet $\Sigma = \{a, b, \dots\}$. We write $|u|$ for the length of a word and ε for the empty word. The set of letters that occur in u is written $\mathrm{alph}(u)$. For a n-letter word $u = a_1 \cdots a_n$ and some index $\ell \in \{0, 1, \dots, n\}$, we write $u_{\leq \ell} \stackrel{\text{def}}{=} a_1 \cdots a_\ell$ and $u_{>\ell} \stackrel{\text{def}}{=} a_{\ell+1} \cdots a_n$ for the ℓ-th prefix and the ℓ-th suffix of u. We write $u_{(\ell)} \stackrel{\text{def}}{=} u_{>\ell} \cdot u_{\leq \ell}$ for the ℓ-th *cyclic shift* of u.

Exponents are used to denote the concatenation of multiple copies of a same word, i.e., u^3 denotes $u\,u\,u$. A *fractional exponent* $p \in \mathbb{Q}$ can be used for u^p if $p \cdot |u|$ is a natural number. E.g., when a, b, c are letters, $(abc)^{\frac{11}{3}}$, or equivalently $(abc)^{3+\frac{2}{3}}$, denotes $abc\,abc\,abc\,ab$.

We write $u \preccurlyeq v$ to denote that u is a (scattered) subword of v, i.e., there exist $2m+1$ words $u_1, \dots, u_m, x_0, x_1, \dots, x_m$ such that $u = u_1 \cdots u_m$ and $v = x_0 u_1 x_1 \dots u_m x_m$. It is well-known that $\preccurlyeq$ is a well-founded partial ordering. For a word $x \in \Sigma^*$, we write $\uparrow x \stackrel{\text{def}}{=} \{y \in \Sigma^* \mid x \preccurlyeq y\}$ to denote the *upward-closure* of x, i.e., the set of all words that contain x as a (scattered) subword.

LCMs. In this paper we consider channel machines with a single communication channel[2]. A *lossy channel machine* (LCM) is a tuple $S = \langle Q, \Sigma, \Delta \rangle$ where $Q = \{p, q, \dots\}$ is a finite set of *control locations*, or just "locations", $\Sigma = \{a, b, \dots\}$ is the finite *message alphabet*, and $\Delta \subseteq Q \times (\{!, ?\} \times \Sigma^*) \times Q$ is a finite set of *transition rules*. A rule $\delta = \langle q, (d, w), q' \rangle$ has a start location q, an end location q' and a *channel action* (d, w). We write $Act_\Sigma \stackrel{\text{def}}{=} \{!, ?\} \times \Sigma^*$ for the set of channel actions over Σ, and often omit the Σ subscript when it can be inferred from the context. We use $\theta, \theta', \dots$ to denote actions and $\sigma, \rho, \dots$ to denote sequences of channel actions.

We will constantly refer to the *written part* and the *read part* of some channel action (or sequence of such). These are formally defined via

$$\begin{aligned} \mathtt{wri}(!w) &\stackrel{\text{def}}{=} w\,, & \mathtt{wri}(?w) &\stackrel{\text{def}}{=} \varepsilon\,, & \mathtt{wri}(\theta_1 \cdots \theta_m) &\stackrel{\text{def}}{=} \mathtt{wri}(\theta_1) \cdots \mathtt{wri}(\theta_m)\,, \\ \mathtt{rea}(!w) &\stackrel{\text{def}}{=} \varepsilon\,, & \mathtt{rea}(?w) &\stackrel{\text{def}}{=} w\,, & \mathtt{rea}(\theta_1 \cdots \theta_m) &\stackrel{\text{def}}{=} \mathtt{rea}(\theta_1) \cdots \mathtt{rea}(\theta_m)\,. \end{aligned} \tag{1}$$

Semantics. The operational semantics of LCMs is given via transition systems. Fix some LCM $S = \langle Q, \Sigma, \Delta \rangle$. Actions in Act_Σ induce a ternary relation $\rightarrow\ \subseteq \Sigma^* \times Act_\Sigma \times \Sigma^*$ on channel contents:

$$x \xrightarrow{!\,w} y \ \stackrel{\text{def}}{\Longleftrightarrow}\ y \preccurlyeq x\,w\,, \qquad\qquad x \xrightarrow{?\,w} y \ \stackrel{\text{def}}{\Longleftrightarrow}\ w\,y \preccurlyeq x\,. \tag{2}$$

Observe how Equation (2) includes the subword relation in the definition of the operational semantics. This models the fact that messages in the channel can be lost nondeterministically during any single computation step. A consequence is the following monotonicity property: if $x' \succcurlyeq x$ and $y \succcurlyeq y'$ then $x \xrightarrow{\theta} y$ implies $x' \xrightarrow{\theta} y'$.

[2] See Appendix D for a generalisation of our results to multi-channel machines.

A *configuration* of S is a pair $c = (q, x) \in Q \times \Sigma^*$ that denotes a current situation where the control of S is set at q while the contents of the channel is x. We let $\mathit{Conf}_S \overset{\text{def}}{=} Q \times \Sigma^*$ denote the set of configurations. The set of rules Δ induces a labelled transition relation $\rightarrow \subseteq \mathit{Conf}_S \times \Delta \times \mathit{Conf}_s$ between configurations defined by

$$(q, x) \xrightarrow{\delta} (q', y) \overset{\text{def}}{\iff} \delta \in \Delta \text{ has the form } \langle q, \theta, q' \rangle \text{ and } x \xrightarrow{\theta} y\,. \tag{3}$$

Several convenient notations are derived from the main transition relation: we write $c \xrightarrow{\theta} c'$ when $c \xrightarrow{\delta} c'$ for a rule δ that carries action θ. When $\sigma = \theta_1\theta_2 \cdots \theta_m$ is a sequence of actions, we write $c \xrightarrow{\sigma} c'$ when there is a sequence of steps $c_0 \xrightarrow{\theta_1} c_1 \xrightarrow{\theta_2} c_2 \cdots \xrightarrow{\theta_m} c_m$ with $c_0 = c$ and $c_m = c'$. Then $c \xrightarrow{*} c'$ means that $c \xrightarrow{\sigma} c'$ for some sequence σ. Similar notations, e.g., "$x \xrightarrow{\theta_1\,\theta_2} y$" or "$x \xrightarrow{\theta^*} y$", are used for channel contents. In fact, since we shall mostly consider fixed paths, or paths of a fixed shape, we will usually concentrate on the channel contents and leave the visited locations implicit.

Flat LCMs. An elementary cycle of length m in a LCM S is a non-empty set $C = \{\langle p_i, \theta_i, q_i \rangle \mid i = 1, \ldots, m\}$ of rules from Δ such that $q_m = p_1$ and $p_i = q_{i-1}$ when $2 \leq i \leq m$, and such that the p_i's are all distinct. A cycle of length 1 is a *self-loop*. The set $\{p_1, \ldots, p_m\}$ is the set of locations *visited* by C. Note that two distinct cycles may have the same visited set if they use different transition rules.

We say that S is *flat* if no control location is visited by two different elementary cycles. An extreme case of flat machines are the machines having no cycles whatsoever, called *acyclic* machines.[3]

When S is flat, there is (at most) one cycle around any location q and we write σ_q for the sequence of actions along this cycle, making sure that σ_q starts with the action leaving q (so that if q, q' are two locations visited by the same cycle, $\sigma_{q'}$ will be a cyclic shift of σ_q). When there is no cycle visiting q we let $\sigma_q = \varepsilon$ by convention.

Vector Addition Systems with States, aka VASSes. When $|\Sigma| = 1$, the information stored in a channel is completely captured by the length of its contents: we may then speak of "counters" rather than "channels" and end up with a model essentially equivalent to the VASSes from [30]. Of course, VASSes, be they reliable or lossy, are really interesting when several counters are considered.

NP-hardness. It is known that reachability and other verification problems are NP-hard for (reliable) FIFO automata: see [20, App. C] and [22]. We strengthen these results in Appendix C with the following theorems that cover reliable and unreliable channels indifferently (recall that *nontermination* is the existence of an infinite run, while *unboundedness* is the existence of arbitrarily large reachable configurations).

▶ **Theorem 2.1** (Hardness for acyclic channel machines). *Reachability, nontermination and unboundedness are* NP*-hard for* acyclic *channel machines, with reliable or with unreliable channels. Hardness already holds for a single channel and a binary alphabet. It also holds for a unary alphabet (i.e., for acyclic VASSes, reliable or lossy) provided one allows several channels (or counters).*

[3] In the finite-automata literature, "acyclic automata" sometimes allow self-loops.

NP-hardness for acyclic machines uses the nondeterminism allowed in channel machines. It is thus interesting to consider *single-path* machines where the control graph is a single line possibly carrying cycles on some locations, as is done in [33] or [17].[4] In such a machine, nondeterminism only occurs in choosing how many times a cycle is visited (and what messages are lost in unreliable systems).

▶ **Theorem 2.2** (Hardness for single-path channel machines). *Reachability, nontermination and unboundedness are* NP*-hard for* single-path *channel machines, with reliable or with unreliable channels. Hardness already holds for a single channel. It also holds for single-path VASSes, reliable or lossy, provided one allows several counters.*

The above NP-hardness does not apply to bounded path schemes *with a fixed number of cycles* and indeed we show in Section 3 that reachability along path schemes with m cycles can be verified in polynomial-time $n^{O(m)}$.

3 Backward reachability in flat LCMs

In this section we consider a generic flat single-channel LCM S with channel alphabet Σ and investigate the complexity of backward-reachability analysis.

3.1 Computing predecessors

The classical approach to deciding reachability in LCMs is the backward-reachability algorithm proposed by Abdulla and Jonsson. They first developed it for lossy channel systems [4] before generalising it to the larger class of Well-Structured Systems [1, 24].

For backward reachability, we write $\texttt{Pre}[\sigma](x)$ for $\{y \in \Sigma^* \mid y \xrightarrow{\sigma} x\}$, the set of σ-predecessors of x, and $\texttt{Pre}[\sigma](\uparrow x)$ for $\{y \in \Sigma^* \mid \exists x' \in \uparrow x : y \xrightarrow{\sigma} x'\}$, the set of σ-predecessors of x "and larger contents". A consequence of the monotonicity of steps is that $\texttt{Pre}[\sigma](\uparrow x)$ is upward-closed set and, unless σ is the empty sequence, coincides with $\texttt{Pre}[\sigma](x)$.

▶ **Definition 3.1** ($\texttt{pr}[\sigma](x)$). *For a channel content $x \in \Sigma^*$ and a sequence σ of channel actions, we write $\texttt{pr}[\sigma](x) = y$ when $\texttt{Pre}[\sigma](\uparrow x) = \uparrow y$.*

In the case of lossy channels, $\texttt{Pre}[\sigma](\uparrow x)$ always has a single minimal element, hence $\texttt{pr}[\sigma](x)$ is always defined. We now explain how to compute it.

For two words x, v we define x/v as the prefix of x that remains when we remove from x its longest suffix that is a subword of v. This operation is always defined and can be computed using the following rules where a, b are letters:

$$x/\varepsilon = x\,, \qquad \varepsilon/v = \varepsilon\,, \qquad (x\,a)/(v\,b) = \begin{cases} x/v & \text{if } a = b,\\ (x\,a)/v & \text{if } a \neq b. \end{cases} \tag{4}$$

This immediately entails $(x/v)/v' = x/(v'\,v)$. We'll also use the following properties:

$$\text{if } x'/v \neq \varepsilon \text{ then } x(x'/v) = (x\,x')/v\,, \qquad \text{if } |x'| > |v| \text{ then } x'/v \neq \varepsilon\,. \tag{5}$$

We may now compute $\texttt{pr}[\sigma](x)$ with:

$$\begin{aligned} \texttt{pr}[?u](x) &= u \cdot x\,, & \texttt{pr}[!v](x) &= x/v\,,\\ \texttt{pr}[\varepsilon](x) &= x\,, & \texttt{pr}[\sigma_1 \cdot \sigma_2](x) &= \texttt{pr}[\sigma_1]\big(\texttt{pr}[\sigma_2](x)\big)\,. \end{aligned} \tag{6}$$

[4] Equivalently, one can analyze general machines "modulo bounded expressions" as is done in [20].

W.r.t. subword ordering, the / operation is monotonic in its first argument and contra-monotonic in the second : $u \preccurlyeq u'$ implies $u/v \preccurlyeq u'/v$ and $x/u \succcurlyeq x/u'$. Concatenation too is monotonic. This generalises to the following useful lemma:

▶ **Lemma 3.2.** *Assume* $\mathtt{pr}[\sigma](x) = y$ *and* $\mathtt{pr}[\sigma](x') = y'$ *where* σ *is some sequence of actions. Then* $x \preccurlyeq x'$ *implies* $y \preccurlyeq y'$*.*

Proof. By induction on σ, using Equations (4) and (6). ◀

3.2 Cycles: repeating a given sequence of actions

We now focus on computing $\mathtt{pr}[\sigma^k](x)$ for σ a sequence of actions and some $k \in \mathbb{N}$.

Without any loss of generality, σ can be written in the general form $?a_1\,!b_1\,?a_2\,!b_2 \cdots ?a_r\,!b_r$ where each a_i and b_i is a letter or the empty word ε. Then $\mathtt{rea}(\sigma) = a_1 a_2 \cdots a_r$ and $\mathtt{wri}(\sigma) = b_1 b_2 \cdots b_r$.

To fix notation, we define "*the small-step sequence for* $\mathtt{pr}[\sigma](x)$", or just "*the SSS*", as the sequence y_r, y'_r, y_{r-1}, y'_{r-1}, …, y_1, y'_1, y_0 of $2r+1$ words given by

$$y_r = x\,, \qquad y'_i = y_i/b_i\,, \qquad y_{i-1} = a_i\, y'_i\,. \tag{7}$$

Clearly, the SSS lists all the intermediary steps in the computation of $\mathtt{pr}[\sigma](x)$ as dictated by Equation (6), and thus it yields $y_0 = \mathtt{pr}[\sigma](x)$.

Our first lemma handles the special case where x is made of copies of $\mathtt{rea}(\sigma)$.

▶ **Lemma 3.3.** *Let* $u = \mathtt{rea}(\sigma)$*.*

(i) *If* x *is a fractional power* u^p *of* u*, then* $y = \mathtt{pr}[\sigma](x)$ *is also a fractional power of* u*, written* $y = u^m$*.*

(ii) *Furthermore, if* $m > 1$*, then* $\mathtt{pr}[\sigma](u^{p+n}) = u^{m+n}$ *for all* $n \in \mathbb{N}$*.*

(iii) *Finally, for all* $n \in \mathbb{N}$*, if* $m > n+1$*, then* $\mathtt{pr}[\sigma](u^{p-n}) = u^{m-n}$*.*

Proof. The lemma holds spuriously if $u = \varepsilon$, so we assume $|u| > 0$. Let us write σ in the general form $?a_1\,!b_1\,?a_2\,!b_2 \cdots ?a_r\,!b_r$, so that $u = a_1 a_2 \cdots a_r$. To simplify notation we will write $u_{(i)}$ for the shift $a_{i+1} \cdots a_r \cdot a_1 \ldots a_i$ that really should be written $u_{(|a_1 \cdots a_i|)}$ (remember that $a_j = \varepsilon$ is possible).

We now claim that, in the SSS $(y_i, y'_i)_i$ for σ and x, each y_i and y'_i is a fractional power of $u_{(i)}$, written $y_i = u_{(i)}^{p_i}$ and $y'_i = u_{(i)}^{p'_i}$.

The proof is by induction on $r - i$. For y_i, there are two cases: (1) $y_r = x$ is a power of u by assumption, hence of $u_{(r)}$, with $p_r = p$; (2) y_{i-1} is $a_i\, y'_i$, i.e., $a_i\, u_{(i)}^{p'_i}$ by ind. hyp., hence a power of $u_{(i-1)}$ with $p_{i-1} = p'_i + \frac{|a_i|}{|u|}$. For y'_i the proof is simpler: by ind. hyp. it is $u_{(i)}^{p_i}/b_i$ and, as a prefix of a power of $u_{(i)}$, is itself a power of $u_{(i)}$, albeit with a perhaps smaller exponent, i.e., $p_i - \frac{|b_i|}{|u|} \leq p'_i \leq p_i$.

(i) Since y coincide with y_0, we obtain $y = u^m$ as required by letting $m = p_0$.

(ii) Equation (8) gathers the (in)equalities we just established:

$$p_r = p\,, \qquad p_{i-1} = p'_i + \frac{|a_i|}{|u|}\,, \qquad \max\Bigl(0, p_i - \frac{|b_i|}{|u|}\Bigr) \leq p'_i \leq p_i\,, \qquad p_0 = m\,. \tag{8}$$

Thus the assumption $m > 1$ entails $p'_i > 0$, i.e. $y'_i \neq \varepsilon$, for all $i = r, r-1, \ldots, 2, 1$. Let us now consider the SSS $(z_i, z'_i)_i$ for $\mathtt{pr}[\sigma](u^{p+1})$. We claim that for all i, $z_i = u_{(i)} y_i$ and $z'_i = u_{(i)} y'_i$, as is easily proven by induction on $r - i$. The crucial case is z'_i, defined as z_i/b_i and equal to $(u_{(i)} y_i)/b_i$ by ind. hyp. Since $y_i/b_i = y'_i \neq \varepsilon$ as just observed, we deduce $(u_{(i)}y_i)/b_i = u_{(i)}(y_i/b_i)$ from Equation (5). This is $u_{(i)}y'_i$ as required. Finally we end up with $\mathtt{pr}[\sigma](u^{p+1}) = z_0 = u_{(0)}y_0 = u\,u^m = u^{m+1}$, and this generalises to $\mathtt{pr}[\sigma](u^{p+n}) = u^{m+n}$.

(iii) With Equation (8), the assumption $m > n+1$ now entails $p_i, p'_i \geq n + \frac{1}{|u|}$ for all i. We claim that the SSS $(z_i, z'_i)_i$ for $\mathtt{pr}[\sigma](u^{p-n})$ satisfies $u^n_{(i)}z_i = y_i$ and $u^n_{(i)}z'_i = y'_i$ for all i, as can be proved by induction on $r-i$. The base case $u^n z_r = u^n u^{p-n} = u^p = y_r$ is clear. Let us now consider $u^n_{(i)}z'_i$. It is $u^n_{(i)}(z_i/b_i)$, that is $u^n_{(i)}(u_{(i)}^{p_i-n}/b_i)$ since $u^n_{(i)}z_i = y_i$ by ind. hyp. and $y_i = u^{p_i}_{(i)}$ by (i). Now $|u_{(i)}^{p_i-n}| = |u|(p_i - n) \geq 1 \geq |b_i|$, so Equation (5) applies and we deduce $u^n_{(i)}(z_i/b_i) = (u^n_{(i)}z_i)/b_i = y_i/b_i$ (by ind. hyp.) $= y'_i$. We have proved $u^n_{(i)}z'_i = y'_i$ as required. Finally, proving $u^n_{(i)}z_i = y_i$ is handled in a similar way. ◀

Note that $m > 1$ is required for part (ii) of the Lemma. For example, with $\sigma = ?a!b?c!c!a$ one has $u = \mathtt{rea}(\sigma) = ac$ and $\mathtt{pr}[\sigma](u^{\frac{1}{2}}) = u^1$. However one can check that $\mathtt{pr}[\sigma](u^{\frac{3}{2}}) = aca = u^{\frac{3}{2}}$.

Equipped with Lemma 3.3, we turn to the general case for $\mathtt{pr}[\sigma^k](x)$.

▶ **Theorem 3.4.** *Let $\sigma \in Act^*_\Sigma$ be a sequence of actions and write u for $\mathtt{rea}(\sigma)$. Let $x \in \Sigma^*$ be some channel content and write y_k for $\mathtt{pr}[\sigma^k](x)$.*

(i) *For every $k \in \mathbb{N}$, y_k has the form $u^{p_k} \cdot x_{<\ell_k}$ for some fractional power p_k and some length $\ell_k \in \{0, 1, \ldots, |x|\}$.*

(ii) *Furthermore, computing p_k and ℓ_k can be done in time $\mathsf{poly}(|\sigma| + |x| + \log k)$.*

Proof.

(i) Write v for $\mathtt{wri}(\sigma)$ and consider the sequence $(x_k)_{k\in\mathbb{N}}$ given by $x_0 \overset{\text{def}}{=} x$ and $x_{k+1} \overset{\text{def}}{=} x_k/v$. Note that $|x_{k+1}| \leq |x_k|$ for all k and write $\boldsymbol{\kappa}$ for the largest index with $x_{\boldsymbol{\kappa}} \neq \varepsilon$. We let $\boldsymbol{\kappa} = -1$ if already we started with $x = \varepsilon$, and $\boldsymbol{\kappa} = \omega$ if all x_k's are non-empty, which happens iff $\mathrm{alph}(x) \not\subseteq \mathrm{alph}(v)$.

If $k \leq \boldsymbol{\kappa}$, $\mathtt{pr}[\sigma^k](x) = u^k \cdot x_k$ and x_k is a prefix of x, so taking $p_k = k$ and $\ell_k = |x_k|$ works.

If $k = \boldsymbol{\kappa}+1$, y_k is $\mathtt{pr}[\sigma](u^{\boldsymbol{\kappa}} \cdot x_{<\ell_{\boldsymbol{\kappa}}})$. Since $x_{<\ell_{\boldsymbol{\kappa}}}/v = \varepsilon$, the result is a prefix of $u^{\boldsymbol{\kappa}+1}$, so has the form $u^{p_{\boldsymbol{\kappa}+1}}$ for some $p_{\boldsymbol{\kappa}+1}$. One also lets $l_{\boldsymbol{\kappa}+1} = 0$.

Finally, if $k > \boldsymbol{\kappa}+1$, we have $y_k = \mathtt{pr}[\sigma^{k-\boldsymbol{\kappa}-1}](y_{\boldsymbol{\kappa}+1}) = \mathtt{pr}[\sigma^{k-\boldsymbol{\kappa}-1}](u^{p_{\boldsymbol{\kappa}+1}})$ and we just have to invoke Lemma 3.3 (and set $l_k = 0$).

(ii) Computing $\boldsymbol{\kappa}$ takes time $O(|x| + |\sigma|)$.

If $k \leq \boldsymbol{\kappa}$, comparing k with $\boldsymbol{\kappa}$ and computing p_k and ℓ_k takes additional time $O(|x| + |\sigma| + \log k)$.

If $k = \boldsymbol{\kappa}+1$, we need to compute $\mathtt{pr}[\sigma](u^{\boldsymbol{\kappa}}\, x_{<\boldsymbol{\kappa}})$ in order to extract $p_{\boldsymbol{\kappa}+1}$. This uses Equation (6) for $O(|\sigma|)$ small steps. Note that we do not build $u^{\boldsymbol{\kappa}}$ explicitly: once x has been consumed, we work on some $u^p_{(i)}$ and just update p and i when applying some $\mathtt{pr}[?a]$, or only update p when applying some $\mathtt{pr}[!b]$, for which we only need to know where are the occurrences of b in u. For each small step, the updates can be computed in time $O(|u| + |x|)$, hence $p_{\boldsymbol{\kappa}+1}$ is computable in quadratic time.

If $k > \boldsymbol{\kappa}+1$, we set $q_0 = p_{\boldsymbol{\kappa}+1}$, $k' = k - \boldsymbol{\kappa} - 1$ and aim for $\mathtt{pr}\big[\sigma^{k'}\big](x')$, starting from $x' = u^{q_0}$. We need to compute $u^{q_{k'}}$ in the sequence $u^{q_0}, u^{q_1}, \ldots$, defined by $u^{q_{i+1}} \overset{\text{def}}{=} \mathtt{pr}[\sigma](u^{q_i})$. Let us first compute q_1 and consider the three possibilities:

(1) If $q_0 = q_1$, y_p is a fixpoint for $\mathtt{pr}[\sigma]$ and we know $p_k = p$.

(2) If $q_0 < q_1$, the exponents increase under $\mathtt{pr}[\sigma]$ and after computing at most $|u|+1$ consecutive values, we'll find two indexes $1 \leq i < j \leq |u|+1$ such that q_i and q_j

have the same fractional parts, i.e., differ by some natural number. We can then use Lemma 3.3.(ii) and compute $q_{j+\left\lfloor\frac{k'-j}{j-i}\right\rfloor} = q_j + \left\lfloor\frac{k'-j}{j-i}\right\rfloor$. From there, we're just at most $|u|$ steps from $q_{k'}$, i.e., p_k.

(3) Finally, if $q_0 > q_1$ a similar technique, now relying on Lemma 3.3.(iii), will let us compute $q_{k'}$ in polynomial time. ◀

The next step is to compute $\mathtt{Pre}[\sigma^*](\uparrow x)$, that is, $\uparrow x \cup \mathtt{Pre}[\sigma](\uparrow x) \cup \mathtt{Pre}[\sigma^2](\uparrow x) \cup \cdots$. Like $\mathtt{Pre}[\sigma](\uparrow x)$, this set is upward-closed. However it may have several minimal elements (see Example 3.8) and one needs to collect all of them in order to represent the set faithfully.

▶ **Definition 3.5** (Iteration number). *The* iteration number $L(\sigma, x)$ *associated with a sequence of actions σ and a channel content x is the smallest integer such that there exists $\ell \leq L(\sigma, x)$ with $\mathtt{pr}[\sigma^\ell](x) \preccurlyeq \mathtt{pr}[\sigma^{L(\sigma,x)+1}](x)$. Note that, by Higman's Lemma, such an integer always exists.*

The point of Definition 3.5 is that it captures the number of iterations that are sufficient to compute $\mathtt{Pre}[\sigma^*](\uparrow x)$.

▶ **Lemma 3.6.** $\mathtt{Pre}[\sigma^*](\uparrow x) = \bigcup_{i=0}^{L(\sigma,x)} \uparrow \mathtt{pr}[\sigma^i](x)$.

Proof. Write y_k for $\mathtt{pr}[\sigma^k](x)$ and L for $L(\sigma, x)$. By definition there is some $\ell \leq L$ with $y_\ell \preccurlyeq y_{L+1}$. By Lemma 3.2, this continues into $y_{\ell+1} \preccurlyeq y_{L+2}$, $y_{\ell+2} \preccurlyeq y_{L+3}$, etc., implying $\uparrow y_\ell \supseteq \uparrow y_{L+1}$, $\uparrow y_{\ell+1} \supseteq \uparrow y_{L+2}$, $\uparrow y_{\ell+2} \supseteq \uparrow y_{L+3}$, … Finally $\mathtt{Pre}[\sigma^*](\uparrow x)$, which is $\bigcup_{i\in\mathbb{N}} \uparrow y_i$ coincides with the finite union $\bigcup_{i=0}^{L(\sigma,x)} \uparrow y_i$. ◀

▶ **Theorem 3.7** (Bounding iteration numbers). $L(\sigma, x) \leq |x|(|\mathtt{rea}(\sigma)| + 1)$ *for any action sequence σ and channel contents x.*

Proof. We write u for $\mathtt{rea}(\sigma)$. Using Theorem 3.4, we write $y_k = \mathtt{pr}[\sigma^k](x) = u^{p_k} \cdot x_{<\ell_k}$ and observe that $p_i \leq p_j$ and $\ell_i \leq \ell_j$ imply $y_i \preccurlyeq y_j$. Recall from the proof of Theorem 3.4 that $|x| = \ell_0 \geq \ell_1 \geq \cdots \geq \ell_i \geq \cdots$ is a decreasing sequence and that $p_k = k$ when $\ell_k > 0$.

There are two cases:

1. If $(\ell_k)_k$ stabilises with some limit value ℓ_∞ that is strictly positive, then $\ell_{|x|-1} = \ell_{|x|}$ and we deduce $y_{|x|-1} \preccurlyeq y_{|x|}$, entailing $L(\sigma, x) < |x|$.
2. If $\ell_\infty = 0$ then, writing k_0 for the first index with $\ell_{k_0} = 0$, we know that $k_0 \leq |x|$ and $p_{k_0} = k_0$. If $p_{k_0+1} \geq p_{k_0}$ then $y_{k_0} \preccurlyeq y_{k_0+1}$. Otherwise $p_{k_0} > p_{k_0+1}$ and as a consequence of Lemma 3.2 the suffix sequence $p_{k_0} > p_{k_0+1} \geq p_{k_o+2} \geq p_{k_0+3} \geq \cdots$ is decreasing. Since the p_k fractions are multiples of $\frac{1}{|u|}$, the sequence $(p_k)_{k\geq k_0}$ can only take $1+|u|k_0$ different values and eventually yield $p_k = p_{k+1}$ for some $k \leq k_0 + k_0|u| \leq |x|(|u| + 1)$, entailing $L(\sigma, x) \leq |x|(|u| + 1)$ as claimed. ◀

The bound given by Theorem 3.7 is tight as the next simple example shows.

▶ **Example 3.8** (Bounds for $L(\sigma, x)$ are tight.). For $\sigma = !ab^5\ ?b^4$ and $x = a^4$, the $(y_k)_k$ sequence with $y_k \overset{\text{def}}{=} \mathtt{pr}[\sigma^k](x)$ is:

$$\begin{array}{ccccccccccccc}
\multicolumn{4}{c}{|x|=l_0>l_1>\cdots>l_4=0} & & \multicolumn{8}{c}{k_0=4=p_{k_0}\wedge p_4\geq p_5\geq p_6\cdots}\\
a^4, & b^4a^3, & b^8a^2, & b^{12}a, & b^{16}, & b^{15}, & b^{14}, & b^{13}, & \ldots & b, & \varepsilon, & \varepsilon, & \ldots\\
y_0 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & & y_{19} & y_{20} & y_{21} &
\end{array}$$

Since $y_{20} \preccurlyeq y_{21}$ is the earliest increasing pair, Definition 3.5 gives $L(!ab^5\ ?b^4, a^4) = 20$.

This generalises to $L(!ab^{n+1}\ ?b^n, a^m) = m(n + 1)$ for any $n, m \in \mathbb{N}$, which is exactly the $|x|(|\mathtt{rea}(\sigma)| + 1)$ bound given by Theorem 3.7. ⌟

3.3 Bounding runs

Let $\texttt{Post}^*(c)$ denote the reachability set $\{c' \mid c \xrightarrow{*} c'\}$. Assume that a flat LCM S is such that $(q', y) \in \texttt{Post}^*(q, x)$. Since S is flat, the run $(q, x) \xrightarrow{*} (q', y)$ has the following shape:

$$\begin{aligned}(q, x) = (q_0, z_0) \xrightarrow{\sigma_0^{n_0}} (q_0, z_0') \xrightarrow{\theta_1} (q_1, z_1) \xrightarrow{\sigma_1^{n_1}} (q_1, z_1') \xrightarrow{\theta_2} (q_2, z_2) \xrightarrow{\sigma_2^{n_2}} \cdots \\ \cdots\ (q_{m-1}, z_{m-1}') \xrightarrow{\theta_m} (q_m, z_m) \xrightarrow{\sigma_m^{n_m}} (q_m, z_m') = (q', y)\,.\end{aligned} \tag{9}$$

In Equation (9), the control locations $(q =)q_0, q_1, \ldots, q_m(= q')$ are all distinct, σ_i is the sequence of actions performed along the (unique) cycle on q_i, and n_i is the number of times this cycle has been traversed along the run. We use $\sigma_i = \varepsilon$ when there is no cycle on q_i, and we use $n_i = 0$ when the cycle is not traversed at all. For $i = 1, \ldots, m$, θ_i is the sequence of actions that labels the transition from q_{i-1} to q_i.

We say that the run in Equation (9) is *minimal* if for all $i = 1, \ldots, m$, z_i is a minimal element in $\texttt{Pre}[\sigma_i^*](\uparrow z_i')$ and n_i is the smallest such $z_i = \texttt{pr}[\sigma_i^{n_i}](z_i')$, and if $z_i = \texttt{pr}[\sigma_i^{n_i}](z_i')$ for $i = 0, \ldots, m$. By allowing $z_0 \preccurlyeq x$, it is always possible to associate a minimal run with some reachability statement "$(q, x) \xrightarrow{*} (q', y)$" and use the tuple

$$\langle q_0, z_0, n_0, z_0', q_1, z_1, n_1, z_1', \ldots, q_m, z_m, n_m, y\rangle \tag{10}$$

as a witness of reachability.

We now try to bound the size of such a witness. One has

$$|z_m'| = |y|\,, \qquad |z_i| \leq |z_i'| + n_i|\texttt{rea}(\sigma_i)|\,, \qquad |z_{i-1}'| \leq |z_i| + |\texttt{rea}(\theta_i)|\,, \tag{11}$$

for all i. We further know from Theorem 3.7, that $n_i \leq |z_i'|(1 + |\texttt{rea}(\sigma_i)|)$ for $i = 0, \ldots, m$.

Thus, writing n for the size $|S| + |x| + |y|$ of the instance (so that $m \leq n$, and $|\texttt{rea}(\sigma_i)|, |\texttt{rea}(\theta_i)| \leq n$ for all i), we have quadratic bounds $O(n^2)$ for n_m and $|z_m|$, cubic bounds $O(n^3)$ for n_{m-1} and $|z_{m-1}|$, …, etc., so that the witness has size $O(n^m)$, hence $2^{O(n)}$.

Unfortunately, as Example 3.9 shows, these bounds cannot be much improved upon.

▶ **Example 3.9.** Consider the flat LCM S depicted in Figure 1. In S, $(q_0, \varepsilon) \xrightarrow{*} (q_0', \varepsilon)$ is witnessed by the following run schema

$$\begin{aligned}(q_0, \varepsilon) \to (q_1, ab) \xrightarrow{*} (q_1, ba^2) \to (q_2, a^2b) \xrightarrow{*} (q_2, ba^4) \to (q_3, a^4b) \xrightarrow{*} \cdots \xrightarrow{*} (q_n, ba^{2^n}) \\ \to (q_n', a^{2^n}b) \xrightarrow{*} (q_n', ba^{2^{n-1}}) \to (q_{n-1}', a^{2^{n-1}}b) \xrightarrow{*} \cdots \to (q_1', a^2b) \xrightarrow{*} (q_1', ba) \to (q_0', \varepsilon)\,.\end{aligned}$$

In fact, there is only one run witnessing $(q_0, \varepsilon) \xrightarrow{*} (q_0', \varepsilon)$ and this run necessarily visits (q_n, ba^{2^n}), a configuration of exponential size, iterating 2^{n-1} times the cycle on q_n. Observe that, starting from (q_0, ε), any message loss will prevent ever reaching q_0'. ⌟

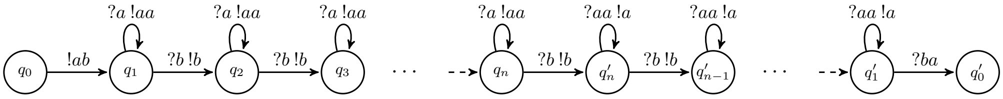


Figure 1 A flat LCM where $(q_0, \varepsilon) \xrightarrow{*} (q_0', \varepsilon)$ requires exponential-sized configurations.

4 SLP-compressed words and an NP algorithm for reachability

In this section we explain how the exponentially long minimal runs analysed in Section 3.3 can be handled efficiently using SLP-compressed words. This provides witnesses of polynomial size that can be validated in polynomial time, thus showing that reachability in flat LCMs is in NP.

4.1 SLP-compressed words

Compressed words are data structures used to represent long words via succinct encodings. If a long word is rather repetitive, it can have a succinct encoding of logarithmic size. Since several operations on long words or decision tests about them can be performed efficiently on the succinct representation, compressed words have been used to provide efficient solutions to algorithmic problems involving exponential-size (but rather repetitive) words, see [37] for a survey.

The most studied encoding is the SLP, for Straight-Line Program, which is in effect an acyclic context-free grammar that generates a single word, called its *expansion*.

From now on, we always use small letters x, y, u, v for usual words, and capital letters X, Y, U, V for SLPs expanding to the corresponding words. Since SLPs are interpreted as plain words, we will use them freely in places where words can be used. It will always be clear when we consider the SLP as a data structure and then we use it to denote its expansion. The main situation where we want to distinguish between the two usages is when reasoning about size and algorithmic complexity: for this we write $|X|$ for the length $|x|$ of the expansion, while we write $\|X\|$ for the size of the SLP as a data structure. For example, if X expands to x then for any fractional power of the form x^p, there is an SLP X^p with $|X^p| = |x^p| = p|x|$ and $\|X^p\| = O(\|X\| + \log p)$.

In the rest of this section we will use well-known, or easy to prove, algorithmic results on SLPs. In particular, all the following problems can be solved in polynomial time:

length: Given a SLP X, compute $|X|$.

factor: Given a SLP X and two positions $0 \leq i \leq j \leq |X|$, construct a SLP of size $O(\|X\|)$ for the factor $X[i:j]$.

concatenation: Given two SLPs X and Y, construct a SLP for $X \cdot Y$.

matching: Given two SLPs X and Y, decide if X is a factor (or a prefix, or a suffix) of Y.

To this list we add results tailored to our needs:

(scattered) subword with a power word: Given a SLP X, a plain word v and some power $k \in \mathbb{N}$, decide if $X \preccurlyeq v^k$. This special case of the fully compressed subsequence test can be done in time $\mathsf{poly}(\|X\| + |v| + \log k)$, see Proposition A.1 in the Appendix.

iterated LCM predecessor: Given a SLP X, a plain word v, and some power $k \in \mathbb{N}$, compute a SLP for X/v^k, i.e., for $\mathtt{pr}[(!v)^k](X)$. This can be done in time $\mathsf{poly}(\|X\| + |v| + \log k)$, see Proposition A.2 in the Appendix.

With the above results, we are ready to lift the computation of $\mathtt{pr}[\sigma^k](x)$ from plain words to SLPs:

▶ **Proposition 4.1.** *Given an SLP X, a sequence of actions σ, and some $k \in \mathbb{N}$, it possible to compute an SLP Y for $\mathtt{pr}[\sigma^k](X)$ in time $\mathsf{poly}(\|X\| + |\sigma| + \log k)$.*

Proof (sketch). We follow the construction described in the proof of Theorem 3.4, now using SLPs. So again let us write u and v for $\mathtt{rea}(\sigma)$ and $\mathtt{wri}(\sigma)$.

The first step is to compute $\boldsymbol{\kappa}$. This is done by dichotomic search, since we can decide in polynomial time whether a candidate n leads to $X/v^n = \varepsilon$. We then build $X_{<\ell_{\boldsymbol{\kappa}}}$ as $X/v^{\boldsymbol{\kappa}}$.

If $k \leq \boldsymbol{\kappa}$, we build a SLP Y for $u^k \cdot (X/v^k)$ and we are done.

If $k \geq \boldsymbol{\kappa} + 1$, we compute a SLP for $y_{\boldsymbol{\kappa}+1} = u^{p_{\boldsymbol{\kappa}+1}}$ by applying $\mathtt{pr}[\sigma]$ on a SLP for $y_{\boldsymbol{\kappa}} = u^{\boldsymbol{\kappa}} \cdot x_{<\ell_{\boldsymbol{\kappa}}}$: this involves computing a SSS involving at most $2m$ operations like prefixing by a_i or computing Y/b_j. This is done in polynomial time and the exponent in $u^{p_{\boldsymbol{\kappa}+1}}$ can be computed by dividing the length of a SLP with the length of u. From there we continue as in the proof of Theorem 3.4. This involves performing a polynomial number of simple $\mathtt{pr}$ operations and some simple reasoning on the exponents. ◀

4.2 Reachability for flat LCMs is in NP

We now explain how Equation (10) can be replaced by an SLP-based witness of the form

$$\langle q_0, Z_0, n_0, Z'_0, q_1, Z_1, n_1, Z'_1, \ldots, q_m, Z_m, n_m, Y\rangle\ . \tag{10'}$$

▶ **Lemma 4.2.** *If $\langle q_0, z_0, n_0, z'_0, q_1, n_1, z_1, z'_1, \ldots, q_m, z_m, n_m, y\rangle$ is a minimal witness for $(q, x) \xrightarrow{*} (q', y)$ in S, then there exist SLPs $Z_0, Z'_0, Z_1, \ldots, Z_m, Y$ representing $z_0, z'_0, z_1, \ldots, z_m, y$ that have size polynomial in $|S| + |y|$.*

Proof. By induction on $m - i$. We start with Y for y which does not need any compression (and let $Z'_m = Y$ for the inductive reasoning).

Then any Z_i has the shape $U_i^{p_i} \cdot (Z'_i)_{<\ell_i}$ for some p_i and ℓ_i. Now $\|(Z'_i)_{<\ell_i}\|$ is in $O(\|Z'_i\|)$ and since p_i is in $2^{O(|S|)}$ – as shown in Section 3.3 – , the size of the SLP for $u_i^{p_i}$ is is $O(|u_i| + |S|)$, i.e., $O(|S|)$.

Now any Z'_{i-1} is $\mathtt{pr}[\theta_i](Z_i)$ and is easily obtained from Z_i and θ_i according to Equation (6). One can ensure that $\|Z'_i\|$ is in $O(\|Z_i\| + |S|)$.

Finally, and since each SLP has size linearly bounded in the size of the following one (the bounds propagate from right to left), we have a quadratic bound on the individual sizes for the Z_i and Z'_i, hence a cubic bound on the SLP witness overall (recall that the n_i, written in binary, have size $O(|S|)$). ◀

▶ **Theorem 4.3.** *Deciding whether $(q, x) \xrightarrow{*} (q', y)$ in a flat LCM S is* NP*-complete.*

Proof. NP-hardness is proven in Appendix C and we just provide a NP decision algorithm.

As expected, the algorithm just guesses a SLP-based witness and checks that it is indeed a valid witness. For a positive instance of the problem, a witness exists and has polynomial size as shown in Lemma 4.2. Now checking that it is valid, i.e., that each Z_i is indeed $\mathtt{pr}[\sigma^{n_i}](Z'_i)$ etc., can be done in polynomial time as shown with Proposition 4.1.[5] ◀

5 NP algorithms for liveness properties

We show in this section how, for flat LCMs, liveness properties like nontermination, unboundedness, and existence of a Büchi run (i.e., a run visiting a given location infinitely many times), effectively reduce to reachability. This only requires characterising and computing the set of configurations from which infinite runs are possible but Section 3 provides all the necessary tools.

With any sequence of channel actions σ we associate $I_\sigma \stackrel{\text{def}}{=} \bigcap_{k=0,1,2,\ldots} \mathtt{Pre}[\sigma^k](\Sigma^*)$.

[5] In fact, it is sufficient to guess the exponents $n_1, \ldots, n_m$ for the σ_i's since the Z_i, Z'_i's can be computed from them.

▶ **Lemma 5.1.** *$I_\sigma \subseteq \Sigma^*$ is an upward-closed set of channel contents. It has a single minimal element or is empty.*

Proof. Write $(y_k)_{k\in\mathbb{N}}$ for the sequence $y_0 \overset{\text{def}}{=} \varepsilon$ and $y_{k+1} = \texttt{pr}[\sigma](y_k)$. Then $\texttt{Pre}[\sigma^k](\Sigma^*) = \uparrow y_k$ for all $k \in \mathbb{N}$ (Definition 3.1) and $I_\sigma = \bigcap_{k\in\mathbb{N}} \texttt{Pre}[\sigma^k](\Sigma^*) = \bigcap_k \uparrow y_k$. From $y_0 \preccurlyeq y_1$ and monotonicity of $\texttt{pr}$ (Lemma 3.2) we obtain $y_0 \preccurlyeq y_1 \preccurlyeq y_2 \preccurlyeq \cdots$ and $\uparrow y_0 \supseteq \uparrow y_1 \supseteq \uparrow y_2 \supseteq \cdots$. Thus we have

$$I_\sigma = \bigcap_{k\in\mathbb{N}} \uparrow y_k = \begin{cases} \uparrow y_K & \text{if } y_K = y_{K+1} \text{ for some } K, \\ \varnothing & \text{if the } (y_k)_{k\in\mathbb{N}} \text{ sequence is strictly increasing.} \end{cases}$$ ◀

We write $\texttt{pr}[\sigma^\omega](\varepsilon) = y$ if $I_\sigma = \uparrow y$, and $\texttt{pr}[\sigma^\omega](\varepsilon) = \bot$ if I_σ is empty.

▶ **Lemma 5.2.** *$\texttt{pr}[\sigma^\omega](\varepsilon)$ can be computed in time $O(|\sigma|^3)$.*

Proof (sketch). We start computing the elements $y_0, y_1, y_2, \ldots$ of the $(y_k)_k$ sequence. If two consecutive values y_K and y_{K+1} coincide, we have found $\texttt{pr}[\sigma^\omega](\varepsilon)$. Otherwise we continue while the sequence is strictly increasing until eventually $|y_k| > |\texttt{rea}(\sigma)|$ for some k (indeed, some $k \leq 1 + |\sigma|$). In this case we can invoke Lemma 3.3.(ii) and conclude that the $(y_k)_k$ sequence will remain strictly increasing, hence $\texttt{pr}[\sigma^\omega](\varepsilon) = \bot$.

For complexity, we note that each y_{k+1} is obtained in time $O(|\sigma| + |y_k|)$ and has length in $O(|\sigma|^2)$ since $|y_{k+1}| \leq |y_k| + |\texttt{rea}(\sigma)|$ for all k. ◀

The set I_σ, represented via $\texttt{pr}[\sigma^\omega](\varepsilon)$, is interesting because it characterises the configurations from which a σ-labelled cycle can be traversed infinitely many times, i.e., it characterises nontermination.

Indeed, the following lemma reduces nontermination to reachability:

▶ **Lemma 5.3** (Existence of infinite runs). *(i) There exists an infinite sequence $x = x_0 \xrightarrow{\sigma} x_1 \xrightarrow{\sigma} x_2 \cdots$ starting from x if, and only if, $\texttt{pr}[\sigma^\omega](\varepsilon) \preccurlyeq x$.*
(ii) There exists an infinite run in S that starts from (q, x) and visits a given $q' \in Q$ infinitely many times if, and only if, q' is on an elementary cycle of S and $(q, x) \xrightarrow{*} (q', \texttt{pr}[\sigma_{q'}^\omega](\varepsilon))$.

Proof.

(i) Write y for $\texttt{pr}[\sigma^\omega](\varepsilon)$. The proof of Lemma 5.2 shows that, unless $y = \bot$, $y = \texttt{pr}[\sigma](y)$ and thus $y \xrightarrow{\sigma} y$.
($\Longleftarrow$): Since $x \succcurlyeq y$, we have $x \xrightarrow{\sigma} y \xrightarrow{\sigma} y \xrightarrow{\sigma} \cdots$ if $\sigma \neq \varepsilon$, and $x \xrightarrow{\sigma} x \xrightarrow{\sigma} x \xrightarrow{\sigma} \cdots$ in the degenerate case where $\sigma = \varepsilon$.
($\Longrightarrow$): We assume $\sigma \neq \varepsilon$ since otherwise $x \succcurlyeq \varepsilon = \texttt{pr}[\sigma^\omega](\varepsilon)$ holds trivially. The infinite sequence $x_0 \xrightarrow{\sigma} x_1 \xrightarrow{\sigma} x_2 \xrightarrow{\sigma} \cdots$ satisfies $x_0 \succcurlyeq \texttt{pr}[\sigma^k](x_k) \succcurlyeq \texttt{pr}[\sigma^k](\varepsilon)$ for all $k \in \mathbb{N}$. Thus $\texttt{pr}[\sigma^\omega](\varepsilon) \neq \bot$ and $x = x_0 \succcurlyeq \texttt{pr}[\sigma^\omega](\varepsilon)$.

(ii) is an immediate consequence of (i). ◀

We can now solve our favorite liveness problems. Here a *co-Büchi run* is an infinite run that visits a given location finitely many times, while *repeated coverability* is the question whether there exists an infinite run $c_0 \to c_1 \to c_2 \to \cdots$ such that infinitely many c_i's are above a given (q, x).

▶ **Theorem 5.4.** *Nontermination, existence of a Büchi run, existence of a co-Büchi run, and repeated coverability are all* NP*-complete for flat LCMs.*

Proof. See Appendix C for NP-hardness. For membership in NP, Lemmas 5.2 and 5.3 reduce nontermination and existence of a Büchi run to a reachability question that we can solve thanks to Theorem 4.3. Note that in flat machines, the existence of a co-Büchi run is easily reduced to a positive Büchi property, so that there only remain to provide an NP-algorithm for repeated coverability.

For this let us define more generally $I_\sigma(x)$ as $\bigcap_{k=0,1,2,\ldots} \texttt{Pre}[\sigma^k](\uparrow x)$, so that I_σ really is shorthand for $I_\sigma(\varepsilon)$. For a location q on a σ_q-labelled cycle, $I_{\sigma_q}(x)$ characterises a form of *repeated coverability* since $y \in I_{\sigma_q}(x)$ iff there is an infinite run from (q, y) such that the channel contains a superword of x every time q is (re)visited. Using some temporal logic, this could be written under the form

$$y \in I_{\sigma_q}(x) \iff (q, y) \models_\exists \mathsf{GF}q \wedge \mathsf{G}(q \implies \mathit{chan} \geq x)\,.$$

The proof of Lemma 5.2 can be extended to the computation of $I_\sigma(x)$. One obtains $I_{\sigma_q}(x) = \uparrow y_0 \cap \uparrow y_1 \cap \cdots \cap \uparrow y_K$ for some K in $O(|\sigma| \cdot |x|)$. Note however that now the $(y_k)_k$ sequence does not necessarily satisfy $y_0 \preccurlyeq y_1$, so that $I_\sigma(x)$ will have in general several minimal elements, and possibly exponentially many. In fact already $\uparrow y_0 \cap \uparrow y_1$ may have exponentially many minimal elements (see [28, § 6.3]). Thus our algorithm for repeated coverability will represent $I_\sigma(x)$ as a conjunction of $K+1$ subword constraints, not via a set of minimal elements. This is sufficient for our purposes: the algorithm for reachability is easily extended to questions of the form "can we reach some (q, x) with $x \preccurlyeq y_0 \wedge \cdots \wedge x \preccurlyeq y_K$?" ◀

Unboundedness reduces to reachability in a very similar way. We say that a sequence of actions σ is *increasing* if $u^{\ell_v} \preccurlyeq v^{\ell_v - 1}$ (and $\ell_v > 0$) for $u \overset{\text{def}}{=} \texttt{rea}(\sigma)$, $v \overset{\text{def}}{=} \texttt{wri}(\sigma)$ and $\ell_v \overset{\text{def}}{=} |v|$. Now $\texttt{pr}[\sigma^\omega](\varepsilon)$ and increasingness of σ characterise unbounded reachability sets.

▶ **Lemma 5.5** (Proof in Appendix B.1). *Let $x \in \Sigma^*$ be some channel contents and σ a sequence of channel actions. T.f.a.e.:*

(i) *For all $k \in \mathbb{N}$ there exists x_k with $x \xrightarrow{\sigma^*} x_k$ and $|x_k| \geq k$.*
(ii) *There exists an infinite unbounded sequence $x \xrightarrow{\sigma^*} x_1 \xrightarrow{\sigma^*} x_2 \xrightarrow{\sigma^*} \cdots$ with $|x_1| < |x_2| < \cdots$.*
(iii) *σ is increasing and $x \succcurlyeq \texttt{pr}[\sigma^\omega](\varepsilon)$.*

▶ **Lemma 5.6** (Existence of unbounded runs). *In a flat LCM, t.f.a.e.*

(i) *The reachability set $\texttt{Post}^*(q, x)$ is infinite.*
(ii) *There is an unbounded run starting from (q, x).*
(iii) *$(q, x) \xrightarrow{*} (q', \texttt{pr}[\sigma_{q'}^\omega](\varepsilon))$ for some control location q' with an increasing $\sigma_{q'}$.*

Proof (sketch).

(ii $\implies$ iii): In an unbounded run, there must be a control location q' that is visited infinitely many times with associated channel contents that are unbounded. Since from q' one can only return to q' by running through the cycle around q', hence performing $\sigma_{q'}$ some number of times, the first visit of q' is some (q', x') satisfying case **(ii)** of Lemma 5.5. We deduce that σ_q' is increasing and that $x' \succcurlyeq \texttt{pr}[\sigma_{q'}^\omega](\varepsilon)$ as in case **(iii)** of the Lemma.
(iii $\implies$ ii): by Lemma 5.5 there exists an unbounded run starting from $(q', \texttt{pr}[\sigma_{q'}^\omega](\varepsilon))$. Hence there is one starting from (q, x).
(i $\iff$ ii): is an application of Kőnig's Lemma, not specific to LCMs, see e.g. [41, §6]. ◀

We can thus reduce unboundedness to reachability of an increasing cycle. With the NP-hardness results proven in Appendix C, one now obtains:

▶ **Theorem 5.7.** *Unboundedness for flat LCMs is* NP*-complete.*

6 Conclusion

We analysed the behaviour of the backward-reachability algorithm for lossy channel machines when a cycle of channel actions can be performed arbitrarily many times. This provides complexity bounds on the size of runs that follow a bounded path scheme of the form $\sigma_1^*\rho_1\sigma_2^*\rho_2\ldots\sigma_m^*\rho_m$, with applications in the verification of flat systems, or in bounded verification for general systems. The main result is an NP upper bound for reachability and, by reduction, several other verification problems like unboundedness or existence of a Büchi run.

Natural directions for future work include extending our approach to deal with richer verification problems, like temporal logic model checking. It would also be interesting to consider more expressive models, like the partially lossy channel systems from [32] or the higher-order lossy channel systems and priority channel systems from [29].

References

1 P. A. Abdulla, K. Čerāns, B. Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1/2):109–127, 2000. doi:10.1006/inco.1999.2843.

2 P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004. doi:10.1023/B:FORM.0000033962.51898.1a.

3 P. A. Abdulla, J. Deneux, J. Ouaknine, and J. Worrell. Decidability and complexity results for timed automata via channel machines. In *Proc. ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 1089–1101. Springer, 2005. doi:10.1007/11523468_88.

4 P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996. doi:10.1006/inco.1996.0053.

5 B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. *Formal Methods in System Design*, 14(3):237–255, 1999. doi:10.1023/A:1008719024240.

6 B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc. CAV '94*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67. Springer, 1994. doi:10.1007/3-540-58179-0_43.

7 A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Theoretical Computer Science*, 221(1–2):211–250, 1999. doi:10.1016/S0304-3975(99)00033-X.

8 M. Bozga, R. Iosif, and F. Konecný. Safety problems are NP-complete for flat integer programs with octagonal loops. In *Proc. VMCAI 2014*, volume 8318 of *Lecture Notes in Computer Science*, pages 242–261. Springer, 2014. doi:10.1007/978-3-642-54013-4_14.

9 M. Bozga, R. Iosif, and Y. Lakhnech. Flat parametric counter automata. *Fundamenta Informaticae*, 91(2):275–303, 2009. doi:10.3233/FI-2009-0044.

10 D. Brand and P. Zafiropulo. On communicating finite-state machines. *JACM*, 30(2):323–342, 1983. doi:10.1145/322374.322380.

11 G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996. doi:10.1006/inco.1996.0003.

12 P. Cégielski, I. Guessarian, Y. Lifshits, and Y. V. Matiyasevich. Window subsequence problems for compressed texts. In *Proc. CSR 2006*, volume 3967 of *Lecture Notes in Computer Science*, pages 127–136. Springer, 2006. doi:10.1007/11753728_15.

13 P. Chambart and Ph. Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Proc. LICS 2008*, pages 205–216. IEEE Comp. Soc. Press, 2008. doi:10.1109/LICS.2008.47.

14 P. Chambart and Ph. Schnoebelen. Toward a compositional theory of leftist grammars and transformations. In *Proc. FOSSACS 2010*, volume 6014 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2010. doi:10.1007/978-3-642-12032-9_17.

15 H. Comon and Y. Jurski. Multiple counters automata, safety analysis, and Presburger arithmetic. In *Proc. CAV '98*, volume 1427 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 1998. doi:10.1007/BFb0028751.

16 N. Decker, P. Habermehl, M. Leucker, A. Sangnier, and D. Thoma. Model-checking counting temporal logics on flat structures. In *Proc. CONCUR 2017*, volume 85 of *Leibniz International Proceedings in Informatics*, pages 29:1–29:17. Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.29.

17 S. Demri, A. K. Dhar, and A. Sangnier. Taming past LTL and flat counter systems. *Information and Computation*, 242:306–339, 2015. doi:10.1016/j.ic.2015.03.007.

18 S. Demri, A. Finkel, V. Goranko, and G. van Drimmelen. Model-checking CTL* over flat Presburger counter systems. *Journal of Applied Non-Classical Logics*, 20(4):313–344, 2010. doi:10.3166/jancl.20.313-344.

19 S. Demri, F. Laroussinie, and Ph. Schnoebelen. A parametric analysis of the state explosion problem in model checking. *Journal of Computer and System Sciences*, 72(4):547–575, 2006. doi:10.1016/j.jcss.2005.11.003.

20 J. Esparza, P. Ganty, and R. Majumdar. A perfect model for bounded verification. In *Proc. LICS 2012*, pages 285–294. IEEE Comp. Soc. Press, 2012. doi:10.1109/LICS.2012.39.

21 A. Finkel. Decidability of the termination problem for completely specificied protocols. *Distributed Computing*, 7(3):129–135, 1994. doi:10.1007/BF02277857.

22 A. Finkel and M. Praveen. Verification of flat FIFO systems. In *Proc. CONCUR 2019*, volume 140 of *Leibniz International Proceedings in Informatics*, pages 12:1–12:17. Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.12.

23 A. Finkel and M. Praveen. Verification of flat FIFO systems. *Logical Methods in Comp. Science*, 16(4), 2020.

24 A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.

25 L. Fribourg and H. Olsén. A decompositional approach for computing least fixed-points of datalog programs with $\mathcal{Z}$-counters. *Constraints*, 2(3/4):305–335, 1997. doi:10.1023/A:1009747629591.

26 D. Gabelaia, A. Kurucz, F. Wolter, and M. Zakharyaschev. Non-primitive recursive decidability of products of modal logics with expanding domains. *Annals of Pure and Applied Logic*, 142(1–3):245–268, 2006. doi:10.1016/j.apal.2006.01.001.

27 P. Ganty and R. Iosif. Interprocedural reachability for flat integer programs. In *Proc. FCT 2015*, volume 9210 of *Lecture Notes in Computer Science*, pages 133–145. Springer, 2015. doi:10.1007/978-3-319-22177-9_11.

28 J. Goubault-Larrecq, S. Halfon, P. Karandikar, K. Narayan Kumar, and Ph. Schnoebelen. The ideal approach to computing closed subsets in well-quasi-orderings. In *Well Quasi-Orders in Computation, Logic, Language and Reasoning*, volume 53 of *Trends in Logic*, chapter 3, pages 55–105. Springer, 2020. doi:10.1007/978-3-030-30229-0_3.

29 Ch. Haase, S. Schmitz, and Ph. Schnoebelen. The power of priority channel systems. *Logical Methods in Comp. Science*, 10(4:4), 2014. doi:10.2168/LMCS-10(4:4)2014.

30 J. Hopcroft and J.-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979. doi:10.1016/0304-3975(79)90041-0.

31 P. Karandikar and Ph. Schnoebelen. Generalized Post embedding problems. *Theory of Computing Systems*, 56(4):697–716, 2015. doi:10.1007/s00224-014-9561-9.

32 Ch. Köcher. Reachability problems on partially lossy queue automata. In *Proc. RP 2019*, volume 11674 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 2019. doi:10.1007/978-3-030-30806-3_12.

33 L. Kuhtz and B. Finkbeiner. Weak Kripke structures and LTL. In *Proc. CONCUR 2011*, volume 6901 of *Lecture Notes in Computer Science*, pages 419–433. Springer, 2011. doi:10.1007/978-3-642-23217-6_28.

34 S. Lasota and I. Walukiewicz. Alternating timed automata. *ACM Trans. Computational Logic*, 9(2), 2008. doi:10.1145/1342991.1342994.

35 J. Leroux, V. Penelle, and G. Sutre. The context-freeness problem is coNP-complete for flat counter systems. In *Proc. ATVA 2014*, volume 8837 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2014. doi:10.1007/978-3-319-11936-6_19.

36 J. Leroux and G. Sutre. Flat counter automata almost everywhere! In *Proc. ATVA 2005*, volume 3707 of *Lecture Notes in Computer Science*, pages 489–503. Springer, 2005. doi:10.1007/11562948_36.

37 M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.

38 N. Markey and Ph. Schnoebelen. A PTIME-complete matching problem for SLP-compressed words. *Information Processing Letters*, 90(1):3–6, 2004. doi:10.1016/j.ipl.2004.01.002.

39 S. Schmitz. Complexity hierarchies beyond Elementary. *ACM Trans. Computation Theory*, 8(1), 2016. doi:10.1145/2858784.

40 S. Schmitz and Ph. Schnoebelen. Multiply-recursive upper bounds with Higman's lemma. In *Proc. ICALP 2011*, volume 6756 of *Lecture Notes in Computer Science*, pages 441–452. Springer, 2011. doi:10.1007/978-3-642-22012-8_35.

41 Ph. Schnoebelen. Lossy counter machines decidability cheat sheet. In *Proc. RP 2010*, volume 6227 of *Lecture Notes in Computer Science*, pages 51–75. Springer, 2010. doi:10.1007/978-3-642-15349-5_4.

42 B. Vauquelin and P. Franchi-Zannettacci. Automates à file. *Theoretical Computer Science*, 11(2):221–225, 1980. doi:10.1016/0304-3975(80)90047-X.

43 T. Yamamoto, H. Bannai, S. Inenaga, and M. Takeda. Faster subsequence and don't-care pattern matching on compressed texts. In *Proc. CPM 2011*, volume 6661 of *Lecture Notes in Computer Science*, pages 309–322. Springer, 2011. doi:10.1007/978-3-642-21458-5_27.

A Some SLP algorithms

We describe here some SLP algorithms that are not readily available in the literature (as far as we know). Formally, by "an SLP X" we mean a grammar (Σ, N, X, P) where $X \in N$ is the axiom (a non terminal), where Σ is the set of terminal letters, and where the production rules in P are either $A_i \to a$ or $A_i \to A_j A_k$ for some $a \in \Sigma$ and some nonterminals A_i, A_j, A_k with $i < j, k$. There is exactly one production rule for each $A_i \in N$, so that each A_i defines a unique word $L(A_i) \in \Sigma^*$.

A.1 Deciding $X \preccurlyeq v^n$

Deciding $X \preccurlyeq Y$ between SLPs is a difficult problem, PP-hard as show in [37]. When x (or y) is a plain word, the problem has polynomial-time solutions [38, 12, 43].

Here we consider the special case where Y is some v^n.

▶ **Proposition A.1.** *Deciding whether $X \preccurlyeq v^n$, where X is an SLP, v is a plain word, and n is a fractional exponent, can be done in time $O(\|X\| \cdot |v| + |v|^2 + \log n)$.*

Proof. For $v \neq \varepsilon$ and some word x such that $\text{alph}(x) \subseteq \text{alph}(v)$, let us define $p(x,v)$ as the smallest fractional power such that $x \preccurlyeq v^p$. Now $p(x,v)$ satisfies the following equalities:

$$\begin{aligned} p(\varepsilon, v) &= 0 \\ p(a, v) &= \frac{i}{|v|}, \text{ if the first occurrence of } a \text{ in } v \text{ is at position } i, \\ p(x\,y, v) &= p(x,v) + p(y, v_{(j)}), \text{ if } p(x,v) \text{ is some } q + \frac{j}{|v|} \text{ with } q \in \mathbb{N}. \end{aligned} \tag{12}$$

Using Equation (12) leads to a dynamic programming algorithm computing $p(X,v)$ for an SLP X. After checking that $\text{alph}(X) \subseteq \text{alph}(v)$, one computes the values of all $p(A, v_{(i)})$ for $i = 1, \ldots, |v|$ and A a nonterminal in SLP X. Each of these $O(\|X\| \cdot |u|)$ values is computed in time $O(1)$ if one precomputes the first occurrences of letters in the cyclic shifts of v, say in time $O(|v|^2)$. Finally, one only has to compare $p(X,v)$ with n. ◀

A.2 Computing X/v^k

▶ **Proposition A.2.** *Building a SLP for X/v^k, where X is an SLP, v is a plain word, and $k \in \mathbb{N}$, can be done in time* $\mathsf{poly}(\|X\| + |v| + \log n)$.

Proof. For given ℓ, deciding whether X/v^k has length at least ℓ is easy: One just applies the definition, builds an SLP X' for the suffix of length $|X| - \ell$ of X, and checks that it is a subword of v^k with Proposition A.1.

Thus one can computes $|X/v^k|$ by finding the length of the result via dichotomic search, repeating the previous process $\log|X|$, i.e., $O(\|X\|)$, times.[6] ◀

B Forward reachability techniques

We collect in this section some proofs relying on forward-reachability analysis.

Let us reuse notations from [2] and define a partial function $x \ominus u$ between channel contents as follows:

$$x \ominus u \overset{\text{def}}{=} \begin{cases} x & \text{if } u = \varepsilon, \\ \text{undefined} & \text{if } x = \varepsilon \text{ and } u \neq \varepsilon, \\ x' \ominus u' & \text{if } x = ax' \text{ and } u = au' \text{ for some } a \in \Sigma, \\ x' \ominus u & \text{if } x = ax' \text{ and } u = bu' \text{ for some } a \neq b \in \Sigma. \end{cases} \tag{13}$$

Observe that $x \ominus u$ is defined if, and only if, $u \preccurlyeq x$. Note also that

$$(x \ominus u) \ominus u' = x \ominus (u\,u')\,, \tag{14}$$

and that $\ominus$ is left-monotone and commutes with with concatenation when defined:

$$\text{if } u \preccurlyeq x \text{ then for all } x' : \begin{cases} x \preccurlyeq x' \text{ implies } x \ominus u \preccurlyeq x' \ominus u\,, \\ (x \ominus u) \cdot x' = (x\,x') \ominus u\,. \end{cases} \tag{15}$$

Now $x \ominus u$ captures the forward effects of $?u$ actions in LCMs:

▶ **Lemma B.1.** $x \xrightarrow{?u} y$ *iff* $y \preccurlyeq x \ominus u$.

[6] A better, dynamic programming, algorithm exists but here we aim for the simplest feasability proof.

We can also use $\ominus$ to characterise the outcome of arbitrary sequences of actions.

▶ **Lemma B.2.** *Let $\sigma \in Act_\Sigma^*$ and $x, y \in \Sigma^*$ be any sequence of actions and channel contents.*

$$x' \xrightarrow{\sigma} y \text{ for some } x' \preccurlyeq x \text{ iff } x \xrightarrow{\sigma} \text{ and } y \preccurlyeq \big(x \cdot \mathtt{wri}(\sigma)\big) \ominus \mathtt{rea}(\sigma)\,.$$

Proof. By induction on the length of σ. The existential quantification on some $x' \preccurlyeq x$ accounts for the case where $\sigma = \varepsilon$ is the empty sequence.

For the inductive step, we consider two cases:

1. $\sigma = {!w} \cdot \sigma'$: For the "$\Longrightarrow$" direction, $x' \xrightarrow{\sigma} y$ implies $x \xrightarrow{\sigma}$ and $x' \xrightarrow{!w} x'' \xrightarrow{\sigma'} y$ for some $x'' \preccurlyeq x'\,w$, which implies

$$\begin{aligned} y &\preccurlyeq \big(x'w \cdot \mathtt{wri}(\sigma')\big) \ominus \mathtt{rea}(\sigma') && \text{by ind. hyp.,}\\ &= \big(x' \cdot \mathtt{wri}(\sigma)\big) \ominus \mathtt{rea}(\sigma) && \text{since } \sigma = {!w} \cdot \sigma',\\ &\preccurlyeq \big(x \cdot \mathtt{wri}(\sigma)\big) \ominus \mathtt{rea}(\sigma) && \text{by monotonicity.} \end{aligned}$$

For the "$\Longleftarrow$" direction, we know that $y \preccurlyeq \big(x.\,\mathtt{wri}(\sigma)\big) \ominus \mathtt{rea}(\sigma) = \big(xw.\,\mathtt{wri}(\sigma')\big) \ominus \mathtt{rea}(\sigma')$, so the ind. hyp. tells us that $x' \xrightarrow{\sigma'} y$ for some $x' \preccurlyeq xw$. We deduce $x \xrightarrow{!w} x' \xrightarrow{\sigma'} y$.

2. $\sigma = {?w} \cdot \sigma'$: For the "$\Longrightarrow$" direction, $x' \xrightarrow{\sigma} y$ implies $x' \xrightarrow{?w} x'' \xrightarrow{\sigma'} y$ for some $x'' \preccurlyeq x' \ominus w$. We have

$$\begin{aligned} y &\preccurlyeq \big(x''.\,\mathtt{wri}(\sigma')\big) \ominus \mathtt{rea}(\sigma') && \text{by ind. hyp.,}\\ &\preccurlyeq \big([x \ominus w] \cdot \mathtt{wri}(\sigma')\big) \ominus \mathtt{rea}(\sigma') && \text{by monotonicity,}\\ &= \big(x \cdot \mathtt{wri}(\sigma')\big) \ominus \big(w \cdot \mathtt{rea}(\sigma')\big) && \text{by Equations (14) and (15),}\\ &= (x \cdot \mathtt{wri}(\sigma)) \ominus \mathtt{rea}(\sigma) && \text{since } \sigma = {?w} \cdot \sigma'. \end{aligned}$$

For the "$\Longleftarrow$" direction, we know that $x \ominus w$ is defined since $x \xrightarrow{\sigma}$. We also know that $y \preccurlyeq \big(x.\,\mathtt{wri}(\sigma)\big) \ominus \mathtt{rea}(\sigma) = \big(x.\,\mathtt{wri}(\sigma')\big) \ominus (w \cdot \mathtt{rea}(\sigma')) = \big((x \ominus w).\,\mathtt{wri}(\sigma')\big) \ominus \mathtt{rea}(\sigma')$, so by ind. hyp. there is some $x' \preccurlyeq x \ominus w$ with $x' \xrightarrow{\sigma'} y$. We deduce $x \xrightarrow{?w} x' \xrightarrow{\sigma'} y$. ◀

B.1 Proof of Lemma 5.5

Write u, v for $\mathtt{rea}(\sigma)$, $\mathtt{wri}(\sigma)$.

(ii $\Longrightarrow$ iii): we only have to prove that σ_q is increasing since Lemma 5.3 entails $x \not\succcurlyeq \mathtt{pr}[\sigma^\omega](\varepsilon)$ already.
By assumption, there is a sequence $x_1, x_2, \ldots$ of channel contents of increasing length, and some numbers $n_1, n_2, \ldots$ in $\mathbb{N}$ such that $x \xrightarrow{\sigma^{n_i}} x_i$. W.l.o.g. we can assume $n_1 < n_2 < \cdots$. With Lemma B.2 we deduce $x_i \preccurlyeq (x\,v^{n_i}) \ominus u^{n_i}$, hence $u^{n_i}\,x_i \preccurlyeq x\,v^{n_i}$, for all $i = 1, 2, \ldots$ If $u = \varepsilon$, σ is trivially increasing, so assume $|u| > 0$ and write $m = |x|$: we get $u^{n_i - m}x_i \preccurlyeq v^{n_i}$ for all i such that $n_i \geq m$. Now take i such that $|x_i| \geq (m+1)|v|$ (and such that $n_i > m$): we get $u^{n_i - m} \preccurlyeq v^{n_i - m - 1}$. We now applies Lemma 6.2 from [2]: "if there is some $k \geq 1$ such that $w_1^k \preccurlyeq w_2^{k-1}$ (for two words w_1, w_2), then in particular one can choose $k = |w_2|$". This yields $u^{|v|} \preccurlyeq v^{|v|-1}$, i.e., σ is increasing.

(iii $\Longrightarrow$ i): we assume that σ is increasing, i.e., $u^{|v|} \preccurlyeq v^{|v|-1}$, and that $x \not\succcurlyeq \mathtt{pr}[\sigma_q^\omega](\varepsilon)$. The second assumption entails that $x \xrightarrow{\sigma^n}$ for all n. The first assumption entails $v^k \preccurlyeq x\,v^{k|v|} \ominus u^{k|v|}$, hence $x \xrightarrow{\sigma^{k|v|}} v^k$ by Lemma B.2, for all $k \in \mathbb{N}$.

(i $\Longrightarrow$ ii): is an application of Kőnig's Lemma, not specific to LCMs, see e.g. [41, §6].

C NP-hardness for flat LCMs and flat FIFO machines

LCMs are derived from FIFO automata [42, 10] and our NP-hardness results apply to both models. *FIFO automata*, sometimes called *queue automata*, or *communicating finite state machines*, are reliable channel machines where messages are never lost. Their operational semantics is based on a reliable notion of steps, formally given by $x \xrightarrow{!\,w}_{\text{rel}} y \overset{\text{def}}{\iff} y = xw$ and $x \xrightarrow{?\,w}_{\text{rel}} y \overset{\text{def}}{\iff} wy = x$, to be compared with Equation (2). This is extended to $x \xrightarrow{\sigma}_{\text{rel}} y$, $c \xrightarrow{*}_{\text{rel}} c'$, etc., as for LCMs.

C.1 Proof of Theorem 2.1: NP-hardness for acyclic machines

We first show hardness for reachability and reduce from SAT. Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a 3CNF with Boolean variables among $V = \{v_1, \ldots, v_n\}$. With φ we associate a machine S_φ as illustrated below in Figure 2.

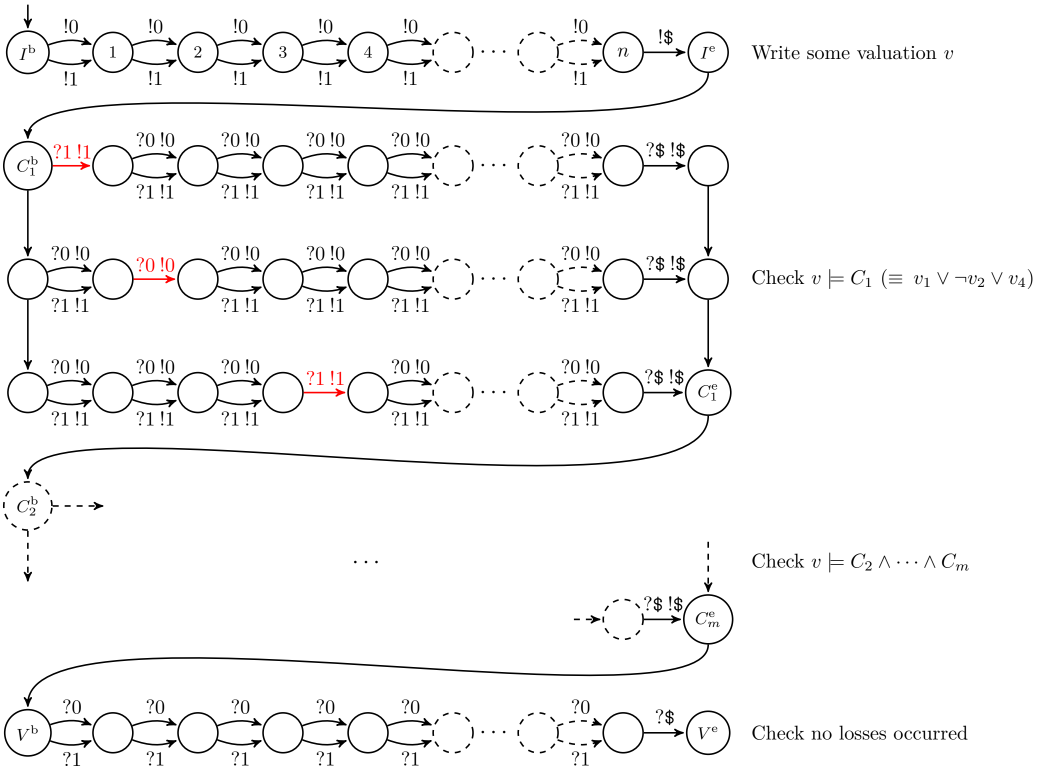


Figure 2 LCM S_φ for satisfiability of $\varphi = (v_1 \vee \neg v_2 \vee v_4) \wedge C_2 \cdots \wedge C_m$.

Let us explain informally how S_φ operates. Starting from I^b it first reaches I^e while writing in the channel a word of the form $w\$$ with $w \in \{0,1\}^n$. This word encodes a valuation of the Boolean variables and carries an end marker \$. Then S_φ crosses from C_1^b to C_1^e: this requires reading the valuation on the channel and checking that it satisfies C_1. For this S_φ has to choose the line corresponding to one of the three literals in C_1, in fact choose one literal made true by the valuation. During this check, the valuation is written back on the channel. Then S_φ checks that the remaining clauses, C_2 to C_m, are satisfied by the valuation, each time reading the valuation and writing it back on the channel. Finally, the last leg from V^b to V^e checks that no message has been lost during all this run.

It is now clear that $(I^{\mathrm{b}},\varepsilon) \xrightarrow{*} (V^{\mathrm{e}},\varepsilon)$ in S_φ if, and only if, φ is satisfiable. The reasoning holds for lossy LCMs and for reliable FIFO automata. We have thus reduced SAT to the reachability problem for both types of acyclic machines.

▶ Remark C.1. The construction of S_φ can be simplified at the cost of making the reduction perhaps less obviously correct: one can either omit the end-marker symbol \$ since in the end the machine checks that no message was lost (thus a binary alphabet suffices), or one can stop the machine at C^{e}_m, getting rid of the V^{b} to Ve part, since the markers ensure that the valuation read while checking a clause C_i is indeed the full valuation written at the previous stage. ⌟

For hardness of nontermination and unboundedness we adapt the previous reduction by adding a single cycle $V^{\mathrm{e}} \xrightarrow{!\$} V^{\mathrm{e}}$ on the last control location. Starting from $(I^{\mathrm{b}},\varepsilon)$, the modified S_φ has an infinite run iff it has an unbounded run iff φ is satisfiable.

The above reductions adapt to flat VASSes and lossy VASSes, i.e., channel machines with unary alphabet, provided that we allow $2n$ channels (or counters) for a valuation on n Boolean variables.

C.2 Proof of Theorem 2.2: NP-hardness for single-path machines

We first show hardness for reachability. For this we reduce from SAT. So let us consider a 3CNF formula φ with Boolean variables among $V = \{v_1,\ldots,v_n\}$. Let us say $\varphi = (v_2 \vee \neg v_3 \vee \neg v_n) \wedge C_2 \wedge \cdots \wedge C_m$, with m clauses.

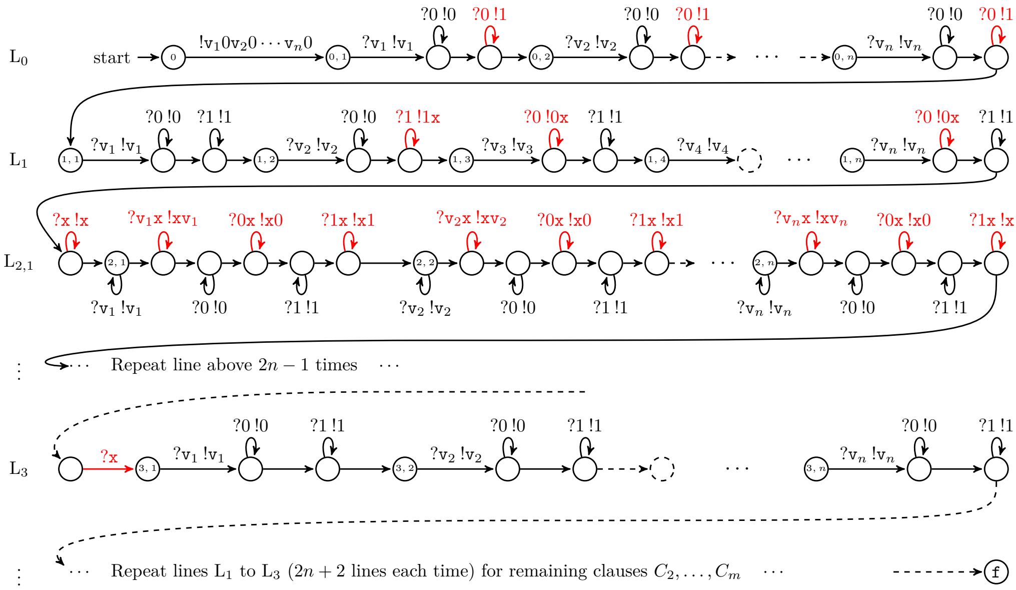


Figure 3 Single-path LCM for satisfiability of $\varphi = (v_2 \vee \neg v_3 \vee \neg v_n) \wedge C_2 \cdots \wedge C_m$.

With φ we associate S_φ, the single-path flat LCM described in Figure 3. This LCM has $O(mn^2)$ control locations[7], and is organised as a series of distinct operations on the channel contents. The operations are grouped in lines and we describe them informally.

[7] Our reduction insists on using only one channel. With multiple channels the same idea would use $O(n+m)$ control locations.

L$_0$, choosing a valuation nondeterministically: S_φ first writes $\mathtt{v}_1 0 \mathtt{v}_2 0 \ldots \mathtt{v}_n 0$ on the channel. This is our encoding for the valuation that is 0 for all variables. Then S_φ reads the valuation and write it back, possibly changing any 0 value with a 1 (this happens at the red-coloured actions), and thus picking an arbitrary valuation nondeterministically. Here we see how the $\mathtt{v}_1, \ldots, \mathtt{v}_n$ markers are used to check positions inside the valuation.

L$_1$, marking where clause C_1 is validated: S_φ now checks whether the valuation stored on the channel makes C_1 true. In this example, we assume that C_1 is $v_2 \vee \neg v_3 \vee \neg v_n$. Again S_φ reads the valuation and writes it back. However, if it reads $\mathtt{v}_2 1$ or $\mathtt{v}_3 0$ or $\mathtt{v}_n 0$, it writes it back followed by a special checkmark symbol $\mathtt{x}$ that "means C_1 has been validated" (see red actions). Note that as many as 3 occurrences of $\mathtt{x}$ can be inserted in the encoding of the valuation.

L$_{2,1}$, pushing $\mathtt{x}$ to the head of the valuation encoding: S_φ now pushes any checkmark symbol to the left. This is done along the $L_{2,1}$ line. While the valuation is read and written back as usual (black actions), any symbol preceding a $\mathtt{x}$ can swap position with it (red actions).

L$_{2,2}$, ..., L$_{2,2n}$, more pushing $\mathtt{x}$ to the left: this behaviour is repeated $2n$ times in total, so that any $\mathtt{x}$ can be pushed completely to the left of the valuation. In case of multiple occurrences of $\mathtt{x}$, we just need one of them to reach the head of the valuation so we assume that the other ones will just be lost.

L$_3$, checking that clause C_1 has been validated: Now S_φ knows where to expect $\mathtt{x}$. The machine can only proceed if indeed a $\mathtt{x}$ is present in the channel, in front of the valuation, and thus if the valuation on the channel satisfies C_1. The rest of the line reads and writes back the valuation, clearing it of any remaining $\mathtt{x}$'s.

Same treatment for the remaining clauses $C_2, \ldots, C_m$: S_φ now continues with similar locations and rules checking that the remaining clauses are validated.

Note that, once the valuation has been picked nondeterministically (in L$_1$), it cannot be modified. Also note that the machine will block if one of the $\mathtt{v}_i$ markers is lost before the last clause has been validated. If one of the 0/1 values of the valuation is lost, this value cannot be used any more for checkmarking a validated clause. Such message losses do not lead to any incorrect behaviour, they can only hinder the validation of a clause.

Finally, starting from $(0, \varepsilon)$, S_φ can reach its final location $\mathtt{f}$ iff φ is satisfiable.

Now the reduction extends to show prove NP-hardness of unboundedness for single-path LCMs with exactly the same adaptation as in the proof for acyclic LCMs. For hardness of nontermination a little more work is needed since every cycle where S_φ reads the valuation and writes it back could become a nonterminating cycle if all but one letter are lost. One possible trick to overcome this is to have two copies of the alphabet, say of two different colours, and to ensure that in all its phases the machine reads in one colour and writes back in the other, so that the valuation is always read and written in alternating colours. Once this is implemented, the system cannot have infinite runs as is. Adding a single loop on $\mathtt{f}$, the final control location, as we did for acyclic LCMs, now provides a correct reduction from SAT to nontermination for single-path LCMs.

The idea behind this reduction can easily be adapted so that it applies to single-path VASSes and lossy VASSes, or equivalently, to channel machines with a unary alphabet. One uses $2n$ channels (or counters) for storing the valuation and m distinct counters for marking the clauses that have been validated.

Restricting to a binary alphabet on a single channel is equally easy for reliable FIFO automata, but more difficult when message losses have to be taken care of. Therefore we won't attempt it in this preliminary version.

D Multiple channels

The analysis we conducted in Section 3 carries over without any difficulty to systems with multiple channels. Lemma 3.3 and Theorem 3.4 remain valid since, once σ and k have been fixed, computing $\texttt{pr}[\sigma^k](\langle x_1, \ldots, x_c\rangle)$ for a system with c channels can be done independently for each of the c channels: one only needs to distribute the actions on σ to their corresponding channel, so that $\texttt{rea}(\sigma)$ now is some tuple $\langle u_1, \ldots, u_c\rangle$. In particular the bound in Theorem 3.7 becomes

$$L\big(\sigma, \langle x_1, \ldots, x_c\rangle\big) \leq \max_{i=1}^{c} |x_i| \cdot (|u_i| + 1)\,, \quad \text{where } \texttt{rea}(\sigma) = \langle u_1, \ldots, u_c\rangle\,.$$

Realizability Without Symmetry

Haruka Tomita
Research Institute for Mathematical Sciences, Kyoto University, Japan

Abstract

In categorical realizability, it is common to construct categories of assemblies and modest sets from applicative structures. In this paper, we introduce several classes of applicative structures and apply the categorical realizability construction to them. Then we obtain closed multicategories, closed categories and skew closed categories, which are more general categorical structures than Cartesian closed categories and symmetric monoidal closed categories. Moreover, we give the necessary and sufficient conditions for obtaining closed multicategories and closed categories of assemblies.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases Realizability, combinatory algebra, closed multicategory, closed category, skew closed category

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.38

Funding This work was supported by JST ERATO Grant Number JPMJER1603, Japan.

Acknowledgements I would like to express the deepest appreciation to my supervisor Masahito Hasegawa for a lot of his support, encouragement and advice. Many of his suggestions are incorporated into this paper. I am also grateful to Naohiko Hoshino for helpful discussions and lending his expertise on realizability. I also want to thank Stephen Lack for constructive discussions and comments. Thanks also to anonymous reviewers for variable suggestions and comments.

1 Introduction

In categorical realizability, we construct categories of assemblies and modest sets from applicative structures. The best known usage is applying this construction to partial combinatory algebras (PCAs) which is a class of applicative structures close to the models of the lambda calculus, as in [12]. From PCAs, we obtain Cartesian closed categories of assemblies and use these categories for models of various logics and programming languages like PCF.

The construction of categories of assemblies does not depend on particular structures of applicative structures. Hence we may apply this construction to other classes of applicative structures. Indeed, another usage is introduced in [3] [2], where by applying the construction to **BCI**-algebras, we can obtain symmetric monoidal closed categories (SMCCs) and use them for models of linear logics and languages.

In this paper, by applying the categorical realizability construction to more general classes of applicative structures than PCAs and **BCI**-algebras, we investigate further correspondences between categorical structures of assemblies and classes of applicative structures. To make assemblies on an applicative structure a category, it is sufficient to assume two elements **B** (corresponding to the lambda term $\lambda xyz.x(yz)$ expressing the composition of functions) and **I** (corresponding to the lambda term $\lambda x.x$ expressing the identity function) in the applicative structure. (Here we say such an applicative structure is a **BI**-algebra.) Therefore, classes between **BCI**-algebras and **BI**-algebras may induce some categorical structures more general than SMCCs. For instance, there may exist some classes which induce non-symmetric categorical structures. Indeed, in this paper we introduce such classes of applicative structures, which induce closed multicategories, closed categories and skew closed categories.

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 38; pp. 38:1–38:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Summary of the correspondences.
(Here (†) means that not only the corresponding applicative structures induce the categorical structures, but also the converse hold in some natural conditions, like Proposition 19.).

	Applicative structures	Structure of assemblies and modest sets	Section
Known results	PCA	Cartesian closed category (†)	2.2
	$\mathbf{BCI}$-algebra	symmetric monoidal closed category (†)	2.3
New results	$\mathbf{BK}(-)^{\bullet}$-algebra	closed category	5.1
	$\mathbf{BII}^{\times}(-)^{\bullet}$-algebra	closed category (†)	4
	$\mathbf{BI}(-)^{\bullet}$-algebra	closed multicategory (†)	3
	$\mathbf{BB'II^{\bullet}}$-algebra	skew closed category	5.3
	$\mathbf{BII^{\bullet}}(-)^{\circ}$-algebra	skew closed category	5.2

Inclusions
PCAs $\subsetneq$ $\mathbf{BCI}$-algebras $\subsetneq$ $\mathbf{BII}^{\times}(-)^{\bullet}$-algebras $\subsetneq$ $\mathbf{BI}(-)^{\bullet}$-algebras $\subsetneq$ $\mathbf{BII^{\bullet}}(-)^{\circ}$-algebras
PCAs $\subsetneq$ $\mathbf{BK}(-)^{\bullet}$-algebras $\subsetneq$ $\mathbf{BII}^{\times}(-)^{\bullet}$-algebras
$\mathbf{BCI}$-algebras $\subsetneq$ $\mathbf{BB'II^{\bullet}}$-algebras $\subsetneq$ $\mathbf{BII^{\bullet}}(-)^{\circ}$-algebras

In category theory, many closed structures without tensor products have been developed. Closed multicategories, closed categories and skew closed categories are the typical ones. Each of them gives certain axiomatization of internal function spaces without using tensor products and does not have symmetries in general unlike SMCCs.

Closed multicategories introduced in [11] are closed categorical structures for multicategories (extensions of categories, whose maps are allowed to have finitely many arguments) and correspond to planar multiplicative intuitionistic linear logic with only linear implication $\multimap$ and without tensor product nor unit. Closed categories introduced in [5] are something like monoidal closed categories without tensor products, which have internal hom objects defined without using tensor products. In [13], it is shown that closed categories are equivalent to closed multicategories with unit objects. Skew closed categories introduced in [17] are categories with slightly weaker structure than closed categories. There is a categorical structure called skew monoidal categories [18], which have the same components as monoidal categories but the invertibility of unitors and associators are not assumed. Skew closed categories are to skew monoidal categories what closed categories are to monoidal categories.

When we try to obtain these categorical structures by categorical realizability, we face a subtle problem. To exclude symmetries, we have to remove the element $\mathbf{C}$ (corresponding to the lambda term $\lambda xyz.xzy$ expressing the swap of arguments) from an applicative structure. However, to obtain closed structures, we need to realize maps sent by internal hom functors $[f,g] : h \mapsto (g \circ h \circ f)$, and realizing them needs some exchange operation. We resolve this problem by introducing restricted exchanges $(-)^{\bullet}$, $(-)^{\circ}$ and $\mathbf{B'}$.

For an applicative structure $(|\mathcal{A}|, \cdot)$, $(-)^{\bullet}$ is a unary operation on $|\mathcal{A}|$ such that $\mathsf{x}^{\bullet} \cdot \mathsf{y} = \mathsf{y} \cdot \mathsf{x}$ for any x and y in $|\mathcal{A}|$. Since $\mathbf{C} \cdot \mathbf{I} \cdot \mathsf{x}$ satisfies the axiom of $\mathsf{x}^{\bullet}$, assuming $(-)^{\bullet}$ is a weaker assumption than assuming $\mathbf{C}$. $\mathbf{BI}(-)^{\bullet}$-algebras, i.e., $\mathbf{BI}$-algebras with $(-)^{\bullet}$, give rise to (non-symmetric) closed multicategories. Moreover, we show that being a $\mathbf{BI}(-)^{\bullet}$-algebra is a necessary condition to give a category of assemblies as a closed multicategory under some natural assumptions in Proposition 19.

Other than $\mathbf{BI}(-)^{\bullet}$-algebras, we also introduce several classes of applicative structures. The correspondences are in table 1. In particular, we also show that $\mathbf{BII}^{\times}(-)^{\bullet}$-algebras are necessary to give categories of assemblies as closed categories under some conditions. Table 2 is the summary of elements and constructions assumed in these applicative structures.

Table 2 Summary of the elements and constructions of applicative structures.

Elements and constructions	Axiom	Section
element $\mathsf{\mathbf{S}}$	$\mathsf{\mathbf{S}xyz} \simeq \mathsf{xz(yz)}$	2.2
element $\mathsf{\mathbf{K}}$	$\mathsf{\mathbf{K}xy} \simeq \mathsf{x}$	2.2
element $\mathsf{\mathbf{B}}$	$\mathsf{\mathbf{B}xyz} = \mathsf{x(yz)}$	2.3
element $\mathsf{\mathbf{C}}$	$\mathsf{\mathbf{C}xyz} = \mathsf{xzy}$	2.3
element $\mathsf{\mathbf{I}}$	$\mathsf{\mathbf{I}x} = \mathsf{x}$	2.3
element $\mathsf{\mathbf{B}}'$	$\mathsf{\mathbf{B}'xyz} = \mathsf{y(xz)}$	5.3
element $\mathsf{\mathbf{I}}^\times$	$\mathsf{\mathbf{I}^\times x\mathbf{I}} = \mathsf{x}$	4
unary operation $(-)^\bullet$	$\mathsf{x^\bullet y} = \mathsf{yx}$	3
unary operation $(-)^\circ$	$\mathsf{x^\circ yz} = \mathsf{y(xz)}$	5.2

To the best of our knowledge, this is the first systematic treatment of realizability semantics for the non-commutative or planar setting. We hope that our analysis brings new insights on categorical realizability and extends its applications to new areas, most notably to non-commutative logics and systems such as Lambek calculus.

The rest of this paper is structured as follows. In Section 2, we recall some basic notions and results in categorical realizability. In Section 3, we introduce $\mathsf{\mathbf{BI}}(-)^\bullet$-algebras and describe how they correspond to the planar lambda calculus and closed multicategories. Section 4 is about $\mathsf{\mathbf{BII}}^\times(-)^\bullet$-algebras and closed categories, which has a similar story to Section 3. In Section 5, we give three other classes of applicative structures and see categorical structures of assemblies on them. In Section 6, we construct concrete examples of $\mathsf{\mathbf{BI}}(-)^\bullet$-algebras other than the planar lambda calculus. As an unexpected one, we show that the computational lambda calculus [14] is a $\mathsf{\mathbf{BII}}^\times(-)^\bullet$-algebra. In Section 7, we discuss related work. Finally, in Section 8, we summarize contents of this paper and describe future work.

Basic knowledge of category theory and the lambda calculus is assumed.

2 Background

In this section, we recall some basic concepts and results. All the definitions and propositions in this section are from [12] and [8].

2.1 Applicative structures and categories of assemblies

▶ **Definition 1.** *A* partial applicative structure $\mathcal{A}$ *is a pair of a set* $|\mathcal{A}|$ *and a partial binary operation* $(\mathsf{x},\mathsf{y}) \mapsto \mathsf{x}\cdot\mathsf{y}$ *on* $|\mathcal{A}|$*. Application associates to the left, and we often omit* $\cdot$ *and write it as juxtaposition. For instance,* $\mathsf{xz(yz)}$ *denotes* $(\mathsf{x}\cdot\mathsf{z})\cdot(\mathsf{y}\cdot\mathsf{z})$*. When the binary operation of* $\mathcal{A}$ *is total, we say* $\mathcal{A}$ *is a* total applicative structure*.*

In the sequel, we use two notations $\downarrow$ and $\simeq$. The down arrow means "defined." For instance, for a partial applicative structure $(|\mathcal{A}|,\cdot)$, $xy\downarrow$ means that $x\cdot y$ is defined. "$\simeq$" denotes the Kleene equality, which means that if the one side of the equation is defined then the other side is also defined and are equal.

▶ **Definition 2.** *Let* $\mathcal{A}$ *be a partial applicative structure.*

(1) *An* assembly *on* $\mathcal{A}$ *is a pair* $X := (|X|, \|\text{-}\|_X)$*, where* $|X|$ *is a set and* $\|\text{-}\|_X$ *is a function sending* $x \in |X|$ *to a non-empty subset* $\|x\|_X$ *of* $|\mathcal{A}|$*.*

(2) *A* map of assemblies $f : X \to Y$ *is a function* $f : |X| \to |Y|$ *such that there exists an element* $\mathsf{r} \in |\mathcal{A}|$ *realizing* f*, where* "r realizes f" *means that*

$$\forall x \in |X|,\ \forall \mathsf{a} \in \|x\|_X,\ \mathsf{ra}\downarrow \text{ and } \mathsf{ra} \in \|f(x)\|_Y.$$

We say that r *is a* realizer *of* f *when* r *realizes* f*.*

If we assume two extra conditions on a partial applicative structure, we can construct a category from assemblies and maps of assemblies.

▶ **Definition 3.** *Let* $\mathcal{A}$ *be a partial applicative structure satisfying that:*

(i) $|\mathcal{A}|$ *has an element* $\mathbf{I}$ *such that for any* $\mathsf{x} \in |\mathcal{A}|$, $\mathbf{I}\mathsf{x}\downarrow$ *and* $\mathbf{I}\mathsf{x} = \mathsf{x}$*;*

(ii) *for any* $\mathsf{r}_1, \mathsf{r}_2 \in |\mathcal{A}|$*, there exists* $\mathsf{r}_{1,2} \in |\mathcal{A}|$ *such that for any* $\mathsf{x} \in |\mathcal{A}|$, $\mathsf{r}_{1,2}\mathsf{x} \simeq \mathsf{r}_1(\mathsf{r}_2\mathsf{x})$.

Then we construct categories as follows:

(1) *The* category $\mathbf{Asm}(\mathcal{A})$ of assemblies *on* $\mathcal{A}$ *consists of assemblies on* $\mathcal{A}$ *as its objects and maps of assemblies as its maps. Identity maps and composition maps are the same as those of* $\mathbf{Sets}$*.*

(2) *The* category $\mathbf{Mod}(\mathcal{A})$ of modest sets *on* $\mathcal{A}$ *is the full subcategory of* $\mathbf{Asm}(\mathcal{A})$*, such that each object* $(|X|, \|\text{-}\|_X)$ *has the property:*

$$\forall x, y \in |X|,\ x \neq y \Rightarrow \|x\|_X \cap \|y\|_X = \emptyset.$$

$\mathbf{Asm}(\mathcal{A})$ and $\mathbf{Mod}(\mathcal{A})$ are indeed categories. For any assembly $(|X|, \|\text{-}\|)$ on $\mathcal{A}$, the identity function on $|X|$ is realized by $\mathbf{I}$. Given two maps of assemblies $X \xrightarrow{f} Y \xrightarrow{g} Z$ realized by r_2 and r_1 respectively, the composition function $g \circ f : |X| \to |Z|$ is realized by $\mathsf{r}_{1,2}$.

In particular, a total applicative structure $\mathcal{A}$ which has $\mathbf{I}$ and $\mathbf{B}$ (such that $\mathbf{B}\mathsf{xyz} = \mathsf{x}(\mathsf{yz})$ for all $\mathsf{x}, \mathsf{y}, \mathsf{z} \in |\mathcal{A}|$) induces the category of assemblies and the category of modest sets. We call such a total applicative structure a **BI**-*algebra.*

In the following two subsections, we introduce two well-known classes of partial applicative structures. The categories constructed from partial applicative structures in these classes have useful categorical structures.

▶ Remark. In the rest of this paper, we only deal with categories of assemblies and not with categories of modest sets. However, all propositions hold when we replace the word "assemblies" by "modest sets."

2.2 PCAs and Cartesian closed categories

In this subsection, we recall a well-known class of partial applicative structures called partial combinatory algebras (PCAs). Assemblies on a PCA form a Cartesian closed category.

▶ **Definition 4.** *A* PCA *is a partial applicative structure* $\mathcal{A}$ *which contains two elements* $\mathbf{S}$ *and* $\mathbf{K}$ *satisfying:*

(i) $\forall \mathsf{x}, \mathsf{y} \in |\mathcal{A}|$, $\mathbf{K}\mathsf{x}\downarrow$ *and* $\mathbf{K}\mathsf{xy} \simeq \mathsf{x}$*;*

(ii) $\forall \mathsf{x}, \mathsf{y}, \mathsf{z} \in |\mathcal{A}|$, $\mathbf{S}\mathsf{x}\downarrow$, $\mathbf{S}\mathsf{xy}\downarrow$ *and* $\mathbf{S}\mathsf{xyz} \simeq \mathsf{xz}(\mathsf{yz})$.

▶ **Example 5.** Suppose infinite supply of variables $x, y, z, \ldots$. *Untyped lambda terms* are terms constructed from the following six rules:

$$\frac{}{x \vdash x}\ (\text{identity}) \quad ; \quad \frac{\Gamma \vdash M \qquad \Delta \vdash N}{\Gamma, \Delta \vdash MN}\ (\text{application})$$

where Γ and Δ are sequences of distinct variables and contain no common variables;

$$\frac{\Gamma, x \vdash M}{\Gamma \vdash \lambda x.M}\text{(abstraction)}\ ;\quad \frac{\Gamma, x, y \vdash M}{\Gamma, y, x \vdash M}\text{(exchange)}\ ;\quad \frac{\Gamma, x, y \vdash M}{\Gamma, x \vdash M[x/y]}\text{(contraction)}$$

where $M[x/y]$ denotes the term obtained by substituting x for all free y in M;

$$\frac{\Gamma \vdash M}{\Gamma, x \vdash M}\text{(weakening)} \quad \text{where } x \text{ is a variable not contained in } \Gamma.$$

Note that abstraction rules are only applied to the rightmost variables. In order to apply the abstraction rule to a variable in a different position, we need to use exchange rules and move the variable to the rightmost place.

We define an equivalence relation $=_\beta$ on lambda terms as the congruence of the relation $(\lambda x.M)N \sim M[N/x]$. Untyped lambda terms form a PCA. The underlying set consists of β-equivalence classes of untyped closed lambda terms (i.e., lambda terms with no free variables) and the application is defined as that of lambda terms. In this example, $\lambda xyz.xz(yz)$ is the representative of **S** and $\lambda xy.x$ is the representative of **K**.

PCAs are closely related to the untyped lambda calculus through the property called *combinatory completeness*.

▶ **Definition 6.** *Let $\mathcal{A}$ be a partial applicative structure. A* polynominal *over $\mathcal{A}$ is a syntactic expression generated by variables, elements of $|\mathcal{A}|$ and applications. For polynominals M and N over $\mathcal{A}$, $M \simeq N$ means that $M[\mathsf{a}_1/x_1, ..., \mathsf{a}_n/x_n] \simeq N[\mathsf{a}_1/x_1, ..., \mathsf{a}_n/x_n]$ holds in $\mathcal{A}$ for any $\mathsf{a}_1, ..., \mathsf{a}_n \in |\mathcal{A}|$, where $\{x_1, ..., x_n\}$ contains all the variables of M and N.*

▶ **Proposition 7.** *(combinatory completeness of PCAs) Let $\mathcal{A}$ be a PCA and M be a polynominal over $|\mathcal{A}|$. For any variable x, there exists a polynominal M' such that the free variables of M' are the free variables of M excluding x and $M'\mathsf{a} \simeq M[\mathsf{a}/x]$ for all $\mathsf{a} \in |\mathcal{A}|$. We write $\lambda^* x.M$ for such M'.*

Proof. We define $\lambda^* x.M$ by induction on the structure of M as follows: $\lambda^* x.x :=$ **SKK**; $\lambda^* x.y :=$ **K**y (when $x \neq y$); $\lambda^* x.MN :=$ **S**$(\lambda^* x.M)(\lambda^* x.N)$. ◀

For the special case of the above proposition, any closed lambda term is β-equivalent to some term constructed from $\lambda xyz.xz(yz)$ and $\lambda xy.x$ only using applications.

Although conditions of PCAs are simple, categorical structures induced by these algebras are quite strong and useful.

▶ **Proposition 8.** *Let $\mathcal{A}$ be a PCA. Then* **Asm**$(\mathcal{A})$ *is Cartesian closed and regular.*

▶ Remark. Since all the examples in this paper are total applicative structures, we shall in future only consider total structures, although the definitions and results do generalize smoothly to partial ones. A discussion about "partial **BCI**-algebras" is found in Remark 1 of [8]. From now on, whenever we say "applicative structure", it means total applicative structures.

2.3 BCI-algebras and symmetric monoidal closed categories

In this subsection we recall another class of applicative structures called **BCI**-algebra. **BCI**-algebras are related to linear structures whereas PCAs are not.

▶ **Definition 9.** *A* **BCI**-algebra *is an applicative structure $\mathcal{A}$ which contains three elements* **B**, **C** *and* **I** *satisfying:*

(i) $\forall \mathsf{x}, \mathsf{y}, \mathsf{z} \in |\mathcal{A}|$, $\mathbf{B}\mathsf{xyz} = \mathsf{x}(\mathsf{yz})$;

(ii) $\forall \mathsf{x}, \mathsf{y}, \mathsf{z} \in |\mathcal{A}|$, $\mathbf{C}\mathsf{xyz} = \mathsf{xzy}$;

(iii) $\forall \mathsf{x} \in |\mathcal{A}|$, $\mathbf{I}\mathsf{x} = \mathsf{x}$.

▶ **Example 10.** *Untyped linear lambda terms* are untyped lambda terms constructed without using weakening and contraction rules.

Untyped linear lambda terms form a **BCI**-algebra. The underlying set consists of β-equivalence classes of closed linear lambda terms and the application is defined as that of the lambda calculus. Here $\lambda xyz.x(yz)$, $\lambda xyz.xzy$ and $\lambda x.x$ are the representatives of **B**, **C** and **I** respectively.

▶ **Proposition 11.** *(combinatory completeness of* **BCI***-algebras) Let $\mathcal{A}$ be a* **BCI***-algebra and M be a polynominal over $|\mathcal{A}|$ whose variables appear exactly once in M. For any variable x in M, there exists a polynominal $\lambda^* x.M$ such that the free variables of $\lambda^* x.M$ are the free variables of M excluding x and $(\lambda^* x.M)\mathsf{a} = M[\mathsf{a}/x]$ for all $\mathsf{a} \in |\mathcal{A}|$.*

Proof. We define $\lambda^* x.M$ by induction on the structure of M as follows:

- $\lambda^* x.x := \mathbf{I}$
- $\lambda^* x.MN := \begin{cases} \mathbf{C}(\lambda^* x.M)N & (x \in FV(M)) \\ \mathbf{B}M(\lambda^* x.N) & (x \in FV(N)) \end{cases}$ ◀

For the special case of the above proposition, any closed linear lambda term is β-equivalent to some term constructed from $\lambda xyz.x(yz)$, $\lambda xyz.xzy$ and $\lambda x.x$ only using applications.

Since **BCI**-algebras are related to the linear lambda calculus, categorical structures of assemblies on **BCI**-algebras are also linear.

▶ **Proposition 12.** *Let $\mathcal{A}$ be a* **BCI***-algebra. Then* $\mathbf{Asm}(\mathcal{A})$ *is a symmetric monoidal closed category (SMCC).*

3 BI$(-)^\bullet$-algebras and closed multicategories

From here, we investigate realizability without symmetry. To obtain $\mathbf{Asm}(\mathcal{A})$ with non-symmetric categorical structures, $\mathcal{A}$ needs to be a structure in between **BI**-algebras and **BCI**-algebras. In this section, we introduce such a new class of applicative structures **BI**$(-)^\bullet$-algebra, whose assemblies form closed multicategories.

▶ **Definition 13.** *Let $\mathcal{A}$ be an applicative structure. For x in $|\mathcal{A}|$, we write $\mathsf{x}^\bullet$ for an element of $|\mathcal{A}|$ (whenever it exists) such that $\mathsf{x}^\bullet\mathsf{a} = \mathsf{a}\mathsf{x}$ for all $\mathsf{a} \in |\mathcal{A}|$. We say that $\mathcal{A}$ is a* **BI**$(-)^\bullet$-algebra *iff it contains* **B**, **I** *and $\mathsf{x}^\bullet$ for each $\mathsf{x} \in |\mathcal{A}|$.*

A PCA is related to the untyped lambda calculus, which has all term construction rules. A **BCI**-algebra is related to the untyped linear lambda calculus, which does not have weakening nor contraction. Similarly, A **BI**$(-)^\bullet$-algebra is related to the untyped planar lambda calculus, which has none of weakening, contraction nor exchange rules.

▶ **Example 14.** *Untyped planar lambda terms* are untyped lambda terms constructed without using weakening, contraction or exchange rules[1]. Untyped closed planar lambda terms form a **BI**$(-)^\bullet$-algebra. The underlying set consists of β-equivalence classes of closed planar lambda

[1] The definition of construction rules of planar lambda terms has two different styles. In our definition, the abstraction rule is only allowed for the rightmost variable. Such a term construction is seen in [1]. On the other hand, there is also a definition that the abstraction rule is only allowed for the leftmost variable, as in [20]. Here we choose the former style for preservation the planarity of terms under the $\beta\eta$-conversions.

terms and the application is defined as that of lambda terms. Here $\lambda xyz.x(yz)$ and $\lambda x.x$ are the representatives of B and I. Let M be a representative of x. Then $\lambda x.xM$ is also a closed planar term and is the representative of $\mathsf{x}^{\bullet}$. We write $\mathcal{L}_{planar}$ for this applicative structure. $\mathcal{L}_{planar}$ does not have a term satisfying the axiom of C. That is intuitively obvious, however, the rigorous proof is a little bit tricky and omitted.

▶ **Proposition 15.** *(combinatory completeness of $\mathsf{BI}(-)^{\bullet}$-algebras) Let $\mathcal{A}$ be a $\mathsf{BI}(-)^{\bullet}$-algebra and M be a polynominal over $|\mathcal{A}|$ whose variables appear exactly once in M. For the rightmost variable x of M, there exists $\lambda^{*}x.M$ such that the free variables of $\lambda^{*}x.M$ are the free variables of M in the same order excluding x and $(\lambda^{*}x.M)\mathsf{a} = M[\mathsf{a}/x]$ for all $\mathsf{a} \in |\mathcal{A}|$.*

Proof. We define $\lambda^{*}x.M$ by induction on the structure of M as follows:

- $\lambda^{*}x.x := \mathsf{I}$
- $\lambda^{*}x.MN := \begin{cases} \mathsf{B}N^{\bullet}(\lambda^{*}x.M) & (x \in FV(M)) \\ \mathsf{B}M(\lambda^{*}x.N) & (x \in FV(N)) \end{cases}$

Note that for $\lambda^{*}x.MN$, x is the rightmost free variable in MN. Therefore, if x is in $FV(M)$, N has no free variables and $N^{\bullet}$ can be defined. ◀

For the special case of the above proposition, any closed planar lambda term is β-equivalent to some term constructed from $\lambda xyz.x(yz)$ and $\lambda x.x$ using applications and the unary operation $(-)^{\bullet} : M \mapsto \lambda x.xM$.

Since $\mathsf{CI}\mathsf{x}$ satisfies the axiom of $\mathsf{x}^{\bullet}$, any BCI-algebra is also a $\mathsf{BI}(-)^{\bullet}$-algebra. $\mathsf{BI}(-)^{\bullet}$-algebras are weaker than BCI-algebra, and thus categories of assemblies on $\mathsf{BI}(-)^{\bullet}$-algebras have weaker categorical structures than those on BCI-algebras. We show that assemblies on a $\mathsf{BI}(-)^{\bullet}$-algebra form a closed multicategory, which is more general than symmetric monoidal closed categories.

Multicategories are extensions of categories, whose maps are allowed to have finitely many arguments. Closed multicategories are closed categorical structures for multicategories and correspond to planar multiplicative intuitionistic linear logic with only linear implication $\multimap$ and without tensor product nor unit. Here the word "planar" means the planarity of the string diagrams of the modeling categories. The string diagrams do not contain symmetries or braids.

First, we recall the definition of closed multicategories in [13].

▶ **Definition 16.** *A* multicategory $\mathbf{C}$ *consists of the following data:*

1. *a collection $Ob(\mathbf{C})$;*
2. *for each $n \geq 0$ and $X_1, X_2, \ldots, X_n, Y \in Ob(\mathbf{C})$, a set $\mathbf{C}(X_1, \ldots, X_n; Y)$;*
3. *for each $X \in Ob(\mathbf{C})$, an element $1_X \in \mathbf{C}(X; X)$, called the* identity map*;*
4. *for each $n, k_1, k_2, \ldots, k_n \in \mathbb{N}$ and X_{ij}, Y_i, Z $(i \leq n, j \leq k_i)$, a function*

$$\circ : \prod_{i}^{n} \mathbf{C}(X_{i1}, \ldots, X_{ik_i}; Y_i) \times \mathbf{C}(Y_1, \ldots, Y_n; Z) \to \mathbf{C}(X_{11}, \ldots, X_{1k_1}, X_{21}, \ldots, X_{nk_n}; Z)$$

called the composition*. $g \circ (f_1, \ldots, f_n)$ denotes the composition of $g \in \mathbf{C}(Y_1, \ldots, Y_n; Z)$ and $f_i \in \mathbf{C}(X_{i1}, \ldots, X_{ik_i}; Y_i)$. The compositions satisfy associativity and identity axioms.*

▶ **Definition 17.** *A* closed multicategory *consists of the following data:*

1. *a multicategory $\mathbf{C}$;*
2. *for each $X_1, X_2, \ldots, X_n, Y \in Ob(\mathbf{C})$, an object $\underline{\mathbf{C}}(X_1, X_2, \ldots, X_n; Y)$, called the* internal hom object*;*

3. *for each* $X_1, \dots, X_n, Y \in Ob(\mathbf{C})$*, a map*

$$ev_{X_1,\dots,X_n;Y} : X_1, \dots, X_n, \underline{\mathbf{C}}(X_1, \dots, X_n; Y) \to Y,$$

called the evaluation map *such that* $\forall Z_1, Z_2, \dots, Z_m \in Ob(\mathbf{C})$*, the function* $\varphi_{X_1,\dots,X_n;Z_1,\dots,Z_m;Y} : \mathbf{C}(Z_1, \dots, Z_m; \underline{\mathbf{C}}(X_1, \dots, X_n; Y)) \to \mathbf{C}(X_1, \dots, X_n, Z_1, \dots, Z_m; Y)$ *sending* f *to* $ev_{X_1,\dots,X_n;Y} \circ (1_{X_1}, \dots, 1_{X_n}, f)$ *is invertible. We write the inverse function* $\Lambda_{X_1,\dots,X_n;Z_1,\dots,Z_m;Y}$.

▶ **Proposition 18.** *Let* $\mathcal{A}$ *be a* $\mathsf{BI}(-)^\bullet$*-algebra. Then* $\mathbf{Asm}(\mathcal{A})$ *is a closed multicategory.*

Proof. Let $\mathbf{C} := \mathbf{Asm}(\mathcal{A})$. It is obvious that $\mathbf{C}$ is a category. We define a bifunctor $[-,-] : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{C}$ as follows:

- $[-,-]$ sends $(|X|, \|\text{-}\|_X)$ and $(|Y|, \|\text{-}\|_Y)$ to $(|[X,Y]|, \|\text{-}\|_{[X,Y]})$, where $|[X,Y]|$ is the set of maps from $(|X|, \|\text{-}\|_X)$ to $(|Y|, \|\text{-}\|_Y)$ in $\mathbf{C}$ and $\|f\|_{[X,Y]} := \{\mathsf{r} \mid \mathsf{r} \text{ realizes } f\}$.
- $[-,-]$ sends $f : X' \to X$ and $g : Y \to Y'$ in $\mathbf{C}$ to the function $[f,g] : [X,Y] \to [X',Y']$ which sends $h : X \to Y$ to $g \circ h \circ f$.

We check that $[-,-]$ certainly forms a functor. Let r_f and r_g be the realizers for f and g, then $\mathsf{B}(\mathsf{B}\mathsf{r}_f{}^\bullet\mathsf{B})(\mathsf{B}\mathsf{r}_g)$ is the realizer for $[f,g]$. Therefore, for any $f : X' \to X$ and $g : Y \to Y'$ in $\mathbf{C}$, $[f,g]$ exists in $\mathbf{C}$. It is easy to see that $[-,-]$ preserves identities and compositions.

Next, we show that $\mathbf{C}$ is a multicategory. For $X_1, \dots, X_n, Y \in \mathbf{C}$ $(n > 0)$, the set $\mathbf{C}(X_1, \dots, X_n; Y)$ is defined as the underlying set of the object[2] $[X_n, [X_{n-1}, [\dots [X_1, Y] \dots]]]$. In the case $n = 0$, $\mathbf{C}(;Y)$ is defined as the underlying set $|Y|$. Identity maps and compositions are usual ones in $\mathbf{Sets}$.

To check that composition maps have realizers, we use Proposition 15. Given $g \in \mathbf{C}(Y_1, \dots, Y_m; Z)$ and $f_l \in \mathbf{C}(X^l_1, \dots, X^l_{k_l}; Y_l)$ $(1 \le l \le m)$, whose realizers are $\mathsf{u}, \mathsf{v}_1, \dots, \mathsf{v}_m$ respectively. Then the composition map $g \circ (f_1, \dots, f_m)$ is realized by r such that for any $\mathsf{a}^1_1, \mathsf{a}^1_2, \dots, \mathsf{a}^1_{k_1}, \mathsf{a}^2_1, \dots, \mathsf{a}^m_{k_m}$ in $|\mathcal{A}|$,

$$\mathsf{r}\mathsf{a}^m_{k_m} \dots \mathsf{a}^m_1 \dots \mathsf{a}^1_{k_1} \dots \mathsf{a}^1_1 = \mathsf{u}(\mathsf{v}_m \mathsf{a}^m_{k_m} \dots \mathsf{a}^m_1) \dots (\mathsf{v}_1 \mathsf{a}^1_{k_1} \dots \mathsf{a}^1_1).$$

The existence of such r follows by the combinatory completeness of $\mathsf{BI}(-)^\bullet$-algebras.

Finally, we show that $\mathbf{C}$ is a closed multicategory. We take internal hom objects $\underline{\mathbf{C}}(X_1, \dots, X_n; Y)$ as $[X_n, [X_{n-1}, [\dots [X_1, Y] \dots]]]$ and evaluation maps as the obvious ones, where the evaluation maps are realized by I. φ is invertible as a function and for a map $g : X_1, \dots, X_n, Z_1, \dots, Z_m \to Y$, $\Lambda(g)$ is indeed realized by a realizer of g. ◀

The converse of Proposition 18 holds under some natural conditions.

▶ **Proposition 19.** *Suppose* $\mathcal{A}$ *is an applicative structure and* $\mathbf{C} := \mathbf{Asm}(\mathcal{A})$ *happens to be a closed multicategory of assemblies.* $\mathcal{A}$ *is a* $\mathsf{BI}(-)^\bullet$*-algebra if the following conditions hold:*

(i) $\underline{\mathbf{C}}(;X) = X$ *and* $\mathbf{C}(;X)$ *is the underlying set* $|X|$*;*
(ii) $f \in \mathbf{C}(X;Y)$ *iff* f *is a function from* $|X|$ *to* $|Y|$ *realized by some element of* $|\mathcal{A}|$*;*
(iii) *identity maps are obvious ones;*
(iv) $\underline{\mathbf{C}}(X;Y) = (\mathbf{C}(X;Y), \|\text{-}\|)$ *where* $\|f\| = \{\mathsf{r} \mid \mathsf{r} \text{ realizes } f\}$*;*

[2] Here we reverse the order of arguments due to the difference between closed multicategories and applicative structures. Internal hom objects of a closed multicategory receive arguments from the left side, whereas in an applicative structure, elements receive arguments from the right side. If we employ another definition of closed multicategories with reversing the order of arguments of the compositions and evaluation maps, then it will be suitable for the order of realizers.

(v) $\underline{\mathbf{C}}(X_1, \dots, X_{n+1}; Y) = \underline{\mathbf{C}}(X_{n+1}; \underline{\mathbf{C}}(X_1, \dots, X_n; Y))$ *and* $\mathbf{C}(X_1, \dots, X_{n+1}; Y)$ *is the underlying set of* $\underline{\mathbf{C}}(X_1, \dots, X_{n+1}; Y)$*;*

(vi) *for* $g : Y_1, \dots, Y_n \to Z$ *and* $f_l : X^l_1, \dots, X^l_{k_l} \to Y_l$*,* $g \circ (f_1, \dots, f_n)$ *sends* $x^1_1, \dots, x^1_{k_1}, \dots, x^n_{k_n}$ *to* $g(f_n(x^n_{k_n}, \dots, x^n_1), \dots, f_1(x^1_{k_1}, \dots, x^1_1))$*;*

(vii) $ev_{X_1,\dots,X_n;Y}$ *sends* $x_1, \dots, x_n, f$ *to* $f(x_n, \dots, x_1)$*;*

(viii) $\Lambda_{X_1,\dots,X_n;Z_1,\dots,Z_m;Y}$ *sends a function* $(x_1, \dots, x_n, z_1, \dots, z_m \mapsto f(z_m, \dots, z_1, x_n, \dots, x_1))$ *to a function* $(z_1, \dots, z_m \mapsto f(z_m, \dots, z_1, -, \dots, -))$.

Proof. Let $X := (|\mathcal{A}|, \|\text{-}\|_X)$, where $\|\mathsf{a}\|_X := \{\mathsf{a}\}$. Suppose $\boldsymbol{\mathsf{I}}_0$ is a realizer of 1_X. Then $\boldsymbol{\mathsf{I}}_0\mathsf{a} \in \{\mathsf{a}\}$ for any $\mathsf{a} \in |\mathcal{A}|$. Therefore $\boldsymbol{\mathsf{I}}_0$ has the property of $\boldsymbol{\mathsf{I}}$.

Let $Y := (|\mathcal{A}| \times |\mathcal{A}|, \|\text{-}\|_Y)$, where $\|(\mathsf{a}, \mathsf{a}')\|_Y := \{\mathsf{aa}'\}$. Given arbitrary two element $\mathsf{r}, \mathsf{r}' \in |\mathcal{A}|$, define $f : X \to Y$ as $\mathsf{a} \mapsto (\mathsf{r}, \mathsf{a})$ and $g : Y \to Y$ as $(\mathsf{a}, \mathsf{a}') \mapsto (\mathsf{r}', \mathsf{aa}')$. Let $L^X_{Y,Y} : \underline{\mathbf{C}}(Y; Y) \to \underline{\mathbf{C}}(\underline{\mathbf{C}}(X; Y); \underline{\mathbf{C}}(X; Y))$ be the map

$$\Lambda_{\underline{\mathbf{C}}(X;Y);\underline{\mathbf{C}}(Y;Y);\underline{\mathbf{C}}(X;Y)}(\Lambda_{X;\underline{\mathbf{C}}(X;Y),\underline{\mathbf{C}}(Y;Y);Y}(ev_{Y;Y} \circ (ev_{X;Y}, 1_{\underline{\mathbf{C}}(Y;Y)}))).$$

$L^X_{Y,Y}$ sends g to $(f \mapsto g \circ f)$. Suppose $\boldsymbol{\mathsf{B}}_0$ is a realizer of $L^X_{Y,Y}$. Then $\boldsymbol{\mathsf{B}}_0\mathsf{r}'\mathsf{r}$ realizes $g \circ f$ and thus $\boldsymbol{\mathsf{B}}_0\mathsf{r}'\mathsf{ra} \in \|g(f(\mathsf{a}))\|_Y = \{\mathsf{r}'(\mathsf{ra})\}$ for any $\mathsf{a} \in |\mathcal{A}|$. Therefore $\boldsymbol{\mathsf{B}}_0$ has the property of $\boldsymbol{\mathsf{B}}$.

Given arbitrary $\mathsf{x} \in |\mathcal{A}|$, define $Ev_\mathsf{x} : \underline{\mathbf{C}}(X; X) \to X$ as $ev_{X;X} \circ (\mathsf{x}, 1_{[X,X]})$. For any $\mathsf{a} \in |\mathcal{A}|$, Ev_x sends $f_\mathsf{a} : X \to X$, $\mathsf{a}' \mapsto \mathsf{aa}'$ to $f_\mathsf{a}(\mathsf{x})$. Suppose $(\mathsf{x}^\bullet)_0$ is a realizer of Ev_x. Then $(\mathsf{x}^\bullet)_0\mathsf{a} \in \{\mathsf{ax}\}$. Therefore $(\mathsf{x}^\bullet)_0$ has the property of $\mathsf{x}^\bullet$. ◀

4 $\mathbf{BII}^\times(-)^\bullet$-algebras and closed categories

In the previous section, we saw that $\mathbf{BI}(-)^\bullet$-algebras correspond to closed multicategories. In this section, we show that a slightly stronger class of applicative structures ($\mathbf{BII}^\times(-)^\bullet$-algebras) corresponds to a slightly stronger categorical structure (closed categories).

Closed categories are something like monoidal closed categories without tensor products, which have internal hom objects defined without using tensor products. It is shown in [13] that closed categories are slightly stronger categorical structures than closed multicategories.

First, we recall the definition of closed categories in [13].

▶ **Definition 20.** *A* closed category *consists of the following data:*

1. *a locally small category* $\mathbf{C}$*;*
2. *a functor* $[-,-] : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{C}$*, called the* internal hom functor*;*
3. *an object* I*, called the* unit object*;*
4. *a natural isomorphism* $i_X : [I, X] \to X$*;*
5. *an extranatural transformation* $j_X : I \to [X, X]$*;*
6. *a transformation* $L^X_{Y,Z} : [Y, Z] \to [[X, Y], [X, Z]]$ *natural in* Y, Z *and extranatural in* X*,*

such that the following axioms hold:

(i) $\forall X, Y \in \mathbf{C}$, $L^X_{Y,Y} \circ j_Y = j_{[X,Y]}$*;*

(ii) $\forall X, Y \in \mathbf{C}$, $i_{[X,Y]} \circ [j_X, 1_{[X,Y]}] \circ L^X_{X,Y} = 1_{[X,Y]}$*;*

(iii) $\forall X, Y, Z, W \in \mathbf{C}$*, the following diagram commutes:*

$$\begin{tikzcd}
{[Z,W]} \arrow[r, "L^Y_{Z,W}"] \arrow[d, "L^X_{Z,W}"'] & {[[Y,Z],[Y,W]]} \arrow[dd, "{[1_{[Y,Z]}, L^X_{Y,W}]}"] \\
{[[X,Z],[X,W]]} \arrow[d, "L^{[X,Y]}_{[X,Z],[X,W]}"'] & \\
{[[[X,Y],[X,Z]],[[X,Y],[X,W]]]} \arrow[r, "{[L^X_{Y,Z}, 1_{[[X,Y],[X,W]]}]}"'] & {[[Y,Z],[[X,Y],[X,W]]]}
\end{tikzcd}$$

(iv) *$\forall X, Y \in \mathbf{C}$, $L^I_{X,Y} \circ [1_X, i_Y] = [i_X, 1_{[I,Y]}]$;*
(v) *$\forall X, Y \in \mathbf{C}$, the function $\gamma : \mathbf{C}(X,Y) \to \mathbf{C}(I, [X,Y])$ which sends $f : X \to Y$ to $[1_X, f] \circ j_X$ is invertible.*

When $\mathcal{A}$ is a $\mathsf{BI}(-)^\bullet$-algebra, the closed multicategory structure of $\mathbf{Asm}(\mathcal{A})$ does not generally extend to a closed category since the natural isomorphism i_X^{-1} is not generally realized. Next, we give the definition of $\mathsf{BII}^\times(-)^\bullet$-algebras, which we assume an extra element $\mathsf{I}^\times$ for the realizer of i_X^{-1}.

► **Definition 21.** *Let $\mathcal{A}$ be a $\mathsf{BI}(-)^\bullet$-algebra. $\mathsf{I}^\times$ is defined as an element of $|\mathcal{A}|$ (whenever it exists) satisfying $\mathsf{I}^\times \mathsf{a} \mathsf{I} = \mathsf{a}$ for all $\mathsf{a} \in |\mathcal{A}|$. If $\mathcal{A}$ has $\mathsf{I}^\times$, we say that $\mathcal{A}$ is a $\mathsf{BII}^\times(-)^\bullet$-*algebra.

► **Example 22.** Any BCI-algebra is a $\mathsf{BII}^\times(-)^\bullet$-algebra, since CI satisfies the axiom of $\mathsf{I}^\times$.

► **Example 23.** $\mathcal{L}_{planar}$ is a $\mathsf{BII}^\times(-)^\bullet$-algebra. For any closed planar term M, M has a β-normal form since it is a linear lambda term. Let $\lambda u.M'$ be the β-normal form of M. Then $(\lambda xyz.x(yz))M(\lambda v.v) =_\beta \lambda z.Mz =_\beta \lambda z.M'[z/u] =_\alpha \lambda u.M' =_\beta M$. Therefore in this case, B satisfies the axiom of $\mathsf{I}^\times$.

► Remark. There exists a $\mathsf{BI}(-)^\bullet$-algebra which is not a $\mathsf{BII}^\times(-)^\bullet$-algebra. Add the constant rule:

$$\frac{}{\vdash c}\ \text{(constant)}$$

to the construction of planar lambda terms and add no evaluation rules on these constants. We write $\mathcal{L}^c_{planar}$ for the applicative structure which consists of closed planar lambda terms with constants. $\mathcal{L}^c_{planar}$ is a $\mathsf{BI}(-)^\bullet$-algebra and does not contains $\mathsf{I}^\times$.

Note that if we further assume the extensionality (η-equality) on $\mathcal{L}^c_{planar}$, then $\lambda xyz.x(yz)$ satisfies the axiom of $\mathsf{I}^\times$ and the applicative structure forms a $\mathsf{BII}^\times(-)^\bullet$-algebra.

► **Proposition 24.** *Let $\mathcal{A}$ be a $\mathsf{BII}^\times(-)^\bullet$-algebra. Then $\mathbf{Asm}(\mathcal{A})$ is a closed category.*

Proof. Since a $\mathsf{BII}^\times(-)^\bullet$-algebra is also a $\mathsf{BI}(-)^\bullet$-algebra, a bifunctor $[-,-]$ can be defined in the same way as in the proof of Proposition 18.

We define the unit object I as $(\{*\}, \|\text{-}\|_I)$, where $\|*\|_I := \{\mathsf{I}\}$. j_X is defined as the function $* \mapsto 1_X$, which is realized by I. i_X is defined as the function sending $(f : * \mapsto x)$ to x, which is realized by $\mathsf{I}^\bullet$. The inverse function of i_X is realized by $\mathsf{I}^\times$. $L^X_{Y,Z}$ is defined as $g \mapsto (f \mapsto g \circ f)$, which is realized by B. It is easy to verify that i,j and L have naturality and satisfy the axioms of closed category.

Finally, we show that γ is invertible. Let $g \in \mathbf{C}(I, [X,Y])$ then $g(*)$ is $\gamma^{-1}(g)$, which is realized by $\mathsf{r}_g\mathsf{I}$, where r_g is a realizer of g. ◄

Like Proposition 19, the converse of the above proposition holds under some conditions.

► **Proposition 25.** *Suppose $\mathcal{A}$ is an applicative structure and $\mathbf{C} := \mathbf{Asm}(\mathcal{A})$ happens to be a closed category. If the following conditions hold, then $\mathcal{A}$ is a $\mathsf{BII}^\times(-)^\bullet$-algebra.*
(i) *$[X,Y] = (\mathbf{C}(X,Y), \|\text{-}\|)$ where $\|f\| = \{\mathsf{r} \mid \mathsf{r}\ \text{realizes}\ f\}$;*
(ii) *$[f,g] : [X,Y] \to [X',Y']$ is a function which sends $h : X \to Y$ to $g \circ h \circ f$;*
(iii) *$L^X_{Y,Z}$ sends $g : Y \to Z$ to the function $(f : X \to Y) \mapsto (g \circ f : X \to Z)$;*
(iv) *the underlying set of the unit object I is the singleton $\{*\}$;*
(v) *i_X sends a function $(f : * \mapsto x)$ to x.*

Proof. Let $X := (|\mathcal{A}|, \|\text{-}\|_X)$, where $\|\mathsf{a}\|_X := \{\mathsf{a}\}$. Suppose $\mathbf{I}_0$ is the realizer of 1_X. Then $\mathbf{I}_0\mathsf{a} \in \{\mathsf{a}\}$ for any $\mathsf{a} \in |\mathcal{A}|$. Therefore $\mathbf{I}_0$ has the property of $\mathbf{I}$.

Let $Y := (|\mathcal{A}| \times |\mathcal{A}|, \|\text{-}\|_Y)$, where $\|(\mathsf{a}, \mathsf{a}')\|_Y := \{\mathsf{aa}'\}$. Given arbitrary two element $\mathsf{r}, \mathsf{r}' \in |\mathcal{A}|$, define $f : X \to Y$ as $\mathsf{a} \mapsto (\mathsf{r}, \mathsf{a})$ and $g : Y \to Y$ as $(\mathsf{a}, \mathsf{a}') \mapsto (\mathsf{r}', \mathsf{aa}')$. $L^X_{Y,Y}$ sends g to $(f \mapsto g \circ f)$. Suppose $\mathbf{B}_0$ is the realizer of $L^X_{Y,Y}$. Then $\mathbf{B}_0\mathsf{r}'\mathsf{r}$ realizes $g \circ f$ and thus $\mathbf{B}_0\mathsf{r}'\mathsf{ra} \in \|g(f(\mathsf{a}))\|_Y = \{\mathsf{r}'(\mathsf{ra})\}$ for any $\mathsf{a} \in |\mathcal{A}|$. Therefore $\mathbf{B}_0$ has the property of $\mathbf{B}$.

Since $I \cong [I, I]$ and $\mathbf{I} \in \|1_I\|_{[I,I]}$, we can assume $\mathbf{I} \in \|*\|_I$ with loss of generality. Suppose $\mathbf{I}^\times_0$ is the realizer of i^{-1}_X. Then $\mathbf{I}^\times_0\mathsf{a}$ realizes the map $* \mapsto \mathsf{a}$ for any $\mathsf{a} \in |\mathcal{A}|$. Thus $(\mathbf{I}^\times_0\mathsf{a})\mathbf{I} \in \{\mathsf{a}\}$ and $\mathbf{I}^\times_0$ has the property of $\mathbf{I}^\times$.

Given arbitrary $\mathsf{x} \in |\mathcal{A}|$, define $f : I \to X$ as $* \mapsto \mathsf{x}$. For any $\mathsf{a} \in |\mathcal{A}|$, $i_X \circ [f, 1_X]$ sends $f_\mathsf{a} : X \to X$, $\mathsf{a}' \mapsto \mathsf{aa}'$ to $f_\mathsf{a}(\mathsf{x})$. Suppose $(\mathsf{x}^\bullet)_0$ is the realizer of $i_X \circ [f, 1_X]$. Then $(\mathsf{x}^\bullet)_0\mathsf{a} \in \{\mathsf{ax}\}$. Therefore $(\mathsf{x}^\bullet)_0$ has the property of $\mathsf{x}^\bullet$. ◀

5 Other cases

In this section, we introduce three classes of applicative structures which are sufficient for inducing some categorical structures on assemblies. Unlike $\mathbf{BI}(-)^\bullet$-algebras for closed multicategories and $\mathbf{BII}^\times(-)^\bullet$-algebras for closed categories, the classes in this section do not provide necessary conditions for inducing such structures.

5.1 $\mathbf{BK}(-)^\bullet$-algebras and closed categories

▶ **Definition 26.** *A* $\mathbf{BK}(-)^\bullet$-algebra *is an applicative structure $\mathcal{A}$ which contains* $\mathbf{B}$, $\mathbf{K}$ *and* $\mathsf{x}^\bullet$ *for each* $\mathsf{x} \in |\mathcal{A}|$.

▶ **Example 27.** Consider untyped lambda terms constructed without using contraction or exchange rules. Then β-equivalence classes of these closed terms form a $\mathbf{BK}(-)^\bullet$-algebra.

▶ **Proposition 28.** *(combinatory completeness of* $\mathbf{BK}(-)^\bullet$*-algebras) Let $\mathcal{A}$ be a* $\mathbf{BK}(-)^\bullet$*-algebra and M be a polynominal over $|\mathcal{A}|$ whose variables appear at most once in M. For a variable x which is the rightmost variable of M or not in M, there exists a polynominal $\lambda^*x.M$ such that the free variables of $\lambda^*x.M$ are the free variables of M excluding x and $(\lambda^*x.M)\mathsf{a} = M[\mathsf{a}/x]$ for all $\mathsf{a} \in |\mathcal{A}|$.*

Proof. We define $\lambda x.M$ by induction on the structure of M as follows:

- $\lambda^*x.x := \mathbf{BB}^\bullet\mathbf{K}$
- $\lambda^*x.y := \mathbf{K}y \quad (x \neq y)$
- $\lambda^*x.MN := \begin{cases} \mathbf{B}N^\bullet(\lambda^*x.M) & (x \in FV(M)) \\ \mathbf{B}M(\lambda^*x.N) & (x \in FV(N)) \\ \mathbf{K}(MN) & (otherwise) \end{cases}$

◀

Since $\mathbf{BB}^\bullet\mathbf{K}$ satisfies the axiom of $\mathbf{I}$ and $\mathbf{K}$ satisfies the axiom of $\mathbf{I}^\times$, any $\mathbf{BK}(-)^\bullet$-algebra is also a $\mathbf{BII}^\times(-)^\bullet$-algebra. Therefore the next corollary follows by Proposition 24.

▶ **Corollary 29.** *Let $\mathcal{A}$ be a* $\mathbf{BK}(-)^\bullet$*-algebra. Then* $\mathbf{Asm}(\mathcal{A})$ *is a closed category.*

5.2 $\mathsf{BII}^\bullet(-)^\circ$-algebras and skew closed categories

▶ **Definition 30.** *Let $\mathcal{A}$ be an applicative structure. For x in $|\mathcal{A}|$, we write x° as an element of $|\mathcal{A}|$ (whenever it exists) such that $\mathsf{x}^\circ\mathsf{a}\mathsf{a}' = \mathsf{a}(\mathsf{x}\mathsf{a}')$ for all $\mathsf{a}, \mathsf{a}' \in |\mathcal{A}|$. We say that $\mathcal{A}$ is a* $\mathsf{BII}^\bullet(-)^\circ$-algebra *iff it contains B, I, $\mathsf{I}^\bullet$ and x° for each $\mathsf{x} \in |\mathcal{A}|$.*

Since $\mathsf{B}\mathsf{x}^\bullet\mathsf{B}$ satisfies the axiom of x°, any $\mathsf{BI}(-)^\bullet$-algebra is also a $\mathsf{BII}^\bullet(-)^\circ$-algebra. Assemblies on $\mathsf{BII}^\bullet(-)^\circ$-algebras form skew closed categories, which are weaker closed categorical structure than closed categories.

There is a categorical structure called skew monoidal categories [18], which have the same components as monoidal categories but the invertibility of unitors and associators are not assumed. Skew closed categories are to skew monoidal categories what closed categories are to monoidal categories.

First we recall the definition of skew closed categories in [17].

▶ **Definition 31.** *A* (left) skew closed category $\mathbf{C}$ *consists of the following data:*

1. *a locally small category $\mathbf{C}$;*
2. *a functor $[-,-] : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{C}$, called the* internal hom functor*;*
3. *an object I, called the* unit object*;*
4. *a natural transformation $i_X : [I, X] \to X$;*
5. *an extranatural transformation $j_X : I \to [X, X]$;*
6. *a transformation $L^X_{Y,Z} : [Y, Z] \to [[X, Y], [X, Z]]$ natural in Y, Z and extranatural in X,*

such that the following axioms hold:

(i) $\forall X, Y \in \mathbf{C}$, $L^X_{Y,Y} \circ j_Y = j_{[X,Y]}$;

(ii) $\forall X, Y \in \mathbf{C}$, $i_{[X,Y]} \circ [j_X, 1_{[X,Y]}] \circ L^X_{X,Y} = 1_{[X,Y]}$;

(iii) $\forall X, Y, Z, W \in \mathbf{C}$, *the following diagram commutes:*

$$\begin{tikzcd}
{[Z,W]} \arrow[rr, "L^Y_{Z,W}"] \arrow[d, "L^X_{Z,W}"'] & & {[[Y,Z],[Y,W]]} \arrow[dd, "{[1_{[Y,Z]}, L^X_{Y,W}]}"] \\
{[[X,Z],[X,W]]} \arrow[d, "L^{[X,Y]}_{[X,Z],[X,W]}"'] & & \\
{[[[X,Y],[X,Z]],[[X,Y],[X,W]]]} \arrow[rr, "{[L^X_{Y,Z}, 1_{[[X,Y],[X,W]]}]}"'] & & {[[Y,Z],[[X,Y],[X,W]]]}
\end{tikzcd}$$

(iv) $\forall X, Y \in \mathbf{C}$, $[1_{[I,X]}, i_Y] \circ L^I_{X,Y} = [i_X, 1_Y]$;

(v) $i_I \circ j_I = 1_I$.

A left skew closed category is called left normal *when the function $\gamma : \mathbf{C}(X, Y) \to \mathbf{C}(I, [X, Y])$, $f \mapsto [1, f] \circ j_X$ is invertible for any $X, Y \in \mathbf{C}$.*

▶ **Proposition 32.** *Let $\mathcal{A}$ be a $\mathsf{BII}^\bullet(-)^\circ$-algebra. Then $\mathbf{Asm}(\mathcal{A})$ is a left normal skew closed category.*

Proof. We define the functor $[-,-]$ as in the proof of Proposition 18, where $[f, g]$ is realized by $\mathsf{B}\mathsf{r}_f{}^\circ(\mathsf{B}\mathsf{r}_g)$. The rest of the proof is the same as Proposition 24 except for the existence of i_X^{-1}. ◀

Since any $\mathsf{BI}(-)^\bullet$-algebra is also a $\mathsf{BII}^\bullet(-)^\circ$-algebra, the next corollary follows.

▶ **Corollary 33.** *Let $\mathcal{A}$ be a $\mathsf{BI}(-)^\bullet$-algebra. Then $\mathbf{Asm}(\mathcal{A})$ is a left normal skew closed category.*

5.3 $\mathbf{BB'II^\bullet}$-algebras and skew closed categories

Unlike PCAs and **BCI**-algebras, $\mathbf{BI}(-)^\bullet$-algebras, $\mathbf{BK}(-)^\bullet$-algebras and $\mathbf{BII^\bullet}(-)^\circ$-algebras need infinitely many assumptions due to $(-)^\bullet$ or $(-)^\circ$. In this subsection, we introduce a class of applicative structure $\mathbf{BB'II^\bullet}$-algebras, which induces skew closed categories and needs only four assumptions.

▶ **Definition 34.** *A* $\mathbf{BB'II^\bullet}$-algebra *is an applicative structure $\mathcal{A}$ which contains four elements* $\mathbf{B}$, $\mathbf{B'}$, $\mathbf{I}$ *and* $\mathbf{I^\bullet}$*, where* $\mathbf{B'}$ *is a element such that* $\mathbf{B'}\mathsf{xyz} = \mathsf{y}(\mathsf{xz})$ *for all* $\mathsf{x}, \mathsf{y}, \mathsf{z} \in |\mathcal{A}|$.

Since $\mathbf{B'}\mathsf{x}$ satisfies the axiom of x°, any $\mathbf{BB'II^\bullet}$-algebra is also a $\mathbf{BII^\bullet}(-)^\circ$-algebra. Therefore the next corollary follows by Proposition 32.

▶ **Corollary 35.** *Let $\mathcal{A}$ be a* $\mathbf{BB'II^\bullet}$*-algebra. Then* $\mathbf{Asm}(\mathcal{A})$ *is a left normal skew closed category.*

▶ Remark. A category of assemblies on a $\mathbf{BB'II^\bullet}$-algebra is not a closed category in general because i_X^{-1} is not realized. If a $\mathbf{BB'II^\bullet}$-algebra has $\mathbf{I^\times}$, then $\mathbf{B}(\mathbf{B'}(\mathbf{B}(\mathbf{BI^\bullet})(\mathbf{BB'I^\times})))\mathbf{B'}$ satisfies the axiom of $\mathbf{C}$, and thus this $\mathbf{BB'II^\bullet}$-algebra becomes a **BCI**-algebra.

6 Examples

In this section, we introduce three examples of $\mathbf{BI}(-)^\bullet$-algebras.

6.1 Propositions derivable in the planar logic

In this subsection, we construct $\mathcal{F}$ as a $\mathbf{BI}(-)^\bullet$-algebra.

We define the *planar logic* as a sequent calculus whose formulas are constructed from propositional variables and an implication symbol $\multimap$, and whose derivation rules are the following ones:

$$\frac{}{A \vdash_p A}\ (\text{identity}) \quad \text{where } A \text{ is a formula;} \quad \frac{A, \Gamma \vdash_p B}{\Gamma \vdash_p A \multimap B}\ (\multimap\text{-introduction})\ ;$$

$$\frac{\Gamma \vdash_p A \qquad \Delta \vdash_p A \multimap B}{\Gamma, \Delta \vdash_p B}\ (\multimap\text{-elimination})\ ,$$

where Γ and Δ are sequences of distinct formulas.

Let $\mathcal{F}$ be the powerset of $\{A \mid \vdash_p A \text{ is derivable in the planar logic}\}$. Then $\mathcal{F}$ gives rise to a $\mathbf{BI}(-)^\bullet$-algebra. Indeed, we can define the applicative structure on $\mathcal{F}$ as follows:

- For $M, N \in \mathcal{F}$, the application $MN := \{A_1 \mid \exists A_2 \in N, (A_2 \multimap A_1) \in M\}$.
- $\mathbf{B} := \{(A_1 \multimap A_2) \multimap ((A_3 \multimap A_1) \multimap (A_3 \multimap A_2)) \mid A_1, A_2, A_3 \text{ are formulas}\}$.
- $\mathbf{I} := \{A_1 \multimap A_1 \mid A_1 \text{ is a formula}\}$.
- For $M \in \mathcal{F}$, $M^\bullet := \{(A_1 \multimap A_2) \multimap A_2 \mid A_2 \text{ is a formula and } A_1 \in M\}$.

6.2 Binary trees from ordered groups

In this subsection, we construct $\mathcal{T}$ as a $\mathbf{BII^\times}(-)^\bullet$-algebra.

Take an ordered group $(G, \cdot, e, \leq)$. Let T be a set of binary trees whose leaves are labeled by elements of G:

$$t ::= g \mid t \multimap t \quad (g \in G).$$

We define a function $|\text{-}| : T \to G$ by induction: $|g| := g$ and $|t_1 \multimap t_2| := |t_1|^{-1} \cdot |t_2|$.

Let $\mathcal{T}$ be the powerset of $\{t \in T \mid e \leq |t|\}$. Then $\mathcal{T}$ gives rise to a $\mathbf{BII^\times}(-)^\bullet$-algebra. Indeed, we can define the applicative structure on $\mathcal{T}$ as follows:

- For $M, N \in \mathcal{T}$, $MN := \{t_1 \mid \exists t_2 \in N, (t_2 \multimap t_1) \in M\}$.
- $\mathsf{B} := \{(t_1 \multimap t_2) \multimap ((t_3 \multimap t_1) \multimap (t_3 \multimap t_2)) \mid t_1, t_2, t_3 \in T\}$.
- $\mathsf{I} := \{t_1 \multimap t_1 \mid t_1 \in T\}$.
- $\mathsf{I}^\times := \{t_1 \multimap ((t_2 \multimap t_2) \multimap t_1) \mid t_1, t_2 \in T\}$
- For $M \in \mathcal{T}$, $M^\bullet := \{(t_1 \multimap t_2) \multimap t_2 \mid t_2 \in T, t_1 \in M\}$.

▶ Remark. Whether $\mathcal{T}$ includes C or not depends on G. For instance, when G is Abelian, $\mathcal{T}$ has C as $\{(t_1 \multimap (t_2 \multimap t_3)) \multimap (t_2 \multimap (t_1 \multimap t_3)) \mid t_1, t_2, t_3 \in T\}$.

The example in this subsection is based on $Comod(\overline{G})$ introduced in [6]. $Comod(\overline{G})$ is a category constructed from a group G, whose objects are sets equipped with G valued functions and whose maps are relations between the objects compatible to the valuations. For any (not necessarily ordered) group G, $Comod(\overline{G})$ is a pivotal category. $\mathcal{T}$ is a set of maps from the unit object to a reflexive object of $Comod(\overline{G})$ with alteration for order structures.

6.3 Computational lambda calculus and its models

In this subsection, we show the *untyped computational lambda calculus* [14] is a $\mathsf{BII}^\times(-)^\bullet$-algebra. The following axiomatization is from [15].

Suppose infinite supply of variables $x, y, z, \ldots$. The values, terms and evaluation contexts are defined as follows:

- $V \in \text{Values} ::= x \mid (\lambda x.M)$;
- $M \in \text{Terms} ::= V \mid M(M')$;
- $E \in \text{EvalCtx} ::= [-] \mid EM \mid VE$.

An equivalence relation $=_R$ on Terms is defined as the congruence of the following equations:

1. $\beta_v :\quad (\lambda x.M)V = M[V/x]$;
2. $\eta_v :\quad \lambda x.Vx = V \quad (x \notin FV(V))$;
3. $\beta_\Omega :\quad (\lambda x.E[x])M = E[M] \quad (x \notin FV(E))$,
   where $E[N]$ denotes the term obtained by substituting N for $[-]$ in E.

Then, (Terms/$=_R$) forms a $\mathsf{BII}^\times(-)^\bullet$-algebra. Here the application is that of the computational lambda calculus. $\lambda xyz.x(yz)$, $\lambda x.x$, $\lambda xy.yx$ and $\lambda x.xM$ are representatives of B, I, $\mathsf{I}^\times$ and $M^\bullet$ respectively.

Although the computational lambda calculus consists of the same terms as the ordinary lambda calculus, this example is not a PCA nor a BCI-algebra. Intuitively, the computational lambda calculus is sound for reasoning about effectful programs, which cannot be discarded, duplicated nor exchanged in general; hence it cannot have S, K nor C.

The syntactical proof is as follows. Assume that the computational lambda calculus is a BCI-algebra. Then there exists a term C' such that $\mathsf{C}'MN =_R NM$ in the computational lambda calculus for any term M and N. Take two different variables u and v which are not free in C'. Then $\mathsf{C}'(uu)(vv) =_R (vv)(uu)$ holds in the computational lambda calculus. Since the CPS-translation $[\![-]\!]$ is sound [16], $[\![-]\!]$ sends $\mathsf{C}'(uu)(vv)$ and $(vv)(uu)$ to the $\beta\eta$-equal terms.

$$[\![\mathsf{C}'(uu)(vv)]\!] =_{\beta\eta} \lambda k.[\![\mathsf{C}']\!](\lambda z.uu(\lambda w.zw(\lambda x.vv(\lambda y.xyk))))$$
$$[\![(vv)(uu)]\!] =_{\beta\eta} \lambda k.vv(\lambda x.uu(\lambda y.xyk))$$

The former contains a subterm $uu(...vv...)$, whereas the latter contains a subterm $vv(...uu...)$. Therefore, these two terms are not $\beta\eta$-equal, and it yields a contradiction.

We can also obtain $\mathbf{BII}^{\times}(-)^{\bullet}$-algebras from models of the computational lambda calculus. Take a Cartesian closed category $\mathbf{C}$ and a strong monad T on $\mathbf{C}$ with an object X such that $X \cong (X \to TX)$. Then $\mathbf{C}(1, TX)$ is a model of the computational lambda calculus and forms a $\mathbf{BII}^{\times}(-)^{\bullet}$-algebra.

7 Related work

In [19], Zeilberger showed certain kind of planar maps with orientations are generated by combining several "imploid moves" including those corresponding to **B** and **I**. In particular for an untyped case, his work gives the combinatory completeness of $\mathbf{BI}(-)^{\bullet}$-algebras. Moreover, this paper suggests that we can obtain models of $\mathbf{BI}(-)^{\bullet}$-algebras from reflexive objects of skew closed categories.

In this paper, we deal with several classes of applicative structures other than PCAs nor **BCI**-algebras. [7] is a textbook covering basic facts about combinatory algebras. In [10], Komori investigated $\mathbf{BB'I}$ logic, one of combinatory logics with restricted exchanges. Our $\mathbf{BB'II}^{\bullet}$-algebra is inspired by it and assemblies on $\mathbf{BB'II}^{\bullet}$-algebras are models of $\mathbf{BB'I}$ logics with an extra axiom $\vdash ((\phi \to \phi) \to \psi) \to \psi$. Futhermore, using "guarded merge" introduced in that paper, we can construct lambda terms which form a $\mathbf{BB'II}^{\bullet}$-algebra.

$\mathbf{BI}(-)^{\bullet}$-algebras give rise to models of certain fragments of the Lambek calculus. Correspondences about Lambek calulus and closed multicategories are in [11]. A basic Lambek calculus is $L(\bullet, I, \backslash, /)$ which has products, a unit and implications for both sides, whose models are monoidal biclosed category. Closed multicategories are models of $L(\backslash)$ or $L(/)$. Closed categories are models of $L(I, \backslash)$ or $L(I, /)$.

Realizability models for exponential modalities ! are introduced in [3] [2] as "linear combinatory algebras (LCAs)." In [8], linear/non-linear realizability models are constructed from adjoint pairs between **BCI**-algebras and PCAs. We are currently developing the construction of realizability models for exchange modalities in the same way as exponential modalities. Exchange modalities are modalities associating the Lambek calculus to the commutative linear logic. The characterization of an exchange modality on the Lambek calculus is in [4] and categorical models of exchange modalities are introduced in [9]. A categorical model of exchange modality on $L(\bullet, I, \backslash, /)$ is given as a monoidal adjunction between a monoidal biclosed category and an SMCC. With some appropriate conditions, adjoint pairs between $\mathbf{BI}(-)^{\bullet}$-algebras and **BCI**-algebras may give rise to realizability models of exchange modalities on $L(\backslash)$ or $L(/)$.

8 Conclusion

In this paper, we have presented several classes of applicative structures and identified categorical structures of assemblies on them. In particular, we have shown that $\mathbf{BI}(-)^{\bullet}$-algebras are the necessary and sufficient conditions for obtaining closed multicategories under some conditions, and that $\mathbf{BII}^{\times}(-)^{\bullet}$-algebras are those for closed categories.

There are several directions for further investigation of this paper. We conclude this paper by describing three of future tasks.

First, we may investigate more correspondences between applicative structures and categorical structures of assemblies. We have not given appropriate classes of applicative structures inducing other important categorical structures such as non-symmetric monoidal closed categories nor monoidal biclosed categories.

Second, as mentioned in Section 7, we may characterize realizability models of exchange modalities. Furthermore, if we obtain a class of applicative structures for monoidal biclosed categories, adjoint pairs between such applicative structures and **BCI**-algebras could give rise to categorical models of exchange modalities on $L(\bullet, I, \backslash, /)$.

Third, more examples of $\mathbf{BI}(-)^{\bullet}$-algebras are desired. Finding interesting examples, we may get new perspectives for analyzing non-commutative logical systems.

References

1 Samson Abramsky. Temperley-lieb algebra: from knot theory to logic and computation via quantum mechanics. *arXiv preprint arXiv:0910.2737*, 2009.

2 Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002.

3 Samson Abramsky and Marina Lenisa. A fully complete PER model for ML polymorphic types. In *International Workshop on Computer Science Logic*, pages 140–155. Springer, 2000.

4 Valeria de Paiva and Harley Eades. Dialectica categories for the lambek calculus. In *International Symposium on Logical Foundations of Computer Science*, pages 256–272. Springer, 2018.

5 Samuel Eilenberg and G Max Kelly. Closed categories. In *Proceedings of the Conference on Categorical Algebra*, pages 421–562. Springer, 1966.

6 Masahito Hasegawa. A quantum double construction in Rel. *Mathematical Structures in Computer Science*, 22(4):618–650, 2012.

7 J Roger Hindley and Jonathan P Seldin. *Lambda-calculus and Combinators, an Introduction*, volume 13. Cambridge University Press Cambridge, 2008.

8 Naohiko Hoshino. Linear realizability. In *International Workshop on Computer Science Logic*, pages 420–434. Springer, 2007.

9 Jiaming Jiang, Harley Eades III, and Valeria de Paiva. On the lambek calculus with an exchange modality. *arXiv preprint arXiv:1904.06847*, 2019.

10 Yuichi Komori. Syntactical investigations into BI logic and BB'I logic. *Studia Logica*, 53(3):397–416, 1994.

11 Joachim Lambek. Deductive systems and categories II. standard constructions and closed categories. In *Category theory, homology theory and their applications I*, pages 76–122. Springer, 1969.

12 John R Longley. *Realizability toposes and language semantics*. PhD thesis, University of Edinburgh, 1995.

13 Oleksandr Manzyuk. Closed categories vs. closed multicategories. *Theory and Applications of Categories*, 26(5):132–175, 2012.

14 Eugenio Moggi. Computational lambda-calculus and monads. Technical report ECS-LFCS-88-66, University of Edinburgh, 1988.

15 Amr Sabry. Note on axiomatizing the semantics of control operators. Technical report CIS-TR-96-03, Department of Computer Science, University of Oregon, 1996.

16 Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and symbolic computation*, 6(3-4):289–360, 1993.

17 Ross Street. Skew-closed categories. *Journal of Pure and Applied Algebra*, 217(6):973–988, 2013.

18 Kornel Szlachányi. Skew-monoidal categories and bialgebroids. *Advances in Mathematics*, 231(3-4):1694–1730, 2012.

19 Noam Zeilberger. A theory of linear typings as flows on 3-valent graphs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 919–928. ACM, 2018.

20 Noam Zeilberger and Alain Giorgetti. A correspondence between rooted planar maps and normal plain lambda terms. *Logical Methods in Computer Science*, 11(3):1–39, 2015.