

Computer Science Logic 2018

CSL 2018, September 4–8, 2018, Birmingham, United Kingdom

Edited by

Dan R. Ghica

Achim Jung



Editors

Dan R. Ghica	Achim Jung
School of Computer Science	School of Computer Science
University of Birmingham	University of Birmingham
D.R.Ghica@cs.bham.ac.uk	A.Jung@cs.bham.ac.uk

ACM Classification 2012

General and reference → General conference proceedings, Theory of computation, Software and its engineering → Formal language definitions, Software and its engineering → Formal software verification

ISBN 978-3-95977-088-0

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <http://www.dagstuhl.de/dagpub/978-3-95977-088-0>.

Publication date

August, 2018

Bibliographic information published by the Deutsche Nationalbibliothek

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

License

This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0): <http://creativecommons.org/licenses/by/3.0/legalcode>.



In brief, this license authorises each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.CSL.2018.0

ISBN 978-3-95977-088-0

ISSN 1868-8969

<http://www.dagstuhl.de/lipics>

LIPICs – Leibniz International Proceedings in Informatics

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

Editorial Board

- Luca Aceto (*Chair*, Gran Sasso Science Institute and Reykjavik University)
- Susanne Albers (TU München)
- Christel Baier (TU Dresden)
- Javier Esparza (TU München)
- Michael Mitzenmacher (Harvard University)
- Madhavan Mukund (Chennai Mathematical Institute)
- Anca Muscholl (University Bordeaux)
- Catuscia Palamidessi (INRIA)
- Raimund Seidel (Saarland University and Schloss Dagstuhl – Leibniz-Zentrum für Informatik)
- Thomas Schwentick (TU Dortmund)
- Reinhard Wilhelm (Saarland University)

ISSN 1868-8969

<http://www.dagstuhl.de/lipics>

■ Contents

Preface	
<i>Dan R. Ghica and Achim Jung</i>	0:ix–0:x
The Ackermann Award 2018	
<i>Dexter Kozen and Thomas Schwentick</i>	1:1–1:5

Regular Papers

Relating Structure and Power: Comonadic Semantics for Computational Resources	
<i>Samson Abramsky and Nihil Shah</i>	2:1–2:17
Climbing up the Elementary Complexity Classes with Theories of Automatic Structures	
<i>Faried Abu Zaid, Dietrich Kuske, and Peter Lindner</i>	3:1–3:16
High-Level Signatures and Initial Semantics	
<i>Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi</i>	4:1–4:22
The True Concurrency of Herbrand’s Theorem	
<i>Aurore Alcolei, Pierre Clairambault, Martin Hyland, and Glynn Winskel</i>	5:1–5:22
Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities	
<i>Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper</i>	6:1–6:17
Definable Inapproximability: New Challenges for Duplicator	
<i>Albert Atserias and Anuj Dawar</i>	7:1–7:21
Safety, Absoluteness, and Computability	
<i>Arnon Avron, Shahar Lev, and Nissan Levi</i>	8:1–8:17
Combining Linear Logic and Size Types for Implicit Complexity	
<i>Patrick Baillot and Alexis Ghyselen</i>	9:1–9:21
Beyond Admissibility: Dominance Between Chains of Strategies	
<i>Nicolas Basset, Ismaël Jecker, Arno Pauly, Jean-François Raskin, and Marie Van den Bogaard</i>	10:1–10:22
Rule Algebras for Adhesive Categories	
<i>Nicolas Behr and Paweł Sobociński</i>	11:1–11:21
Submodular Functions and Valued Constraint Satisfaction Problems over Infinite Domains	
<i>Manuel Bodirsky, Marcello Mamino, and Caterina Viola</i>	12:1–12:22
Graphical Conjunctive Queries	
<i>Filippo Bonchi, Jens Seeber, and Paweł Sobociński</i>	13:1–13:23
Approximating Probabilistic Automata by Regular Languages	
<i>Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan</i>	14:1–14:23

27th EACSL Annual Conference on Computer Science Logic.

Editors: Dan R. Ghica and Achim Jung



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An Application of Parallel Cut Elimination in Unit-Free Multiplicative Linear Logic to the Taylor Expansion of Proof Nets <i>Jules Chouquet and Lionel Vaux Auclair</i>	15:1–15:17
Fully Abstract Models of the Probabilistic λ -calculus <i>Pierre Clairambault and Hugo Paquet</i>	16:1–16:17
Uniform Inductive Reasoning in Transitive Closure Logic via Infinite Descent <i>Liron Cohen and Reuben N. S. Rowe</i>	17:1–17:16
A Recursion-Theoretic Characterisation of the Positive Polynomial-Time Functions <i>Anupam Das and Isabel Oitavem</i>	18:1–18:17
Non-Wellfounded Proof Theory For (Kleene+Action)(Algebras+Lattices) <i>Anupam Das and Damien Pous</i>	19:1–19:18
Symmetric Circuits for Rank Logic <i>Anuj Dawar and Gregory Wilsenach</i>	20:1–20:16
Beyond Polarity: Towards a Multi-Discipline Intermediate Language with Sharing <i>Paul Downen and Zena M. Ariola</i>	21:1–21:23
Expressivity Within Second-Order Transitive-Closure Logic <i>Flavio Ferrarotti, Jan Van den Bussche, and Jonni Virtema</i>	22:1–22:18
Quantifying Bounds in Strategy Logic <i>Nathanaël Fijalkow, Bastien Maubert, Aniello Murano, and Sasha Rubin</i>	23:1–23:23
A Fully Abstract Game Semantics for Countable Nondeterminism <i>W. John Gowers and James D. Laird</i>	24:1–24:18
Dependency Concepts up to Equivalence <i>Erich Grädel and Matthias Hoelzel</i>	25:1–25:21
Finite Bisimulations for Dynamical Systems with Overlapping Trajectories <i>Béatrice Bérard, Patricia Bouyer, and Vincent Jugé</i>	26:1–26:17
A Contextual Reconstruction of Monadic Reflection <i>Toru Kawata</i>	27:1–27:14
An Algebraic Decision Procedure for Two-Variable Logic with a Between Relation <i>Andreas Krebs, Kamal Lodaya, Paritosh K. Pandya, and Howard Straubing</i>	28:1–28:17
Basic Operational Preorders for Algebraic Effects in General, and for Combined Probability and Nondeterminism in Particular <i>Aliaume Lopez and Alex Simpson</i>	29:1–29:17
Canonical Models and the Complexity of Modal Team Logic <i>Martin Lück</i>	30:1–30:23
A Decidable Fragment of Second Order Logic With Applications to Synthesis <i>P. Madhusudan, Umang Mathur, Shambwaditya Saha, and Mahesh Viswanathan</i> .	31:1–31:19
Quantitative Foundations for Resource Theories <i>Dan Marsden and Maaïke Zwart</i>	32:1–32:17

On Compositionality of Dinatural Transformations <i>Guy McCusker and Alessio Santamaria</i>	33:1–33:22
Synthesizing Optimally Resilient Controllers <i>Daniel Neider, Alexander Weinert, and Martin Zimmermann</i>	34:1–34:17
Local Validity for Circular Proofs in Linear Logic with Fixed Points <i>Rémi Nollet, Alexis Saurin, and Christine Tasson</i>	35:1–35:23
Parity Games with Weights <i>Sven Schewe, Alexander Weinert, and Martin Zimmermann</i>	36:1–36:17
MacNeille Completion and Buchholz’ Omega Rule for Parameter-Free Second Order Logics <i>Kazushige Terui</i>	37:1–37:19

■ Preface

Computer Science Logic (CSL) is the annual conference of the European Association for Computer Science Logic (EACSL). It is an interdisciplinary conference, spanning across both basic and application oriented research in mathematical logic and computer science. CSL started as a series of international workshops on Computer Science Logic, and became at its sixth meeting the Annual Conference of the EACSL.

The 27th annual EACSL conference Computer Science Logic (CSL 2018) was held in Birmingham (UK) from September 4 to September 7, 2018. It was hosted by the School of Computer Science of the University of Birmingham, and held on its Edgbaston campus.

The conference received 100 abstracts of which 86 were followed up by paper submissions. Each paper was assigned for reviewing to at least three programme committee members, assisted by 132 external reviewers. The reviewing process consisted of two stages. First, submissions with potential technical problems or deemed not original enough were rejected. Less than 15% of submissions fell into this category. Of the remaining papers the 36 submissions deemed as the most interesting were selected for presentation at the conference and publication in these proceedings. The number was dictated by the duration of the conference and individual talks. All papers deemed “very interesting” by at least two members of the PC were accepted, while each accepted paper was deemed as “very interesting” by at least one member.

The invited speakers for this conference were:

- Bob Coecke, University of Oxford
- Emmanuel Filiot, Université libre de Bruxelles
- Catuscia Palamidessi, École polytechnique (Paris-Saclay)
- Christine Tasson, Université Paris Diderot
- Szymon Toruńczyk, Uniwersytet Warszawski

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 2018 to Amina Doumane for her thesis *On the Infinitary Proof Theory of Logics with Fixed Points*. The award was officially presented at the conference on September 7, 2018. The citation of the award, an abstract of the thesis and a biographical sketch of the recipient is included in the proceedings.

We wish to thank all members of the programme committee and all external reviewers for their hard and highly professional work on reviewing and discussing the papers. Our thanks also go to Marco Devesas Campos for maintaining the conference web site and publicising the conference. We also wish to thank Thomas Schwentick who, as the EACSL president, provided useful guidance. Michael Wagner from the Dagstuhl/LIPIcs team assisted us in the production of the proceedings, for which we are grateful.

The conference also hosted the workshop *An Intersection of Neighbourhoods* which took place the day after, September 8th. The workshop, organised by Dan Ghica on behalf of the School of Computer Science of the University of Birmingham, was dedicated to Achim Jung’s contributions to research in domain theory, topological logic, programming language semantics, and computer science education, on the occasion of his 60th birthday. The invited speakers were Samson Abramsky (University of Oxford), Thorsten Altenkirch (University



of Nottingham), Mai Gehrke (Université Côte d'Azur), Michael Huth (Imperial College London), Ho Weng Kin (Nanyang Technological University), Jimmie Lawson (Louisiana State University), Michael Mislove (Tulane University), Frank Pfenning (Carnegie Mellon University), and Alex Simpson (University of Ljubljana).

■ Programme Committee

Christel Baier, TU Dresden
Martin Berger, University of Sussex
Lars Birkedal, Aarhus University
Veronique Bruyere, University of Mons
Agata Ciabattoni, TU Wien
Ugo Dal Lago, University of Bologna
Ross Duncan, University of Strathclyde
Jamie Gabbay, Heriot-Watt University
Marco Gaboardi, University at Buffalo, SUNY
Dan R. Ghica, University of Birmingham (Co-chair)
Russ Harmer, CNRS & ENS Lyon
Achim Jung, University of Birmingham (Co-chair)
Juha Kontinen, University of Helsinki
Jean Krivine, Université Paris Diderot & IRIF
Slawek Lasota, University of Warsaw
Marina Lenisa, University of Udine
Anca Muscholl, University of Bordeaux
Wied Pakusa, RWTH Aachen University
Daniela Petrisan, Université Paris Diderot
Sebastian Siebertz, University of Warsaw
Alexandra Silva, University College London



■ External Reviewers

Adrien Husson,
Ahmet Kara,
Ales Bizjak,
Alessandro Facchini,
Alex Kavvos,
Alex Simpson,
Alexander Leitsch,
Andrea Aler Tubella,
Andrea Schalk,
Andrea Vezzosi,
Ankush Das,
Annika Kanckos,
Anuj Dawar,
Anupam Das,
Arne Meier,
Arno Pauly,
Bahareh Afshari,
Bakhadyr Khoussainov,
Bas Spitters,
Brendan Fong,
Carla Piazza,
Chris Heunen,
Christine Tasson,
Christof Löding,
Chunyan Mu,
Damian Niwinski,
Damiano Mazza,
Daniel de Carvalho,
Daniel Neuen,
Daniel R. Licata,
Didier Galmiche,
Ekaterina
Komendantskaya,
Elaine Pimentel,
Emanuel Kieronski,
Emilio Jesus
Gallego Arias,
Emmanuel Beffara,
Erich Grädel,
Eryk Kopczyński,
Flavien Breuvert,
Francesco Gavazzo,
Fredrik Dahlqvist,
Furio Honsell,
Gabriel Scherer,
Georg Moser,
George Metcalfe,
Gianluca Curzi,
Giulio Guerrieri,
Harsh Beohar,
Heribert Vollmer,
Ian Cassar,
Ian Mackie,
Ivan Scagnetto,
James Brotherston,
James Wood,
James Worrell,
Jean-Francois Raskin,
Joanna Ochremiak,
Jonni Virtema,
Julien Signoles,
Karin Quaas,
Karoliina Lehtinen,
Kazushige Terui,
Kazuyuki Asada,
Kevin Dunne,
Koji Nakazawa,
Kord Eickmeyer,
Krzysztof Kapulkin,
Laure Daviaud,
Laurent Regnier,
Lauri Hella,
Leo Stefanescu,
Lionel Vaux,
Lorenzo Clemente,
Łukasz Czajka,
Manfred Kufleitner,
Marc de Visme,
Marcin Przybyłko,
Marco Comini,
Marco Faella,
Maribel Fernandez,
Marino Miculan,
Mario Alvarez-Picallo,
Markus N. Rabe,
Martin Avanzini,
Martin Lück,
Matteo Sammartino,
Matthijs Vákár,
Maurice Chandoo,
Michael Shulman,
Michele Loreti,
Miika Hannula,
Nao Hirokawa,
Nathanaël Fijalkow,
Nicolai Vorobjov,
Nicolas Markey,
Olivier Laurent,
Paolo Baldi,
Paolo Pistone,
Patrick Gardy,
Paul Blain Levy,
Paweł Sobocinski,
Pierre Bourhis,
Pierre Vial,
Pierre-Marie Pédrot,
Pietro Di Gianantonio,
Pietro Galliani,
Quentin Hautem,
Radu Mardare,
Ranald Clouston,
Rasmus Ibsen-Jensen,
Reiko Heckel,
Reuben Rowe,
Richard Blute,
Romain Péchoux,
Roman Rabinovich,
Samir Genaim,
Sascha Klüppelholz,
Simon Docherty,
Stefan Göller,
Sunil Easaw Simon,
Szymon Toruńczyk,
Tarmo Uustalu,
Tatjana Petrov,
Thomas Place,
Thomas Wies,
Thomas Zeume,
Tom Hirschowitz,
Tom van Dijk,
Ulrich Berger,
Willem Heijltjes,
Youssef Oualhadj.



The Ackermann Award 2018

Dexter Kozen

Computer Science Department, Cornell University, Ithaca, NY 14853, USA
kozen@cs.cornell.edu

Thomas Schwentick

Fakultät für Informatik, TU Dortmund, Dortmund, Germany
thomas.schwentick@udo.edu

Abstract

The Ackermann Award is the EACSL Outstanding Dissertation Award for Logic in Computer Science. It is presented during the annual conference of the EACSL (CSL'xx). This contribution reports on the 2018 edition of the award.

2012 ACM Subject Classification Theory of computation, Software and its engineering → Formal language definitions, Software and its engineering → Formal software verification

Keywords and phrases Ackermann Award

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.1

Category Award Description

1 The Ackermann Award 2018

The fourteenth Ackermann Award is presented at CSL'18 in Birmingham, UK. The 2018 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 2016 and 31 December 2017. The Jury received eleven nominations for the 2018 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 Asia, Europe and North America.

The EACSL Ackermann Award is generously sponsored by the association *Alumni der Informatik Dortmund e. V.*¹

The topics covered a wide range of topics 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 **2018 Ackermann Award** to:

Amina Doumane from France, for her thesis
On the Infinitary Proof Theory of Logics with Fixed Points
approved by the Université Paris Diderot in 2017.

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



Citation

Amina Doumane receives the *2018 Ackermann Award* of the European Association of Computer Science Logic (EACSL) for her thesis

On the infinitary proof theory of logics with fixed points.

Doumane's thesis is a new and significant contribution to proof theory in computation, focussing on logics extended with fixpoints. As a main contribution, it gives the first constructive proof of the completeness of Kozen's axiomatisation of the linear-time mu-calculus, which is an ingenious application of automata over infinite words. Another large part studies the infinitary proof theory of a fixpoint extension of multiplicative additive linear logic, a challenging topic due to the non-well-founded nature of infinitary proofs. The dissertation is lengthy, but sustains a high level of technical sophistication throughout, including a masterful and innovative blend of proof-theoretic and automata-theoretic techniques.

Background of the Thesis

Amina Doumane's thesis lies at the interface between two of the main areas of logic in computer science: proof theory and verification.

Proof theory deals with the definition of formal proof objects and the study of their structure, with a particular emphasis on various forms of computational content in proofs. Indeed, for about fifty years, several proof transformations originally designed to obtain normal forms of proofs (typically, to ease their study in logic) have been shown to correspond to interesting computational mechanisms, often independently implemented in programming languages. This conceptual bridge is known as the *Curry-Howard correspondence*. In its simplest form, it relates proof normalization in natural deduction for intuitionistic logic with program reductions in lambda calculus. It has been further extended to incorporate classical logic, sequent calculus, cut elimination, and focalization, generating in this way a fruitful dialogue between logic and programming.

In *verification*, logic also plays a central role. In this context, one is particularly interested in logics that allow expressive specifications of software systems while remaining decidable. Automata theory is often used for this purpose, exploiting its deep connections with the logics under consideration. One may also rely on deductive systems such as analytic tableaux that are similar to those studied in proof theory, but appear here in the context of verification algorithms.

Amina Doumane has worked more specifically on fixed point logics, also called μ -calculi, such as the modal μ -calculus, but also on first-order logic extended with (co)inductive predicates. To reason informally about these logics, various (co)inductive proof principles have been proposed. Dr. Doumane has formalized and transferred these principles to first-order logic and studied their properties extensively. She allows infinite proofs (non-well-founded derivation trees) while imposing some validity condition to rule out unsound derivations, to obtain formal proofs that may be seen as modelling the informal proofs by infinite descent. This approach, which can be found in some form in many tableaux systems for μ -calculi, is of particular interest since it introduces objects which are close to the (infinitary) semantics of the considered fixed point logics. It often yields useful support for algorithmic methods and provides an intermediate system between semantics and finitary proof systems.

Despite the natural character and the usefulness of such infinitary deduction systems, no general framework had been developed for their study at the beginning of Amina Doumane's PhD. Moreover, infinite proofs had not been considered from the point of view of structural proof theory. The only exception was the seminal work of Luigi Santocanale who proved, together with Jérôme Fortier, that an infinitary sequent calculus – for a purely additive logic – satisfied the cut-elimination property. However, the logical fragment they captured was quite restrictive.

Contributions of the Thesis

In this setting, Amina Doumane has obtained several important results during her PhD, while developing her scientific vision:

- After some initial results on the semantics of linear logic with fixed points in Ludics, the thesis investigates completeness problems in more expressive logics and develops potential connections with ω -automata. Amina Doumane considered the linear-time μ -calculus and, together with David Baelde, Lucca Hirschi and Alexis Saurin, obtained a completeness result restricted to a fragment corresponding to inclusions of Büchi automata. This result is a consequence of the completeness theorem proved by Kaivola in 1995, but the approach differs, relying on infinite proofs to obtain a new and more perspicuous argument.
- The previous work crucially relies on structural aspects of infinitary calculi (notably, the proper distinction of occurrences) which come from proof theory. This has motivated further developments aimed at giving a truly proof-theoretic status to infinite proofs. Specifically, Amina has shown that the infinitary calculus for multiplicative additive linear logic enjoys cut elimination and focalization. These two results form the basis of the modern study of proofs, an open and exciting field of future research, especially regarding the computational expressivity of these calculi. One should note here that, while this result adds only multiplicative connectives to the earlier result by Fortier and Santocanale, this addition is both highly challenging and significant, since it now seems easy to obtain cut elimination for richer systems, e.g., classical first-order logic with fixed points.
- Finally, Amina has pursued her own earlier work on completeness for linear-time μ -calculus. By identifying new connections between infinitary proofs and automata theory (e.g., non-determinization of alternating parity automata), she has managed to obtain a new *constructive* completeness argument; previous completeness proofs were non-constructive. For this result, published at LICS 2017, she has received the *Kleene award for the best student paper*.

Biographical Sketch

Amina Doumane completed her early education in Khouribga and Rabat, Morocco. She obtained a *Mathematical logics master MPRI* in 2013 and a *Computer Science master MPRI* in 2014 at University Paris Diderot. Her PhD work was carried out at the University Paris Diderot under the supervision of David Baelde, Pierre-Louis Curien and Alexis Saurin. Since completing her PhD in 2017, she has been working as a postdoctoral researcher with Damien Pous at ENS Lyon. Besides the already mentioned *Kleene award for the best student paper* at LICS 2017, she received the *Gilles Kahn thesis prize* from the *Société Informatique de France* for her PhD thesis.

2 **Jury**

The jury for the **Ackermann Award 2018** 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),
- Mikołaj Bojańczyk (University of Warsaw),
- Anuj Dawar (University of Cambridge),
- Dexter Kozen (Cornell University),
- Dale Miller (INRIA Saclay), SigLog representative,
- Luke Ong (University of Oxford),
- Simona Ronchi Della Rocca (University of Torino), the vice-president of EACSL,
- Thomas Schwentick (TU Dortmund University), the president of EACSL.

3 **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.

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
Mickaël Randour from Belgium.

2016, Marseille:

Nicolai Kraus from Germany

2017, Stockholm:


Amaury Pouly from France.

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

Relating Structure and Power: Comonadic Semantics for Computational Resources


Samson Abramsky¹

Oxford University Department of Computer Science
Wolfson Building, Parks Road, Oxford OX1 3QD, U.K.
samson.abramsky@cs.ox.ac.uk

 <https://orcid.org/0000-0003-3921-6637>

Nihil Shah

Oxford University Department of Computer Science
Wolfson Building, Parks Road, Oxford OX1 3QD, U.K.
nihil@berkeley.edu

 <https://orcid.org/0000-0003-2844-0828>

Abstract

Combinatorial games are widely used in finite model theory, constraint satisfaction, modal logic and concurrency theory to characterize logical equivalences between structures. In particular, Ehrenfeucht-Fraïssé games, pebble games, and bisimulation games play a central role. We show how each of these types of games can be described in terms of an indexed family of comonads on the category of relational structures and homomorphisms. The index k is a resource parameter which bounds the degree of access to the underlying structure. The coKleisli categories for these comonads can be used to give syntax-free characterizations of a wide range of important logical equivalences. Moreover, the coalgebras for these indexed comonads can be used to characterize key combinatorial parameters: tree-depth for the Ehrenfeucht-Fraïssé comonad, tree-width for the pebbling comonad, and synchronization-tree depth for the modal unfolding comonad. These results pave the way for systematic connections between two major branches of the field of logic in computer science which hitherto have been almost disjoint: categorical semantics, and finite and algorithmic model theory.

2012 ACM Subject Classification Theory of computation → Finite Model Theory, Theory of computation → Categorical semantics

Keywords and phrases Finite model theory, combinatorial games, Ehrenfeucht-Fraïssé games, pebble games, bisimulation, comonads, coKleisli category, coalgebras of a comonad

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.2

1 Introduction

There is a remarkable divide in the field of logic in Computer Science, between two distinct strands: one focussing on semantics and compositionality (“Structure”), the other on expressiveness and complexity (“Power”). It is remarkable because these two fundamental aspects of our field are studied using almost disjoint technical languages and methods, by almost disjoint research communities. We believe that bridging this divide is a major issue in Computer Science, and may hold the key to fundamental advances in the field.

¹ Samson Abramsky’s work was supported by EPSRC grant EP/N018745/1.



In this paper, we develop a novel approach to relating categorical semantics, which exemplifies the first strand, to finite model theory, which exemplifies the second. It builds on the ideas introduced in [2], but goes much further, showing clearly that there is a strong and robust connection, which can serve as a basis for many further developments.

The setting

Relational structures and the homomorphisms between them play a fundamental rôle in finite model theory, constraint satisfaction and database theory. The existence of a homomorphism $A \rightarrow B$ is an equivalent formulation of constraint satisfaction, and also equivalent to the preservation of existential positive sentences [7]. This setting also generalizes what has become a central perspective in graph theory [15].

Model theory and deception

In a sense, the purpose of model theory is “deception”. It allows us to see structures not “as they really are”, *i.e.* up to isomorphism, but only up to *definable properties*, where definability is relative to a logical language \mathcal{L} . The key notion is *logical equivalence* $\equiv^{\mathcal{L}}$. Given structures \mathcal{A}, \mathcal{B} over the same vocabulary:

$$\mathcal{A} \equiv^{\mathcal{L}} \mathcal{B} \stackrel{\Delta}{\iff} \forall \varphi \in \mathcal{L}. \mathcal{A} \models \varphi \iff \mathcal{B} \models \varphi.$$

If a class of structures \mathcal{K} is definable in \mathcal{L} , then it must be saturated under $\equiv^{\mathcal{L}}$. Moreover, for a wide class of cases of interest in finite model theory, the converse holds [20].

The idea of syntax-independent characterizations of logical equivalence is quite a classical one in model theory, exemplified by the Keisler-Shelah theorem [30]. It acquires additional significance in finite model theory, where model comparison games such as Ehrenfeucht-Fraïssé games, pebble games and bisimulation games play a central role [21].

We offer a new perspective on these ideas. We shall study these games, not as external artefacts, but as semantic constructions in their own right. Each model-theoretic comparison game encodes “deception” in terms of limited access to the structure. These limitations are indexed by a parameter which quantifies the resources which control this access. For Ehrenfeucht-Fraïssé games and bisimulation games, this is the number of rounds; for pebble games, the number of pebbles.

Main Results

We now give a conceptual overview of our main results. Technical details will be provided in the following sections.

We shall consider three forms of model comparison game: Ehrenfeucht-Fraïssé games, pebble games and bisimulation games [21]. For each of these notions of game G , and value of the resource parameter k , we shall define a corresponding *comonad* \mathbb{C}_k on the category of relational structures and homomorphisms over some relational vocabulary. For each structure \mathcal{A} , $\mathbb{C}_k\mathcal{A}$ is another structure over the same vocabulary, which encodes the limited access to \mathcal{A} afforded by playing the game on \mathcal{A} with k resources. There is always an associated homomorphism $\varepsilon_{\mathcal{A}} : \mathbb{C}_k\mathcal{A} \rightarrow \mathcal{A}$ (the *counit* of the comonad), so that $\mathbb{C}_k\mathcal{A}$ “covers” \mathcal{A} . Moreover, given a homomorphism $h : \mathcal{A} \rightarrow \mathcal{B}$, there is a *Kleisli coextension* homomorphism $h^* : \mathbb{C}_k\mathcal{A} \rightarrow \mathbb{C}_k\mathcal{B}$. This allows us to form the *coKleisli category* $\text{Kl}(\mathbb{C}_k)$ for the comonad. The objects are relational structures, while the morphisms from \mathcal{A} to \mathcal{B} in $\text{Kl}(\mathbb{C}_k)$ are exactly the homomorphisms of the form $\mathbb{C}_k\mathcal{A} \rightarrow \mathcal{B}$. Composition of these morphisms uses the Kleisli coextension. The connection between this construction and the corresponding form of game G is expressed by the following result:

► **Theorem 1.** *The following are equivalent:*

1. *There is a coKleisli morphism $\mathbb{C}_k\mathcal{A} \rightarrow \mathcal{B}$*
2. *Duplicator has a winning strategy for the existential G-game with k resources, played from \mathcal{A} to \mathcal{B} .*

The existential form of the game has only a “forth” aspect, without the “back”. This means that Spoiler can only play in \mathcal{A} , while Duplicator only plays in \mathcal{B} . This corresponds to the asymmetric form of the coKleisli morphisms $\mathbb{C}_k\mathcal{A} \rightarrow \mathcal{B}$. Intuitively, Spoiler plays in $\mathbb{C}_k\mathcal{A}$, which gives them limited access to \mathcal{A} , while Duplicator plays in \mathcal{B} . The Kleisli coextension guarantees that Duplicator’s strategies can always be lifted to $\mathbb{C}_k\mathcal{B}$; while we can always compose a strategy $\mathbb{C}_k\mathcal{A} \rightarrow \mathbb{C}_k\mathcal{B}$ with the counit on \mathcal{B} to obtain a coKleisli morphism.

This asymmetric form may seem to limit the scope of this approach, but in fact this is not the case. For each of these comonads \mathbb{C}_k , we have the following equivalences:

- $\mathcal{A} \rightrightarrows_k \mathcal{B}$ iff there are coKleisli morphisms $\mathbb{C}_k\mathcal{A} \rightarrow \mathcal{B}$ and $\mathbb{C}_k\mathcal{B} \rightarrow \mathcal{A}$. Note that there need be no relationship between these morphisms.
- $\mathcal{A} \cong_{\text{Kl}(\mathbb{C}_k)} \mathcal{B}$ iff \mathcal{A} and \mathcal{B} are isomorphic in the coKleisli category $\text{Kl}(\mathbb{C}_k)$. This means that there are morphisms $\mathbb{C}_k\mathcal{A} \rightarrow \mathcal{B}$ and $\mathbb{C}_k\mathcal{B} \rightarrow \mathcal{A}$ which are inverses of each other in $\text{Kl}(\mathbb{C}_k)$.

Clearly, $\cong_{\text{Kl}(\mathbb{C}_k)}$ strictly implies \rightrightarrows_k . We can also define an intermediate “back-and-forth” equivalence \leftrightarrow_k , parameterized by a winning condition $W_{\mathcal{A},\mathcal{B}} \subseteq \mathbb{C}_k\mathcal{A} \times \mathbb{C}_k\mathcal{B}$.

For each of our three types of game, there are corresponding fragments \mathcal{L}_k of first-order logic:

- For Ehrenfeucht-Fraïssé games, \mathcal{L}_k is the fragment of quantifier-rank $\leq k$.
- For pebble games, \mathcal{L}_k is the k -variable fragment.
- For bismulation games over relational vocabularies with symbols of arity at most 2, \mathcal{L}_k is the modal fragment [4] with modal depth $\leq k$.

In each case, we write $\exists\mathcal{L}_k$ for the existential positive fragment of \mathcal{L}_k , and $\mathcal{L}_k(\#)$ for the extension of \mathcal{L}_k with counting quantifiers [21].

We can now state our first main result, in a suitably generic form.

► **Theorem 2.** *For finite structures \mathcal{A} and \mathcal{B} :*

- (1) $\mathcal{A} \equiv^{\exists\mathcal{L}_k} \mathcal{B} \iff \mathcal{A} \rightrightarrows_k \mathcal{B}$.
- (2) $\mathcal{A} \equiv^{\mathcal{L}_k} \mathcal{B} \iff \mathcal{A} \leftrightarrow_k \mathcal{B}$.
- (3) $\mathcal{A} \equiv^{\mathcal{L}_k(\#)} \mathcal{B} \iff \mathcal{A} \cong_{\text{Kl}(\mathbb{C}_k)} \mathcal{B}$.

Note that this is really a family of three theorems, one for each type of game G . Thus in each case, we capture the salient logical equivalences in syntax-free, categorical form.

We now turn to the significance of indexing by the resource parameter k . When $k \leq l$, we have a natural inclusion morphism $\mathbb{C}_k\mathcal{A} \rightarrow \mathbb{C}_l\mathcal{A}$, since playing with k resources is a special case of playing with $l \geq k$ resources. This tells us that the smaller k is, the easier it is to find a morphism $\mathbb{C}_k\mathcal{A} \rightarrow \mathcal{B}$. Intuitively, the more we restrict Spoiler’s abilities to access the structure of \mathcal{A} , the easier it is for Duplicator to win the game.

The contrary analysis applies to morphisms $\mathcal{A} \rightarrow \mathbb{C}_k\mathcal{B}$. The smaller k is, the *harder* it is to find such a morphism. Note, however, that if \mathcal{A} is a finite structure of cardinality k , then $\mathcal{A} \rightrightarrows_k \mathbb{C}_k\mathcal{A}$. In this case, with k resources we can access the whole of \mathcal{A} . What can we say when k is strictly smaller than the cardinality of \mathcal{A} ?

It turns out that there is a beautiful connection between these indexed comonads and combinatorial invariants of structures. This is mediated by the notion of *coalgebra*, another fundamental (and completely general) aspect of comonads. A coalgebra for a comonad \mathbb{C}_k on a structure \mathcal{A} is a morphism $\mathcal{A} \rightarrow \mathbb{C}_k\mathcal{A}$ satisfying certain properties. We define the *coalgebra number* of a structure \mathcal{A} , with respect to the indexed family of comonads \mathbb{C}_k , to be the least k such that there is a \mathbb{C}_k -coalgebra on \mathcal{A} .

We now come to our second main result.

► **Theorem 3.**

- For the pebbling comonad, the coalgebra number of \mathcal{A} corresponds precisely to the tree-width of \mathcal{A} .
- For the Ehrenfeucht-Fraïssé comonad, the coalgebra number of \mathcal{A} corresponds precisely to the tree-depth of \mathcal{A} [27].
- For the modal comonad, the coalgebra number of \mathcal{A} corresponds precisely to the modal unfolding depth of \mathcal{A} .

The main idea behind these results is that coalgebras on \mathcal{A} are in bijective correspondence with decompositions of \mathcal{A} of the appropriate form. We thus obtain categorical characterizations of these key combinatorial parameters.

2 Game Comonads

In this section we will define the comonads corresponding to each of the forms of model comparison game we consider.

Firstly, a few notational preliminaries. A relational vocabulary σ is a set of relation symbols R , each with a specified positive integer arity. A σ -structure \mathcal{A} is given by a set A , the universe of the structure, and for each R in σ with arity k , a relation $R^{\mathcal{A}} \subseteq A^k$. A homomorphism $h : \mathcal{A} \rightarrow \mathcal{B}$ is a function $h : A \rightarrow B$ such that, for each relation symbol R of arity k in σ , for all a_1, \dots, a_k in A : $R^{\mathcal{A}}(a_1, \dots, a_k) \Rightarrow R^{\mathcal{B}}(h(a_1), \dots, h(a_k))$. We write $\mathcal{R}(\sigma)$ for the category of σ -structures and homomorphisms.

We shall write $A^{\leq k}$ for the set of non-empty sequences of length $\leq k$ on a set A . We use list notation $[a_1, \dots, a_j]$ for such sequences, and indicate concatenation by juxtaposition. We write $s \sqsubseteq t$ for the prefix ordering on sequences. If $s \sqsubseteq t$, there is a unique s' such that $ss' = t$, which we refer to as the suffix of s in t . For each positive integer n , we define $\mathbf{n} := \{1, \dots, n\}$.

We shall need a few notions on posets. The comparability relation on a poset (P, \leq) is $x \uparrow y$ iff $x \leq y$ or $y \leq x$. A chain in a poset (P, \leq) is a subset $C \subseteq P$ such that, for all $x, y \in C$, $x \uparrow y$. A forest is a poset (F, \leq) such that, for all $x \in F$, the set of predecessors $\downarrow(x) := \{y \in F \mid y \leq x\}$ is a finite chain. The height $\text{ht}(F)$ of a forest F is $\max_C |C|$, where C ranges over chains in F .

We recall that a comonad (G, ε, δ) on a category \mathcal{C} is given by a functor $G : \mathcal{C} \rightarrow \mathcal{C}$, and natural transformations $\varepsilon : G \Rightarrow I$ (the counit), and $\delta : G \Rightarrow G^2$ (the comultiplication), subject to the conditions that the following diagrams commute, for all objects A of \mathcal{C} :

$$\begin{array}{ccc}
 GA & \xrightarrow{\delta_A} & GGA \\
 \delta_A \downarrow & & \downarrow G\delta_A \\
 GGA & \xrightarrow{\delta_{GA}} & GGA \\
 \end{array}
 \qquad
 \begin{array}{ccc}
 GA & \xrightarrow{\delta_A} & GGA \\
 \delta_A \downarrow & \searrow & \downarrow G\varepsilon_A \\
 GGA & \xrightarrow{\varepsilon_{GA}} & GA
 \end{array}$$

An equivalent formulation is *comonad in Kleisli form* [23]. This is given by an object map G , arrows $\varepsilon_A : GA \rightarrow A$ for every object A of \mathcal{C} , and a Kleisli coextension operation which takes $f : GA \rightarrow B$ to $f^* : GA \rightarrow GB$. These must satisfy the following equations:

$$\varepsilon_A^* = \text{id}_{GA}, \quad \varepsilon \circ f^* = f, \quad (g \circ f^*)^* = g^* \circ f^*.$$

We can then extend G to a functor by $Gf = (f \circ \varepsilon)^*$; and if we define the comultiplication $\delta : G \Rightarrow G^2$ by $\delta_A = \text{id}_{GA}^*$, then (G, ε, δ) is a comonad in the standard sense. Conversely, given a comonad (G, ε, δ) , we can define the coextension by $f^* = Gf \circ \delta_A$. This allows us to define the coKleisli category $\text{Kl}(G)$, with objects the same as those of \mathcal{C} , and morphisms from A to B given by the morphisms in \mathcal{C} of the form $GA \rightarrow B$. Kleisli composition of $f : GA \rightarrow B$ with $g : GB \rightarrow C$ is given by $g \bullet f := g \circ f^*$.

2.1 The Ehrenfeucht-Fraïssé Comonad

We shall define a comonad \mathbb{E}_k on $\mathcal{R}(\sigma)$ for each positive integer k . It will be convenient to define \mathbb{E}_k in Kleisli form. For each structure \mathcal{A} , we define a new structure $\mathbb{E}_k\mathcal{A}$, with universe $\mathbb{E}_k\mathcal{A} := A^{\leq k}$. We define the map $\varepsilon_{\mathcal{A}} : \mathbb{E}_k\mathcal{A} \rightarrow \mathcal{A}$ by $\varepsilon_{\mathcal{A}}[a_1, \dots, a_j] = a_j$. For each relation symbol R of arity n , we define $R^{\mathbb{E}_k\mathcal{A}}$ to be the set of n -tuples (s_1, \dots, s_n) of sequences which are pairwise comparable in the prefix ordering, and such that $R^{\mathcal{A}}(\varepsilon_{\mathcal{A}}s_1, \dots, \varepsilon_{\mathcal{A}}s_n)$. Finally, we define the coextension. Given a homomorphism $f : \mathbb{E}_k\mathcal{A} \rightarrow \mathcal{B}$, we define $f^* : A^{\leq k} \rightarrow B^{\leq k}$ by $f^*[a_1, \dots, a_j] = [b_1, \dots, b_j]$, where $b_i = f[a_1, \dots, a_i]$, $1 \leq i \leq j$.

► **Proposition 4.** *The triple $(\mathbb{E}_k, \varepsilon, (\cdot)^*)$ is a comonad in Kleisli form.*

Intuitively, an element of $A^{\leq k}$ represents a play in \mathcal{A} of length $\leq k$. A coKleisli morphism $\mathbb{E}_k\mathcal{A} \rightarrow \mathcal{B}$ represents a Duplicator strategy for the existential Ehrenfeucht-Fraïssé game with k rounds, where Spoiler plays only in \mathcal{A} , and $b_i = f[a_1, \dots, a_i]$ represents Duplicator's response in \mathcal{B} to the i 'th move by Spoiler. The winning condition for Duplicator in this game is that, after k rounds have been played, the induced relation $\{(a_i, b_i) \mid 1 \leq i \leq k\}$ is a partial homomorphism from \mathcal{A} to \mathcal{B} .

These intuitions are confirmed by the following result.

► **Theorem 5.** *The following are equivalent:*

1. *There is a homomorphism $\mathbb{E}_k\mathcal{A} \rightarrow \mathcal{B}$.*
2. *Duplicator has a winning strategy for the existential Ehrenfeucht-Fraïssé game with k rounds, played from \mathcal{A} to \mathcal{B} .*

2.2 The Pebbling Comonad

We now turn to the case of pebble games. The following construction appeared in [2]. Given a structure \mathcal{A} , we define $\mathbb{P}_k\mathcal{A}$, which will represent plays of the k -pebble game on \mathcal{A} .² The universe is $(\mathbf{k} \times A)^+$, the set of finite non-empty sequences of moves (p, a) , where $p \in \mathbf{k}$ is a pebble index, and $a \in A$. We shall use the notation $s = [(p_1, a_1), \dots, (p_n, a_n)]$ for these sequences, which may be of arbitrary length. Thus the universe of $\mathbb{P}_k\mathcal{A}$ is always infinite, even if \mathcal{A} is a finite structure. This is unavoidable, by [2, Theorem 7]. We define $\varepsilon_{\mathcal{A}} : \mathbb{P}_k\mathcal{A} \rightarrow \mathcal{A}$ to send a play $[(p_1, a_1), \dots, (p_n, a_n)]$ to a_n , the A -component of its last move.

Given an n -ary relation $R \in \sigma$, we define $R^{\mathbb{P}_k\mathcal{A}}(s_1, \dots, s_n)$ iff (1) the s_i are pairwise comparable in the prefix ordering; (2) the pebble index of the last move in each s_i does not appear in the suffix of s_i in s_j for any $s_j \supseteq s_i$; and (3) $R^{\mathcal{A}}(\varepsilon_{\mathcal{A}}(s_1), \dots, \varepsilon_{\mathcal{A}}(s_n))$.

Finally, given a homomorphism $f : \mathbb{P}_k\mathcal{A} \rightarrow \mathcal{B}$, we define $f^* : \mathbb{P}_k\mathcal{A} \rightarrow \mathbb{P}_k\mathcal{B}$ by $f^*[(p_1, a_1), \dots, (p_j, a_j)] = [(p_1, b_1), \dots, (p_j, b_j)]$, where $b_i = f[(p_1, a_1), \dots, (p_i, a_i)]$, $1 \leq i \leq j$.

► **Proposition 6.** *The triple $(\mathbb{P}_k, \varepsilon, (\cdot)^*)$ is a comonad in Kleisli form.*

The following is [2, Theorem 13].

► **Theorem 7.** *The following are equivalent:*

1. *There is a homomorphism $\mathbb{P}_k\mathcal{A} \rightarrow \mathcal{B}$.*
2. *There is a winning strategy for Duplicator in the existential k -pebble game from \mathcal{A} to \mathcal{B} .*

² In [2] we used the notation \mathbb{T}_k for this comonad.

2.3 The Modal Comonad

For the modal case, we assume that the relational vocabulary σ contains only symbols of arity at most 2. We can thus regard a σ -structure as a Kripke structure for a multi-modal logic, where the universe is thought of as a set of worlds, each binary relation symbol R_α gives the accessibility relation for one of the modalities, and each unary relation symbol P give the valuation for a corresponding propositional variable. If there are no unary symbols, such structures are exactly the labelled transition systems widely studied in concurrency [25].

Modal logic localizes its notion of satisfaction in a structure to a world. We shall reflect this by using the category of *pointed relational structures* $\mathcal{R}_*(\sigma)$. Objects of this category are pairs (\mathcal{A}, a) where \mathcal{A} is a σ -structure and $a \in A$. Morphisms $h : (\mathcal{A}, a) \rightarrow (\mathcal{B}, b)$ are homomorphisms $h : \mathcal{A} \rightarrow \mathcal{B}$ such that $h(a) = b$. Of course, the same effect could be achieved by expanding the vocabulary σ with a constant, but pointed categories appear in many mathematical contexts.

For each $k > 0$, we shall define a comonad \mathbb{M}_k , where $\mathbb{M}_k(\mathcal{A}, a)$ corresponds to unravelling the structure \mathcal{A} , starting from a , to depth k . The universe of $\mathbb{M}_k(\mathcal{A}, a)$ comprises the unit sequence $[a]$, which is the distinguished element, together with all sequences of the form $[a_0, \alpha_1, a_1, \dots, \alpha_j, a_j]$, where $a = a_0$, $1 \leq j \leq k$, and $R_{\alpha_i}^A(a_i, a_{i+1})$, $0 \leq i < j$. The map $\varepsilon_{\mathcal{A}} : \mathbb{M}_k(\mathcal{A}, a) \rightarrow (\mathcal{A}, a)$ sends a sequence to its last element. Unary relation symbols P are interpreted by $P^{\mathbb{M}_k(\mathcal{A}, a)}(s)$ iff $P^{\mathcal{A}}(\varepsilon_{\mathcal{A}}s)$. For binary relations R_α , the interpretation is $R_\alpha^{\mathbb{M}_k(\mathcal{A}, a)}(s, t)$ iff for some $a' \in A$, $t = s[\alpha, a']$. Given a morphism $f : \mathbb{M}_k(\mathcal{A}, a) \rightarrow (\mathcal{B}, b)$, we define $f^* : \mathbb{M}_k(\mathcal{A}, a) \rightarrow \mathbb{M}_k(\mathcal{B}, b)$ recursively by $f^*[a] = [b]$, $f^*(s[\alpha, a']) = f^*(s)[\alpha, b']$ where $b' = f(s[\alpha, a'])$. This is well-defined since f is a morphism by assumption.

► **Proposition 8.** *The triple $(\mathbb{M}_k, \varepsilon, (\cdot)^*)$ is a comonad in Kleisli form on $\mathcal{R}_*(\sigma)$.*

We recall the notion of *simulation* between Kripke structures [5]. Given structures \mathcal{A} , \mathcal{B} , we define relations $\preceq_k \subseteq A \times B$, $k \geq 0$, by induction on k : $\preceq_0 = A \times B$, and $a \preceq_{k+1} b$ iff (1) for all unary P , $P^{\mathcal{A}}(a)$ implies $P^{\mathcal{B}}(b)$, and (2) for all R_α , if $R_\alpha^{\mathcal{A}}(a, a')$, then for some b' , $R_\alpha^{\mathcal{B}}(b, b')$ and $a' \preceq_k b'$. It is standard that these relations are equivalently formulated in terms of a modified existential Ehrenfeucht-Fraïssé game [5, 14].

► **Theorem 9.** *Let \mathcal{A} , \mathcal{B} be Kripke structures, with $a \in A$ and $b \in B$, and $k > 0$. The following are equivalent:*

1. *There is a homomorphism $f : \mathbb{M}_k(\mathcal{A}, a) \rightarrow (\mathcal{B}, b)$.*
2. *$a \preceq_k b$.*
3. *There is a winning strategy for Duplicator in the k -round simulation game from (\mathcal{A}, a) to (\mathcal{B}, b) .*

3 Logical Equivalences

We now show how our game comonads can be used to give syntax-free characterizations of a range of logical equivalences, which play a central rôle in finite model theory and modal logic.

We shall be considering logics \mathcal{L} which arise as fragments of $\mathcal{L}_{\infty, \omega}$, the extension of first-order logic with infinitary conjunctions and disjunctions, but where formulas contain only finitely many variables. In particular, we will consider the fragments \mathcal{L}_k , of formulas with quantifier rank $\leq k$, and \mathcal{L}^k , the k -variable fragment. These play a fundamental rôle in finite model theory.

We shall also consider two variants for each of these fragments \mathcal{L} . One is the existential positive fragment $\exists\mathcal{L}$, which contains only those formulas of \mathcal{L} built using existential quantifiers, conjunction and disjunction. The other is $\mathcal{L}(\#)$, the extension of \mathcal{L} with counting

quantifiers. These have the form $\exists_{\leq n}$, $\exists_{\geq n}$, where the semantics of $\mathcal{A} \models \exists_{\geq n} x. \psi$ is that there exist at least n distinct elements of \mathcal{A} satisfying ψ .

Each of these logics \mathcal{L} induces an equivalence on structures in $\mathcal{R}(\sigma)$:

$$\mathcal{A} \equiv^{\mathcal{L}} \mathcal{B} \stackrel{\Delta}{\iff} \forall \varphi \in \mathcal{L}. \mathcal{A} \models \varphi \iff \mathcal{B} \models \varphi.$$

Our aim is to characterize these equivalences in terms of our game comonads, and more specifically, to use morphisms in the coKleisli categories as witnesses for these equivalences.

Two equivalences can be defined uniformly for any indexed family of comonads \mathbb{C}_k :

- $\mathcal{A} \rightleftarrows_k^{\mathbb{C}} \mathcal{B}$ iff there are coKleisli morphisms $\mathbb{C}_k \mathcal{A} \rightarrow \mathcal{B}$ and $\mathbb{C}_k \mathcal{B} \rightarrow \mathcal{A}$. Note that there need be no relationship between these morphisms. This is simply the equivalence induced by the preorder collapse of the coKleisli category.
- $\mathcal{A} \cong_k^{\mathbb{C}} \mathcal{B}$ iff \mathcal{A} and \mathcal{B} are isomorphic in the coKleisli category $\text{Kl}(\mathbb{C}_k)$. This means that there are morphisms $\mathbb{C}_k \mathcal{A} \rightarrow \mathcal{B}$ and $\mathbb{C}_k \mathcal{B} \rightarrow \mathcal{A}$ which are inverses of each other in $\text{Kl}(\mathbb{C}_k)$. Clearly, $\cong_k^{\mathbb{C}}$ strictly implies $\rightleftarrows_k^{\mathbb{C}}$.

We shall also define an intermediate, “back-and-forth” equivalence $\leftrightarrow_k^{\mathbb{C}}$. This will be more specific to “game comonads” defined on a concrete category such as $\mathcal{R}(\sigma)$, but it will still be defined and shown to have the appropriate properties in considerable generality. We assume that for each structure \mathcal{A} , the universe $\mathbb{C}_k \mathcal{A}$ has a forest order \sqsubseteq , as seen in our concrete constructions using the prefix ordering on sequences. We add a root \perp for convenience. We write the covering relation for this order as \prec ; thus $s \prec t$ iff $s \sqsubseteq t$, $s \neq t$, and for all u , $s \sqsubseteq u \sqsubseteq t$ implies $u = s$ or $u = t$. We shall also assume that, for any coKleisli morphism $f : \mathbb{C}_k \mathcal{A} \rightarrow \mathcal{B}$, the Kleisli coextension preserves the covering relation: $s \prec s'$ implies $f^*(s) \prec f^*(s')$.

The definition will be parameterized on a set $W_{\mathcal{A}, \mathcal{B}} \subseteq \mathbb{C}_k \mathcal{A} \times \mathbb{C}_k \mathcal{B}$ of “winning positions” for each pair of structures \mathcal{A}, \mathcal{B} . We assume that a function $f : \mathbb{C}_k \mathcal{A} \rightarrow \mathcal{B}$ such that, for all $s \in \mathbb{C}_k \mathcal{A}$, $(s, f^*(s)) \in W_{\mathcal{A}, \mathcal{B}}$, is a coKleisli morphism.

We define the back-and-forth \mathbb{C}_k game between \mathcal{A} and \mathcal{B} as follows.

At the start of each round of the game, the position is specified by $(s, t) \in \mathbb{C}_k \mathcal{A} \times \mathbb{C}_k \mathcal{B}$. The initial position is (\perp, \perp) . The round proceeds as follows. Either Spoiler chooses some $s' \succ s$, and Duplicator responds with $t' \succ t$, resulting in a new position (s', t') ; or Spoiler chooses some $t'' \succ t$ and Duplicator responds with $s'' \succ s$, resulting in (s'', t'') . Duplicator wins the round if the new position is in $W_{\mathcal{A}, \mathcal{B}}$.

We can then define $\mathcal{S}(\mathcal{A}, \mathcal{B})$ to be the set of all functions $f : \mathbb{C}_k \mathcal{A} \rightarrow \mathcal{B}$ such that, for all $s \in \mathbb{C}_k \mathcal{A}$, $(s, f^*(s)) \in W_{\mathcal{A}, \mathcal{B}}$.

We define a *locally invertible pair* (F, G) from \mathcal{A} to \mathcal{B} to be a pair of sets $F \subseteq \mathcal{S}(\mathcal{A}, \mathcal{B})$, $G \subseteq \mathcal{S}(\mathcal{B}, \mathcal{A})$, satisfying the following conditions:

1. For all $f \in F$, $s \in \mathbb{C}_k \mathcal{A}$, for some $g \in G$, $g^* f^*(s) = s$.
2. For all $g \in G$, $t \in \mathbb{C}_k \mathcal{B}$, for some $f \in F$, $f^* g^*(t) = t$.

We define $\mathcal{A} \leftrightarrow_k^{\mathbb{C}} \mathcal{B}$ iff there is a non-empty locally invertible pair from \mathcal{A} to \mathcal{B} .

► **Proposition 10.** *The following are equivalent:*

1. $\mathcal{A} \leftrightarrow_k^{\mathbb{C}} \mathcal{B}$.
2. *There is a winning strategy for Duplicator in the \mathbb{C}_k game between \mathcal{A} and \mathcal{B} .*

Proof. Assuming (1), with a locally invertible pair (F, G) , we define a strategy for Duplicator inductively, such that after each round, the play is within the set

$$\{(s, f^*(s)) \mid s \in \mathbb{C}_k \mathcal{A}, f \in F\} = \{(g^*(t), t) \mid t \in \mathbb{C}_k \mathcal{B}, g \in G\}.$$

Assume (s, t) has been played. If Spoiler now plays $s' \succ s$ in $\mathbb{C}_k A$, then there is $f \in F$ such that $f^*(s) = t$, and we respond with $t' = f^*(s') \succ f^*(s)$. Since $f \in \mathcal{S}(\mathcal{A}, \mathcal{B})$, $(s', t') \in W_{\mathcal{A}, \mathcal{B}}$. The case when Spoiler plays in $\mathbb{C}_k B$ is symmetric.

Assuming (2), let Φ be the set of all plays (s, t) following the Duplicator strategy. Define

$$\begin{aligned} F &:= \{f : \mathbb{C}_k A \rightarrow B \mid \forall s \in \mathbb{C}_k A. (s, f^*(s)) \in \Phi\}, \\ G &:= \{g : \mathbb{C}_k B \rightarrow A \mid \forall t \in \mathbb{C}_k B. (g^*(t), t) \in \Phi\}. \end{aligned}$$

Since the strategy is winning, $\Phi \subseteq W_{\mathcal{A}, \mathcal{B}}$, and $F \subseteq \mathcal{S}(\mathcal{A}, \mathcal{B})$, $G \subseteq \mathcal{S}(\mathcal{B}, \mathcal{A})$. We claim that for all $(s, t) \in \Phi$: (A) $\exists f \in F. f^*(s) = t$, and (B) $\exists g \in G. g^*(t) = s$. (A) follows by extending (s, t) to a morphism $f : \mathbb{C}_k A \rightarrow B$. For any $s' \sqsubseteq s$, we assign the corresponding predecessor of t . For any s' which is not a predecessor of s , let $s_1 = s \sqcap s'$, the meet of s and s' . We write t_1 for the corresponding predecessor of t . We define f on s' by assigning t_1 in response to s_1 , and then following Duplicator's responses as Spoiler plays according to s' in $\mathbb{C}_k A$. (B) follows by a symmetric argument.

Now for any $f \in F$ and $s \in \mathbb{C}_k A$, $(s, f^*(s)) \in \Phi$, and hence by (B) we can find $g \in G$ to witness local invertibility; the case for $g \in G$ and $t \in \mathbb{C}_k B$ is symmetric. \blacktriangleleft

The local invertibility condition on a pair of sets (F, G) has a fixpoint characterization, which may be of some interest. We define set functions $\Gamma : \mathcal{P}(\mathcal{S}(\mathcal{A}, \mathcal{B})) \rightarrow \mathcal{P}(\mathcal{S}(\mathcal{B}, \mathcal{A}))$, $\Delta : \mathcal{P}(\mathcal{S}(\mathcal{B}, \mathcal{A})) \rightarrow \mathcal{P}(\mathcal{S}(\mathcal{A}, \mathcal{B}))$:

$$\begin{aligned} \Gamma(F) &= \{g \in T \mid \forall t \in \mathbb{C}_k B. \exists f \in F. f^* g^* t = t\}, \\ \Delta(G) &= \{f \in S \mid \forall s \in \mathbb{C}_k A. \exists g \in G. g^* f^* s = s\}. \end{aligned}$$

These functions are monotone. Moreover, a pair of sets (F, G) is locally invertible iff $F \subseteq \Delta(G)$ and $G \subseteq \Gamma(F)$. These conditions in turn imply that $F \subseteq \Delta\Gamma(F)$, and if this holds, then we can set $G := \Gamma(F)$ to obtain a locally invertible pair (F, G) . Thus existence of a locally invertible pair is equivalent to the existence of non-empty F such that $F \subseteq \Theta(F)$, where $\Theta = \Delta\Gamma$. Since Θ is monotone, by Knaster-Tarski this is equivalent to the greatest fixpoint of Θ being non-empty. (Note that $\Theta(\emptyset) = \emptyset$).

If \mathcal{A} and \mathcal{B} are finite, so is S , and we can construct the greatest fixpoint by a finite descending sequence $S \supseteq \Theta(S) \supseteq \Theta^2(S) \supseteq \dots$. This fixpoint is non-empty iff $\mathcal{A} \leftrightarrow_k^{\mathbb{E}} \mathcal{B}$.

We shall now turn to a detailed study of each of our comonads in turn.

3.1 The Ehrenfecht-Fraïssé comonad

A coKleisli morphism $f : \mathbb{E}_k \mathcal{A} \rightarrow \mathcal{B}$ is an I -morphism if $s \sqsubseteq t$ and $\varepsilon_{\mathcal{A}}(s) = \varepsilon_{\mathcal{A}}(t)$ implies that $f(s) = f(t)$. An equivalent statement is that, if we add a binary relation symbol I to the vocabulary, and set $I^{\mathcal{A}}$ to be the identity relation on A , and $I^{\mathcal{B}}$ to be the identity relation on B , then f is also a homomorphism with respect to I . The significance of this condition is that, if $f : \mathbb{E}_k \mathcal{A} \rightarrow \mathcal{B}$ and $g : \mathbb{E}_k \mathcal{B} \rightarrow \mathcal{A}$ are I -morphisms, then $f^*(s) = t$, $g^*(t) = s$ imply that (s, t) defines a partial isomorphism from \mathcal{A} to \mathcal{B} . We refine the definition of the equivalence $\cong_k^{\mathbb{E}}$ accordingly. We say that $\mathcal{A} \cong_k^{\mathbb{E}} \mathcal{B}$ iff there are I -morphisms $f : \mathbb{E}_k \mathcal{A} \rightarrow \mathcal{B}$ and $g : \mathbb{E}_k \mathcal{B} \rightarrow \mathcal{A}$ with $f^{*-1} = g^*$.

Note that, for any coKleisli morphism $f : \mathbb{E}_k \mathcal{A} \rightarrow \mathcal{B}$, there is an I -morphism $f_I : \mathbb{E}_k \mathcal{A} \rightarrow \mathcal{B}$, obtained by firstly restricting f to non-repeating sequences, then extending it by applying the I -morphism condition for repetitions. It is easy to verify that f_I is a homomorphism. Thus there is no need to modify the equivalence $\cong_k^{\mathbb{E}}$.

We define $W_{\mathcal{A},\mathcal{B}}^{\mathbb{E}_k}$ to be the set of pairs $(s, t) \in \mathbb{E}_k\mathcal{A} \times \mathbb{E}_k\mathcal{B}$ such that $s = [a_1, \dots, a_j]$, $t = [b_1, \dots, b_j]$, and $\{(a_i, b_i) \mid 1 \leq i \leq j\}$ defines a partial isomorphism from \mathcal{A} to \mathcal{B} . This specifies the back-and-forth equivalence $\leftrightarrow_k^{\mathbb{E}}$.

We now recall the *bijection game* [16]. In this variant of the Ehrenfeucht-Fraïssé game, Spoiler wins if the two structures have different cardinality. Otherwise, at round i , Duplicator chooses a bijection ψ_i between A and B , and Spoiler chooses an element a_i of A . This determines the choice by Duplicator of $b_i = \psi_i(a_i)$. Duplicator wins after k rounds if the relation $\{(a_i, b_i) \mid 1 \leq i \leq k\}$ is a partial isomorphism.

► **Proposition 11.** *The following are equivalent, for finite structures \mathcal{A} and \mathcal{B} :*

1. $\mathcal{A} \cong_k^{\mathbb{E}} \mathcal{B}$.
2. *There is a winning strategy for Duplicator in the k -round bijection game.*

Proof. Assuming (1), we have I -morphisms $f : \mathbb{E}_k\mathcal{A} \rightarrow \mathcal{B}$ and $g : \mathbb{E}_k\mathcal{B} \rightarrow \mathcal{A}$ with $g^* = f^{*-1}$. For each $s \in \{\emptyset\} \cup A^{<k}$, we can define a map $\psi_s : A \rightarrow B$, by $\psi_s(a) = f(s[a])$. This is a bijection, with inverse defined similarly from g . These bijections provide a strategy for Duplicator. Since each $(s, f^*(s))$ is a partial isomorphism, this is a winning strategy.

Conversely, a winning strategy provides bijections ψ_s , which we can use to define f by $f(s[a]) = \psi_s(a)$. The winning conditions imply that this is an I -isomorphism in the coKleisli category. ◀

We can now state our main result on logical equivalences for the Ehrenfeucht-Fraïssé comonad.

► **Theorem 12.**

1. *For all structures \mathcal{A} and \mathcal{B} : $\mathcal{A} \equiv^{\exists\mathcal{L}^k} \mathcal{B} \iff \mathcal{A} \rightleftharpoons_k^{\mathbb{E}} \mathcal{B}$.*
2. *For all structures \mathcal{A} and \mathcal{B} : $\mathcal{A} \equiv^{\mathcal{L}^k} \mathcal{B} \iff \mathcal{A} \leftrightarrow_k^{\mathbb{E}} \mathcal{B}$.*
3. *For all finite structures \mathcal{A} and \mathcal{B} : $\mathcal{A} \equiv^{\mathcal{L}^k(\#)} \mathcal{B} \iff \mathcal{A} \cong_k^{\mathbb{E}} \mathcal{B}$.*

Proof. (1) follows from Theorem 5 and standard results [19]. (2) follows from Proposition 10 and the Ehrenfeucht-Fraïssé theorem [11]. (3) follows from Proposition 11 and results originating in [16] and expounded in [21]. ◀

If we modify $W_{\mathcal{A},\mathcal{B}}^{\mathbb{E}_k}$, and hence $\leftrightarrow_k^{\mathbb{E}}$, by asking for partial correspondences rather than partial isomorphisms, we obtain a characterization of elementary equivalence for equality-free logic [6].

3.2 The Pebbling Comonad

A similar notion of I -morphism applies to the pebbling comonad as we saw previously with the Ehrenfeucht-Fraïssé comonad [2].

Given $s = [(p_1, a_1), \dots, (p_n, a_n)] \in \mathbb{P}_k\mathcal{A}$ and $t = [(p_1, b_1), \dots, (p_n, b_n)] \in \mathbb{P}_k\mathcal{B}$, we define $\phi_{s,t} = \{(a_p, b_p) \mid p \in \mathbf{k}, p \text{ occurs in } s\}$, where the last occurrence of p in s is on a_p , and the corresponding last occurrence in t is on b_p . We define $W_{\mathcal{A},\mathcal{B}}^{\mathbb{P}_k}$ to be the set of all such (s, t) for which $\phi_{s,t}$ is a partial isomorphism. This specifies the back-and-forth equivalence $\leftrightarrow_k^{\mathbb{P}}$.

We now state the following result, characterizing the equivalences induced by finite-variable logics \mathcal{L}^k .

► **Theorem 13.**

1. *For all structures \mathcal{A} and \mathcal{B} : $\mathcal{A} \equiv^{\exists\mathcal{L}^k} \mathcal{B} \iff \mathcal{A} \rightleftharpoons_k^{\mathbb{P}} \mathcal{B}$.*
2. *For all finite structures \mathcal{A} and \mathcal{B} : $\mathcal{A} \equiv^{\mathcal{L}^k} \mathcal{B} \iff \mathcal{A} \leftrightarrow_k^{\mathbb{P}} \mathcal{B}$.*
3. *For all finite structures \mathcal{A} and \mathcal{B} : $\mathcal{A} \equiv^{\mathcal{L}^k(\#)} \mathcal{B} \iff \mathcal{A} \cong_k^{\mathbb{P}} \mathcal{B}$.*

Proof. This follows from Theorems 14, 18 and 20 of [2]. ◀

3.3 The Modal Comonad

The key notion of equivalence in modal logic is bisimulation [5, 29]. We shall define the finite approximants to bisimulation [17].³ Given Kripke structures \mathcal{A} and \mathcal{B} , we define a family of relations $\sim_k \subseteq A \times B$: $\sim_0 = A \times B$; $a \sim_{k+1} b$ iff (1) for all unary P , $P^{\mathcal{A}}(a)$ iff $P^{\mathcal{B}}(b)$; and (2) for all binary R_α , $R_\alpha^{\mathcal{A}}(a, a')$ implies for some b' , $R_\alpha^{\mathcal{B}}(b, b')$ and $a' \sim_k b'$, and $R_\alpha^{\mathcal{B}}(b, b')$ implies for some a' , $R_\alpha^{\mathcal{A}}(a, a')$ and $a' \sim_k b'$.

We define $W_{\mathcal{A}, \mathcal{B}}^{\mathbb{M}_k}$ to be the set of all $(s, t) \in \mathbb{M}_k(\mathcal{A}, a) \times \mathbb{M}_k(\mathcal{B}, b)$ such that $s = [a_0, \alpha_1, a_1, \dots, \alpha_j, a_j]$, $t = [b_0, \alpha_1, b_1, \dots, \alpha_j, b_j]$, and for all i and all unary P , $P^{\mathcal{A}}(a_i)$ iff $P^{\mathcal{B}}(b_i)$. This specifies the back-and-forth equivalence $\leftrightarrow_k^{\mathbb{M}}$.

► **Theorem 14.** *For pointed Kripke structures (\mathcal{A}, a) and (\mathcal{B}, b) : $a \sim_k b$ iff $(\mathcal{A}, a) \leftrightarrow_k^{\mathbb{M}} (\mathcal{B}, b)$.*

Turning to logic, we will consider \mathcal{M}_k , the *modal fragment* of modal depth $\leq k$. This arises from the standard translation of (multi)modal logic into $\mathcal{L}_{\infty, \omega}$ [5]. Let us fix a relational vocabulary σ with symbols of arity ≤ 2 . For each unary symbol P , there will be a corresponding propositional variable p . Formulas are built from these propositional variables by propositional connectives, and modalities \Box_α , \Diamond_α corresponding to the binary relation symbols R_α in σ . Modal formulas φ then admit a translation into formulas $\llbracket \varphi \rrbracket = \psi(x)$ in one free variable. The translation sends propositional variables p to $P(x)$, commutes with the propositional connectives, and sends $\Diamond_\alpha \varphi$ to $\exists y. R_\alpha(x, y) \wedge \psi(y)$, where $\psi(x) = \llbracket \varphi \rrbracket$. This translation is semantics-preserving: given a σ -structure \mathcal{A} and $a \in A$, then $\mathcal{A}, a \models \varphi$ in the sense of Kripke semantics iff $\mathcal{A} \models \psi(a)$ in the standard model-theoretic sense, where $\psi(x) = \llbracket \varphi \rrbracket$.

We define the modal depth of a modal formula φ as the maximum nesting depth of modalities occurring in φ . \mathcal{M}_k is then the image of the translation of modal formulas of modal depth $\leq k$. The existential positive fragment $\exists \mathcal{M}_k$ arises from the modal sublanguage in which formulas are built from propositional variables using only conjunction, disjunction and the diamond modalities \Diamond_α .

Extensions of the modal language with counting capabilities have been studied in the form of *graded modalities* [10]. These have the form \Diamond_α^n , \Box_α^n , where $\mathcal{A}, a \models \Diamond_\alpha^n \varphi$ if there are at least n R_α -successors of a which satisfy φ . We define $\mathcal{M}_k(\#)$ to be the extension of the modal fragment with graded modalities.

A corresponding notion of graded bisimulation is given in [10]. This is in turn related to *resource bisimulation* [8], which has been introduced in the concurrency setting. The two notions are shown to coincide for image-finite Kripke structures in [3], who also show that they can be presented in a simplified form. We recall that a Kripke structure \mathcal{A} is image-finite if for all $a \in A$ and R_α , $R_\alpha(a) := \{a' \mid R_\alpha^{\mathcal{A}}(a, a')\}$ is finite.

Adapting the results in [3], we define approximants \sim_k^g for graded bisimulation: $\sim_0^g = A \times B$, and $a \sim_{k+1}^g b$ if for all P , $P^{\mathcal{A}}(a)$ iff $P^{\mathcal{B}}(b)$, and for all R_α , there is a bijection $\theta : R_\alpha^{\mathcal{A}}(a) \cong R_\alpha^{\mathcal{B}}(b)$ such that, for all $a' \in R_\alpha^{\mathcal{A}}(a)$, $a' \sim_k^g \theta(a')$.

We can also define a corresponding graded bisimulation game between (\mathcal{A}, a) and (\mathcal{B}, b) . At round 0, the elements $a_0 = a$ and $b_0 = b$ are chosen. Duplicator wins if for all P , $P^{\mathcal{A}}(a)$ iff $P^{\mathcal{B}}(b)$, otherwise Spoiler wins. At round $i + 1$, Spoiler chooses some R_α , and Duplicator chooses a bijection $\theta_i : R_\alpha^{\mathcal{A}}(a_i) \cong R_\alpha^{\mathcal{B}}(b_i)$. If there is no such bijection, Spoiler wins. Otherwise, Spoiler then chooses $a_{i+1} \in R_\alpha^{\mathcal{A}}(a_i)$, and $b_{i+1} := \theta_i(a_{i+1})$. Duplicator wins this round if for all P , $P^{\mathcal{A}}(a_{i+1})$ iff $P^{\mathcal{B}}(b_{i+1})$, otherwise Spoiler wins.

³ Our focus on finite approximants in this paper is for uniformity, and because they are relevant in resource terms. We can extend the comonadic semantics beyond the finite levels. We shall return to this point in the final section.

This game is evidently analogous to the bijection game we encountered previously.

► **Proposition 15.** *The following are equivalent:*

1. *There is a winning strategy for Duplicator in the k -round graded bisimulation game between (\mathcal{A}, a) and (\mathcal{B}, b) .*
2. $a \sim_k^{\mathcal{G}} b$.
3. $(\mathcal{A}, a) \cong_k^{\mathbb{M}} (\mathcal{B}, b)$.

► **Theorem 16.**

1. *For all Kripke structures \mathcal{A} and \mathcal{B} : $\mathcal{A} \equiv^{\exists \mathcal{M}_k} \mathcal{B} \iff \mathcal{A} \rightleftarrows_k^{\mathbb{M}} \mathcal{B}$.*
2. *For all Kripke structures \mathcal{A} and \mathcal{B} : $\mathcal{A} \equiv^{\mathcal{M}_k} \mathcal{B} \iff \mathcal{A} \leftrightarrow_k^{\mathbb{M}} \mathcal{B}$.*
3. *For all image-finite Kripke structures \mathcal{A} and \mathcal{B} : $\mathcal{A} \equiv^{\mathcal{M}_k(\#)} \mathcal{B} \iff \mathcal{A} \cong_k^{\mathbb{M}} \mathcal{B}$.*

Proof. (1) follows from Proposition 9 and standard results on preservation of existential positive modal formulas by simulations [5]. (2) follows from Theorem 14 and the Hennessy-Milner Theorem [17, 5]. (3) follows from Proposition 15 and the results in [10, 3]. ◀

4 Coalgebras and combinatorial parameters

Another fundamental aspect of comonads is that they have an associated notion of *coalgebra*. A coalgebra for a comonad (G, ε, δ) is a morphism $\alpha : A \rightarrow GA$ such that the following diagrams commute:

$$\begin{array}{ccc} A & \xrightarrow{\alpha} & GA \\ \alpha \downarrow & & \downarrow \delta_A \\ GA & \xrightarrow{G\alpha} & G^2A \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{\alpha} & GA \\ \text{id}_A \searrow & & \downarrow \varepsilon_A \\ & & A \end{array}$$

Our use of indexed comonads \mathbb{C}_k opens up a new kind of question for coalgebras. Given a structure \mathcal{A} , we can ask: what is the least value of k such that a \mathbb{C}_k -coalgebra exists on \mathcal{A} ? We call this the *coalgebra number* of \mathcal{A} . We shall find that for each of our comonads, the coalgebra number is a significant combinatorial parameter of the structure.

4.1 The Ehrenfeucht-Fraïssé comonad and tree-depth

A graph is $G = (V, \frown)$, where V is the set of vertices, and \frown is the adjacency relation, which is symmetric and irreflexive. A forest cover for G is a forest (F, \leq) such that $V \subseteq F$, and if $v \frown v'$, then $v \uparrow v'$. The tree-depth $\text{td}(G)$ is defined to be $\min_F \text{ht}(F)$, where F ranges over forest covers of G .⁴ It is clear that we can restrict to forest covers of the form (V, \leq) , since given a forest cover (F, \leq) of $G = (V, \frown)$, $(V, \leq \cap V^2)$ is also a forest cover of G , and $\text{ht}(V) \leq \text{ht}(F)$. Henceforth, by forest covers of G we shall mean those with universe V .

Given a σ -structure \mathcal{A} , the Gaifman graph $\mathcal{G}(\mathcal{A})$ is (A, \frown) , where $a \frown a'$ iff for some relation $R \in \sigma$, for some $(a_1, \dots, a_n) \in R^{\mathcal{A}}$, $a = a_i$, $a' = a_j$, $a \neq a'$. The tree-depth of \mathcal{A} is $\text{td}(\mathcal{G}(\mathcal{A}))$.

► **Theorem 17.** *Let \mathcal{A} be a finite σ -structure, and $k > 0$. There is a bijective correspondence between*

1. \mathbb{E}_k -coalgebras $\alpha : \mathcal{A} \rightarrow \mathbb{E}_k \mathcal{A}$.
2. Forest covers of $\mathcal{G}(\mathcal{A})$ of height $\leq k$.

⁴ We formulate this notion in order-theoretic rather than graph-theoretic language, but it is equivalent to the definition in [27].

Proof. Suppose that $\alpha : \mathcal{A} \rightarrow \mathbb{E}_k \mathcal{A}$ is a coalgebra. For $a \in A$, let $\alpha(a) = [a_1, \dots, a_j]$. The first coalgebra equation says that $\alpha(a_i) = [a_1, \dots, a_i]$, $1 \leq i \leq j$. The second says that $a_j = a$. Thus $\alpha : A \rightarrow A^{\leq k}$ is an injective map whose image is a prefix-closed subset of $A^{\leq k}$. Defining $a \leq a'$ iff $\alpha(a) \sqsubseteq \alpha(a')$ yields a forest order on A , of height $\leq k$. If $a \frown a'$ in $\mathcal{G}(\mathcal{A})$, for some a_1, \dots, a_n with $a = a_i$, $a' = a_j$, we have $R^{\mathcal{A}}(a_1, \dots, a_n)$. Since α is a homomorphism, we must have $R^{\mathbb{E}_k \mathcal{A}}(\alpha(a_1), \dots, \alpha(a_n))$, hence $\alpha(a_i) \uparrow \alpha(a_j)$, and so $a_i \uparrow a_j$. Thus (A, \leq) is a forest cover of \mathcal{A} , of height $\leq k$.

Conversely, given such a forest cover (A, \leq) , for each $a \in A$, its predecessors form a chain $a_1 < \dots < a_j$, with $a_j = a$, and $j \leq k$. We define $\alpha(a) = [a_1, \dots, a_j]$, which yields a map $\alpha : A \rightarrow A^{\leq k}$, which evidently satisfies the coalgebra equations. If $R^{\mathcal{A}}(a_1, \dots, a_n)$, then since (A, \leq) is a forest cover, we must have $a_i \uparrow a_j$ for all i, j , and hence $\alpha(a_i) \uparrow \alpha(a_j)$. Thus α is a homomorphism. ◀

We write $\kappa^{\mathbb{E}}(\mathcal{A})$ for the coalgebra number of \mathcal{A} with respect to the the Ehrenfeucht-Fraïssé comonad.

► **Theorem 18.** *For all finite structures \mathcal{A} : $\text{td}(\mathcal{A}) = \kappa^{\mathbb{E}}(\mathcal{A})$.*

4.2 The pebbling comonad and tree-width

We review the notions of tree decompositions and tree-width. A tree (T, \leq) is a forest with a least element (the root). A tree is easily seen to be a meet-semilattice: every pair of elements x, x' has a greatest lower bound $x \wedge x'$ (the greatest common ancestor). The path from x to x' is the set $\text{path}(x, x') := [x \wedge x', x] \cup [x \wedge x', x']$, where we use interval notation: $[y, y'] := \{z \in T \mid y \leq z \leq y'\}$.

A tree-decomposition of a graph $G = (V, \frown)$ is a tree (T, \leq) together with a labelling function $\lambda : T \rightarrow \mathcal{P}(V)$ satisfying the following conditions:

- (TD1) for all $v \in V$, for some $x \in T$, $v \in \lambda(x)$;
- (TD2) if $v \frown v'$, then for some $x \in T$, $\{v, v'\} \subseteq \lambda(x)$;
- (TD3) if $v \in \lambda(x) \cap \lambda(x')$, then for all $y \in \text{path}(x, x')$, $v \in \lambda(y)$.

The width of a tree decomposition is given by $\max_{x \in T} |\lambda(x)| - 1$. We define the tree-width $\text{tw}(G)$ of a graph G as $\min_T \text{width}(T)$, where T ranges over tree decompositions of G .

We shall now give an alternative formulation of tree-width which will provide a useful bridge to the coalgebraic characterization. It is also interesting in its own right: it clarifies the relationship between tree-width and tree-depth, and shows how pebbling arises naturally in connection with tree-width.

A k -pebble forest cover for a graph $G = (V, \frown)$ is a forest cover (V, \leq) together with a pebbling function $p : V \rightarrow \mathbf{k}$ such that, if $v \frown v'$ with $v \leq v'$, then for all $w \in (v, v']$, $p(v) \neq p(w)$.

The following result is implicit in [2], but it seems worthwhile to set it out more clearly.

► **Theorem 19.** *Let G be a finite graph. The following are equivalent:*

1. G has a tree decomposition of width $< k$.
2. G has a k -pebble forest cover.

Proof. (1) \Rightarrow (2). Assume that $G = (V, \frown)$ has a tree decomposition (T, \leq, λ) of width $< k$. We say that a tree decomposition is *orderly* if it has the following property: for all $x \in T$, there is at most one $v \in \lambda(x)$ such that for all $x' < x$, $v \notin \lambda(x')$. Nice tree decompositions are orderly [18]; hence by standard results, without loss of generality we can assume that the given tree decomposition is orderly.

For any $v \in V$, the set of $x \in T$ such that $v \in \lambda(x)$ is non-empty by (TD1), and closed under meets by (TD3). Since T is a tree, this implies that this set has a least element $\tau(v)$. This defines a function $\tau : V \rightarrow T$. The fact that tree decomposition is orderly implies that τ is injective. We can define an order on V by $v \leq v'$ iff $\tau(v) \leq \tau(v')$. This is isomorphic to a sub-poset of T , and hence is a forest order.

We define $p : V \rightarrow \mathbf{k}$ by induction on this order. Assuming $p(v')$ is defined for all $v' < v$, we consider $\tau(v)$. Since the tree decomposition is orderly, this means in particular that $p(v')$ is defined for all $v' \in S := \lambda(\tau(v)) \setminus \{v\}$. Since the decomposition is of width $< k$, we must have $|S| < k$. We set $p(v) := \min(\mathbf{k} \setminus \{p(v') \mid v' \in S\})$.

To verify that (V, \leq) is a forest cover, suppose that $v \frown v'$. By (TD2), for some $x \in T$, $\{v, v'\} \subseteq \lambda(x)$. We have $\tau(v) \leq x \geq \tau(v')$, and since T is a tree, we must have $\tau(v) \uparrow \tau(v')$, whence $v \uparrow v'$.

Finally, we must verify the condition on the pebbling function p . Suppose that $v \frown v'$, and $v < w \leq v'$. Since $v \frown v'$, for some x , $\{v, v'\} \subseteq \lambda(x)$. But then $\tau(v) < \tau(w) \leq \tau(v') \leq x$. Since $v \in \lambda(\tau(v)) \cap \lambda(x)$, by (TD3), $v \in \lambda(\tau(w))$. By construction of the pebbling function, this implies $p(v) \neq p(w)$.

(2) \Rightarrow (1). Suppose that (V, \leq, p) is a k -pebble forest cover of G . We define a tree $T = V_\perp$ by adjoining a least element \perp to V . We say that v is an active predecessor of v' if $v \leq v'$, and for all $w \in (v, v']$, $p(v) \neq p(w)$. We define the labelling function by setting $\lambda(v)$ to be the set of active predecessors of v ; $\lambda(\perp) := \emptyset$. Since $p|_{\lambda(v)}$ is injective, $|\lambda(v)| \leq k$.

We verify the tree decomposition conditions. (TD1) holds, since $v \in \lambda(v)$. (TD2) If $v \frown v'$, then $v \uparrow v'$. Suppose $v \leq v'$. Then v is an active predecessor of v' , and $\{v, v'\} \subseteq \lambda(v')$. (TD3) Suppose $v \in \lambda(v_1) \cap \lambda(v_2)$. Then v is an active predecessor of both v_1 and v_2 . This implies that for all $w \in \text{path}(v_1, v_2)$, v is an active predecessor of w , and hence $v \in \lambda(w)$. \blacktriangleleft

► **Theorem 20.** *Let \mathcal{A} be a finite σ -structure. There is a bijective correspondence between:*

1. \mathbb{P}_k -coalgebras $\alpha : \mathcal{A} \rightarrow \mathbb{P}_k \mathcal{A}$
2. k -pebble forest covers of $\mathcal{G}(\mathcal{A})$.

Proof. See [2, Theorem 6]. \blacktriangleleft

We write $\kappa^{\mathbb{P}}(\mathcal{A})$ for the coalgebra number of \mathcal{A} with respect to the the pebbling comonad.

► **Theorem 21.** *For all finite structures \mathcal{A} : $\text{tw}(\mathcal{A}) = \kappa^{\mathbb{P}}(\mathcal{A}) - 1$.*

4.3 The modal comonad and synchronization tree depth

Let \mathcal{A} be a Kripke structure. It will be convenient to write labelled transitions $a \xrightarrow{\alpha} a'$ for $R_\alpha(a, a')$. Given $a \in A$, the submodel generated by a is obtained by restricting the universe to the set of a' such that there is a path $a \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_k} a'$. This submodel forms a synchronization tree [24] if for all a' , there is a unique such path. The height of such a tree is the maximum length of any path from the root a .

► **Proposition 22.** *Let \mathcal{A} be a Kripke structure, with $a \in A$. The following are equivalent:*

1. *There is a coalgebra $\alpha : (\mathcal{A}, a) \rightarrow \mathbb{M}_k(\mathcal{A}, a)$.*
2. *The submodel generated by a is a synchronization tree of height $\leq k$.*

We define the modal depth $\text{md}(\mathcal{A}, a) = k$ if the submodel generated by a is a synchronization tree of height k .

► **Theorem 23.** *Let \mathcal{A} be a Kripke structure, and $a \in A$ be such that the submodel generated by a is a synchronization tree of finite height. Then $\text{md}(\mathcal{A}, a) = \kappa^{\mathbb{M}}(\mathcal{A}, a)$.*

Note the conditional nature of this result, which contrasts with those for the other comonads. The modal comonad is defined in such a way that the universe $\mathbb{M}_k(A, a)$ reflects information about the possible transitions. Thus having a coalgebra at all, regardless of the value of the resource parameter, is a strong constraint on the structure of the transition system.

5 Further Directions

From the categorical perspective, there is considerable additional structure which we have not needed for the results in this paper, but which may be useful for further investigations.

Coequaliser requirements. In Moggi’s work on computational monads, there is an “equaliser requirement” [26]. The dual version for a comonad (G, ε, δ) is that for every object A , the following diagram is a coequaliser:

$$G^2 A \begin{array}{c} \xrightarrow{G\varepsilon_A} \\ \xrightarrow{\varepsilon_{GA}} \end{array} GA \xrightarrow{\varepsilon_A} A$$

This says in particular that the counit is a regular epi, and hence GA “covers” A in a strong sense.

This coequaliser requirement holds for all our comonads. For \mathbb{E}_k , this is basically the observation that, given a sequence of sequences $[s_1, \dots, s_j]$, we have $\varepsilon[\varepsilon s_1, \dots, \varepsilon s_j] = \varepsilon s_j$. The other cases are similar.

Indexed and graded structure. Our comonads $\mathbb{E}_k, \mathbb{P}_k, \mathbb{M}_k$ are not merely discretely indexed by the resource parameter. In each case, there is a functor $(\mathbb{Z}^+, \leq) \rightarrow \mathbf{Comon}(\mathcal{R}(\sigma))$ from the poset category of the positive integers to the category of comonads on $\mathcal{R}(\sigma)$. Thus if $k \leq l$ there is a natural transformation with components $i_A^{k,l} : \mathbb{E}_k \mathcal{A} \rightarrow \mathbb{E}_l \mathcal{A}$, which preserves the counit and comultiplication; and similarly for the other comonads. Concretely, this is just including the plays of up to k rounds in the plays of up to l rounds, $k \leq l$.

Another way of parameterizing comonads by resource information is grading [12]. Recall that comonads on \mathcal{C} are exactly the comonoids in the strict monoidal category $([\mathcal{C}, \mathcal{C}], \circ, I)$ of endofunctors on \mathcal{C} [22]. Generalizing this description, a graded comonad is an oplax monoidal functor $G : (M, \cdot, 1) \rightarrow ([\mathcal{C}, \mathcal{C}], \circ, I)$ from a monoid of grades into this endofunctor category. This means that for each $m \in M$, there is an endofunctor G_m , there is a graded counit natural transformation $\varepsilon : G_1 \Rightarrow I$, and for all $m, m' \in M$, there is a graded comultiplication $\delta^{m,m'} : G_{m \cdot m'} \Rightarrow G_m G_{m'}$.

The two notions can obviously be combined. We can see our comonads as (trivially) graded, by viewing them as oplax monoidal functors $(\mathbb{Z}^+, \leq, \min, 1) \rightarrow ([\mathcal{C}, \mathcal{C}], \circ, I)$. Given $k \leq l$, we have e.g. $\mathbb{E}_k \Rightarrow \mathbb{E}_k \mathbb{E}_k \Rightarrow \mathbb{E}_k \mathbb{E}_l$. The question is whether there are more interesting graded structures which arise naturally in considering richer logical and computational settings.

Colimits and infinite behaviour. In this paper, we have dealt exclusively with finite resource levels. However, there is an elegant means of passing to infinite levels. We shall illustrate this with the modal comonad. Using the inclusion morphisms described in the previous discussion of indexed structure, for each structure \mathcal{A} we have a diagram

$$\mathbb{M}_1 \mathcal{A} \rightarrow \mathbb{M}_2 \mathcal{A} \rightarrow \dots \rightarrow \mathbb{M}_k \mathcal{A} \rightarrow \dots$$

By taking the colimits of these diagrams, we obtain a comonad \mathbb{M}_ω , which corresponds to the usual unfolding of a Kripke structure to all finite levels. This will correspond to the bisimulation approximant \sim_ω , which coincides with bisimulation itself on image-finite structures [17]. Transfinite extensions are also possible. Similar constructions can be applied to the other comonads. This provides a basis for lifting the comonadic analysis to the level of infinite models.

Relations between fragments and parameters. We can define morphisms between the different comonads we have discussed, which yield proofs about the relationships between the logical fragments they characterize. This categorical perspective avoids the cumbersome syntactic translations in the standard proofs of these results. For illustration, there is a comonad morphism $t : \mathbb{E}_k \Rightarrow \mathbb{P}_k$ with components $t_A : \mathbb{E}_k \mathcal{A} \rightarrow \mathbb{P}_k \mathcal{A}$ given by $[a_1, \dots, a_j] \mapsto [(1, a_1), \dots, (j, a_j)]$. Together with theorems 13 and 12, this shows that $\exists \mathcal{L}_k \subseteq \exists \mathcal{L}^k$ and $\mathcal{L}_k(\#) \subseteq \mathcal{L}^k(\#)$. Moreover, composing t with a coalgebra $\mathcal{A} \rightarrow \mathbb{E}_k \mathcal{A}$ yields a coalgebra $\mathcal{A} \rightarrow \mathbb{P}_k \mathcal{A}$, demonstrating that $\text{tw}(\mathcal{A}) + 1 \leq \text{td}(\mathcal{A})$. Another morphism $\mathbb{M}_\omega \Rightarrow \mathbb{P}_2$ shows that modal logic can be embedded into 2-variable logic.

Concluding remarks

Our comonadic constructions for the three major forms of model comparison games show a striking unity, on the one hand, but also some very interesting differences. For the latter, we note the different forms of logical “deception” associated with each comonad, the different forms of back-and-forth equivalences, and the different combinatorial parameters which arise in each case.

One clear direction for future work is to gain a deeper understanding of what makes these constructions work. Another is to understand how widely the comonadic analysis of resources can be applied. We are currently investigating the guarded fragment [4, 14]; other natural candidates include existential second-order logic, and branching quantifiers and dependence logic [32].

Since comonads arise naturally in type theory and functional programming [31, 28], can we connect the study of finite model theory made here with a suitable type theory? Can this lead, via the Curry-Howard correspondence, to the systematic derivation of some significant meta-algorithms, such as decision procedures for guarded logics based on the tree model property [13], or algorithmic metatheorems such as Courcelle’s theorem [9]?

Another intriguing direction is to connect these ideas with the graded quantum monad studied in [1], which provides a basis for the study of quantum advantage in $\mathcal{R}(\sigma)$. This may lead to a form of quantum finite model theory.

References

- 1 Samson Abramsky, Rui Soares Barbosa, Nadish de Silva, and Octavio Zapata. The quantum monad on relational structures. *To appear in proceedings of MFCS 2017*, 2018.
- 2 Samson Abramsky, Anuj Dawar, and Pengming Wang. The pebbling comonad in finite model theory. In *Logic in Computer Science (LICS), 2017 32nd Annual ACM/IEEE Symposium on*, pages 1–12. IEEE, 2017.
- 3 Luca Aceto, Anna Ingólfssdóttir, and Joshua Sack. Resource bisimilarity and graded bisimilarity coincide. *Information Processing Letters*, 111(2):68–76, 2010.
- 4 Hajnal Andréka, István Németi, and Johan van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

- 5 Patrick Blackburn, Maarten De Rijke, and Yde Venema. *Modal Logic*, volume 53. Cambridge University Press, 2002.
- 6 E. Casanovas, P. Dellunde, and R. Jansana. On Elementary Equivalence for Equality-free Logic. *Notre Dame Journal of Formal Logic*, 37(3):506–522, 1996. doi:10.1305/ndjfl/1039886524.
- 7 Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90. ACM, 1977.
- 8 Flavio Corradini, Rocco De Nicola, and Anna Labella. Graded modalities and resource bisimulation. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 381–393. Springer, 1999.
- 9 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.
- 10 M. de Rijke. A Note on Graded Modal Logic. *Studia Logica*, 64(2):271–283, 2000.
- 11 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Springer Science & Business Media, 2005.
- 12 Marco Gaboardi, Shin-ya Katsumata, Dominic Orchard, Flavien Breuvert, and Tarmo Uustalu. Combining effects and coeffects via grading. *ACM SIGPLAN Notices*, 51(9):476–489, 2016.
- 13 Erich Grädel. Decision procedures for guarded logics. In *International Conference on Automated Deduction*, pages 31–51. Springer, 1999.
- 14 Erich Grädel and Martin Otto. The freedoms of (guarded) bisimulation. In *Johan van Benthem on Logic and Information Dynamics*, pages 3–31. Springer, 2014.
- 15 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*. Oxford University Press, 2004.
- 16 L. Hella. Logical hierarchies in PTIME. *Information and Computation*, 121:1–19, 1996.
- 17 Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In *International Colloquium on Automata, Languages, and Programming*, pages 299–309. Springer, 1980.
- 18 Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.
- 19 Phokion G Kolaitis and Moshe Y Vardi. On the expressive power of Datalog: tools and a case study. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 61–71. ACM, 1990.
- 20 Phokion G Kolaitis and Moshe Y Vardi. Infinitary logics and 0–1 laws. *Information and Computation*, 98(2):258–294, 1992.
- 21 Leonid Libkin. *Elements of Finite Model Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer, 2004.
- 22 Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.
- 23 Ernest G Manes. *Algebraic Theories*, volume 26. Springer Science & Business Media, 2012.
- 24 Robin Milner. *A calculus of communicating systems*. Number 92 in Lecture Notes in Comput. Science. Springer-Verlag, 1980.
- 25 Robin Milner. *Communication and concurrency*, volume 84. Prentice Hall New York etc., 1989.
- 26 Eugenio Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991.
- 27 Jaroslav Nešetřil and Patrice Ossona De Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006.

- 28 Dominic Orchard. Programming contextual computations. Technical Report UCAM-CL-TR-854, University of Cambridge, 2014.
- 29 Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31(4):15, 2009.
- 30 Saharon Shelah. Every two elementarily equivalent models have isomorphic ultrapowers. *Israel Journal of Mathematics*, 10(2):224–233, 1971.
- 31 Tarmo Uustalu and Varmo Vene. Comonadic notions of computation. *Electronic Notes in Theoretical Computer Science*, 203(5):263–284, 2008.
- 32 Jouko Väänänen. *Dependence logic: A new approach to independence friendly logic*, volume 70. Cambridge University Press, 2007.

Climbing up the Elementary Complexity Classes with Theories of Automatic Structures

Faried Abu Zaid

TU Ilmenau, Germany

Dietrich Kuske

TU Ilmenau, Germany

Peter Lindner

RWTH Aachen University, Germany

Abstract

Automatic structures are structures that admit a finite presentation via automata. Their most prominent feature is that their theories are decidable. In the literature, one finds automatic structures with non-elementary theory (e.g., the complete binary tree with equal-level predicate) and automatic structures whose theories are at most 3-fold exponential (e.g., Presburger arithmetic or infinite automatic graphs of bounded degree). This observation led Durand-Gasselin to the question whether there are automatic structures of arbitrary high elementary complexity.

We give a positive answer to this question. Namely, we show that for every $h \geq 0$ the forest of (infinitely many copies of) all finite trees of height at most $h + 2$ is automatic and its theory is complete for $\text{STA}(*, \exp_h(n, \text{poly}(n)), \text{poly}(n))$, an alternating complexity class between h -fold exponential time and space. This exact determination of the complexity of the theory of these forests might be of independent interest.

2012 ACM Subject Classification Theory of computation \rightarrow Complexity theory and logic

Keywords and phrases Automatic Structures, Complexity Theory, Model Theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.3

1 Introduction

The idea of an automatic structure goes back to Büchi and Elgot who used finite automata to decide, e.g., Presburger arithmetic [6]. In essence, a structure is automatic if the elements of the universe are strings from a regular language and every relation of the structure is synchronously-rational [11]. The notion was introduced in [13] and a systematic study was initiated by Khoushainov and Nerode [15] and started to attract quite some interest with the work by Blumensath and Grädel [3, 4], see the surveys [23, 1, 24, 14]. One of the main motivations for investigating automatic structures is that their first-order theories are decidable. This decidability holds even if one extends first-order logic by quantifiers “there exist infinitely many” [3], “the number of elements satisfying φ is a finite multiple of p ” [16], and “there exists an infinite relation satisfying φ ” (provided φ mentions the infinite relation only negatively) [19].

Already in [3, 4], the authors observe that the first-order theory of an automatic structure is, in general, non-elementary (i.e., does not belong to n -EXPSPACE for any $n \in \mathbb{N}$). The simplest example is provided by the set of binary words with the prefix relation, the two successor relations, and the equal-length predicate. An inspection of the decidability proof for arbitrary automatic structures shows that validity of a formula in Σ_{n+1} can be decided in n -EXPSPACE. Note that this problem has two inputs: a formula from Σ_{n+1} and an



© Faried Abu Zaid, Dietrich Kuske, and Peter Lindner;
licensed under Creative Commons License CC-BY

27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 3; pp. 3:1–3:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

automatic structure (given by a tuple of automata). In [18], it is shown that fixing one of the two inputs does not make the problem simpler. In other words: both the expression and the data complexity are complete for n -EXPSPACE.

On the positive side, there are also automatic structures whose theories are much simpler. One example is Presburger's arithmetic, i.e., the structure $(\mathbb{N}, +)$ that is automatic [6] and has a theory in 2-EXPSPACE [22, 8]. Another example are automatic structures of bounded degree [20] whose theories are in 2-EXPSPACE. Finally, let us mention structures, which have an automatic presentation over a unary alphabet, e.g. the natural Numbers with successor (\mathbb{N}, S) . The first-order theory of every such structure is decidable in polynomial time [17].

To the authors' knowledge, no automatic structure is known whose theory is elementary but not in 2-EXPSPACE. In this article, we provide such examples. More precisely, for any $h \in \mathbb{N}$, we provide an automatic structure whose theory is complete for the class of problems that can be decided in h -fold exponential time with polynomially many alternations, i.e., for Berman's complexity class $\text{STA}(*, \exp_h(2), \text{poly}(n), \text{poly}(n))$ [2].

This structure is the forest F_{h+2} consisting of countably many copies of all trees of height at most $h + 2$. Containment in $\text{STA}(*, \exp_h(2), \text{poly}(n), \text{poly}(n))$ is shown as follows: Let φ be a first-order sentence of quantifier rank r . In a first step, we show that any tree of height $\leq h + 2$ is indistinguishable from some tree of size h -fold exponential in r by any formula of quantifier rank r . Consequently, to determine the truth of the sentence φ in the forest F_{h+2} , it suffices to determine it in a forest whose trees have size h -fold exponential in r . Since the elements of this forest can be described by words of h -fold exponential size, its model checking can be done in the said complexity class.

For the lower bound, we first reduce any problem in the said complexity class to the theory of the free monoid where quantification is restricted to words of h -fold exponential length. This theory is then reduced to the theory of the forest F_{h+2} . This second step is based on an encoding of h -fold exponential numbers and their addition in the forest.

Thus, technically, the main result of this paper is the complete characterisation of the complexity of the theory of the forest F_{h+2} . Since this forest is automatic, we get an affirmative answer to the open question from the theory of automatic structures. Besides this, the forest F_{h+2} is a natural structure, so that our result can have consequences in other contexts as well.

The results presented in this paper close the gap that was left open in the third author's master thesis [21].

2 Preliminaries

The set of natural numbers is denoted $\mathbb{N} = \{0, 1, 2, \dots\}$; $\mathbb{N}_{>0} = \{1, 2, 3, \dots\}$ denotes the positive natural numbers. For $m, n, r \in \mathbb{N}$ we write $m =_r n$ if $m = n$ or $m, n \geq r$. Inductively, we define the class of functions $\exp_m: \mathbb{N}^2 \rightarrow \mathbb{N}$ for $m, c, n \in \mathbb{N}$:

$$\exp_m(c, n) = \begin{cases} n & \text{if } m = 0 \\ c^{\exp_{m-1}(c, n)} & \text{if } m > 0 \end{cases}$$

Intuitively, $\exp_m(c, n)$ is a stack of c s of height m with the number n on top of this stack. By $\text{poly}(n)$ we denote the class of all polynomial functions $\mathbb{N} \rightarrow \mathbb{N}$.

We assume that the reader is familiar with the basics of automata theory and formal logic, especially first-order logic. We use this section to recall some of the key notions in order to fix our notation.

A (*directed*) graph is a tuple $G = (V, E)$, where V is a set and $E \subseteq V \times V \setminus \{(v, v) \mid v \in V\}$ is a binary irreflexive relation. A *tree* is a finite graph $T = (V, E)$ such that, for some node $r \in V$, any node $v \in V$ has precisely one path from r to v . The node r , being unique, is called the *root* of T . Now let $T = (V, E)$ be a tree and $v \in V$. The *depth* of v is the length of the path from r to v (i.e., the number of edges such that the depth of the root is 0). The *height* of v is the maximal length of a path starting in v . A node v is a *leaf* if its height is 0. The *height* of T is the height of the root r or, equivalently, the maximal depth of a node in T . A subtree is an induced subgraph of a tree $T = (V, E)$ whose vertex set is of the form $\{w \in V \mid w \text{ is reachable from } v\}$ for some node $v \in V$. Note that v is the root of this subtree and every subtree is uniquely determined by its root. Therefore we denote the subtree with root v by T_v .

An *automatic graph* is a graph $G = (V, E)$ such that $V \subseteq \Sigma^*$ is a regular language over some alphabet Σ and the edge relation E is synchronously rational [11].

First-order formulas (over the language of graphs) are built up from variables $\{x_i \mid i \in \mathbb{N}\}$, the Boolean connectives $\{\neg, \vee, \wedge, \rightarrow\}$, the edge relation symbol E , quantifiers $\{\forall, \exists\}$, and the bracket symbols $\{(,)\}$. The *quantifier rank* $\text{qr}(\varphi)$ of a formula φ is the maximal nesting depth of quantifiers within φ . Two graphs G and H are *r-equivalent* (denoted $G \equiv_r H$) if they cannot be distinguished by any formula of quantifier rank $\leq r$. For a tuple $\bar{a} = (a_1, \dots, a_k) \in A^k$ and $B \subseteq A$ let $\bar{a}|_B$ denote the restriction of \bar{a} to the components in B , i.e. the tuple $(a_{i_1}, \dots, a_{i_\ell})$ with $\{i_1, \dots, i_\ell\} = \{i \mid a_i \in B\}$ and $i_1 < i_2 < \dots < i_\ell$.

The *Ehrenfeucht-Fraïssé-game* is a game-theoretic characterisation of elementary equivalence. It is played on two graphs G and H , where the two players, Spoiler and Duplicator, choose alternately elements of these two structures for a prescribed number of rounds. More precisely the i -th round of an r -round Ehrenfeucht-Fraïssé-game on $G = (V^G, E^G)$ and $H = (V^H, E^H)$ ($G_r(G, H)$) has the following form: First Spoiler picks an element a_i from G or an element b_i from H . Duplicator answers by choosing an element b_i from H or an element a_i from G , respectively. Therefore the two players iteratively construct two tuples $(a_1, \dots, a_r) \in (V^G)^r$ and $(b_1, \dots, b_r) \in (V^H)^r$. Duplicator wins if the mapping $a_i \mapsto b_i$ is a partial isomorphism, that is if $a_i = a_j \Leftrightarrow b_i = b_j$ and $(a_i, a_j) \in E^G \Leftrightarrow (b_i, b_j) \in E^H$ for all $1 \leq i, j \leq r$. Otherwise Spoiler wins.

► **Theorem 1** ([5]). *Let G and H be two graphs. Then Duplicator has a winning strategy in the game $G_r(G, H)$ if, and only if, $G \equiv_r H$.*

The main object of study in this paper is the following forest:

► **Definition 2.** For $H \in \mathbb{N}$, let F_H denote the disjoint union of \aleph_0 many copies of all trees of height at most H .

Thus, F_H is the forest of all trees of height at most H , containing countably many copies of every such tree.

► **Remark 3.** *Natural variants of this forest are, among others, the following:*

- *The disjoint union F_H^∞ of \aleph_0 many copies of all countably infinite (or at most countably infinite) trees of height at most H .*
- *The disjoint union F_H^1 of all finite (or at most countably infinite) trees of height at most H up to isomorphism (i.e., one tree per isomorphism class).*

We will show that F_H is automatic which is not the case for F_H^∞ (it is ω -automatic) and we conjecture that also F_H^1 is not automatic.

Nevertheless, the proofs of the complexity results can easily be transformed to show that also the theories of these forests are complete for $\text{STA}(, \exp_{H-2}(2, \text{poly}(n)), \text{poly}(n))$.*

3 Trees of Bounded Height

The goal of this section is to provide an automatic copy of the forest F_H for every $H \in \mathbb{N}$. The idea is to use XML-like notation to describe a tree and to encode an element by marking its position in the tree that it belongs to. Because the nesting-depth of parentheses will be bounded for every H , the resulting languages remain regular. Let $\Sigma = \{\langle \rangle, \langle \rangle, \langle x \rangle, \langle \backslash x \rangle\}$. We define regular languages J_H and K_H for every $H \in \mathbb{N}$:

$$\begin{aligned} J_0 &= \{\langle \rangle \langle \backslash \rangle\} \\ K_0 &= \{\langle x \rangle \langle \backslash x \rangle\} \end{aligned}$$

and

$$\begin{aligned} J_{H+1} &= \langle \rangle J_H^* \langle \backslash \rangle \\ K_{H+1} &= \langle x \rangle J_H^* \langle \backslash x \rangle \cup \langle \rangle J_H^* K_H J_H^* \langle \backslash \rangle. \end{aligned}$$

Every word in $w \in K_H$ contains the tag $\langle x \rangle \dots \langle \backslash x \rangle$ exactly once. This tag marks the selected node in the tree that is presented by w .

Next we show that the edge relation on K_H is synchronously-rational [11]. Two nodes u and v from F_H are connected by a directed edge if, and only if, they belong to the same tree and u is the parent of v . To describe the edge relation E_H of our automatic copy, write $L^{\odot 2} = \left\{ \binom{w}{w} : w \in L \right\}$ for any language L . Then we have

$$\begin{aligned} E_0 &= \emptyset \\ E_1 &= \binom{\langle x \rangle}{\langle \rangle} \binom{\langle \rangle \langle \backslash \rangle}{\langle \rangle \langle \backslash \rangle}^* \binom{\langle \rangle \langle \backslash \rangle}{\langle x \rangle \langle \backslash x \rangle} \binom{\langle \rangle \langle \backslash \rangle}{\langle \rangle \langle \backslash \rangle}^* \binom{\langle \backslash x \rangle}{\langle \backslash \rangle} \\ E_{H+2} &= \binom{\langle x \rangle}{\langle \rangle} (J_{H+1}^{\odot 2})^* \binom{\langle \rangle}{\langle x \rangle} (J_H^{\odot 2})^* \binom{\langle \backslash \rangle}{\langle \backslash x \rangle} (J_{H+1}^{\odot 2})^* \binom{\langle \backslash x \rangle}{\langle \backslash \rangle} \\ &\quad \cup \binom{\langle \rangle}{\langle \rangle} (J_{H+1}^{\odot 2})^* E_{H+1} (J_{H+1}^{\odot 2})^* \binom{\langle \backslash \rangle}{\langle \backslash \rangle}. \end{aligned}$$

Note that the languages that we defined so far do not induce an isomorphic copy of F_H . We need to modify the languages such that every tree of height at most H will appear infinitely often. Therefore let $L_H = \$^* K_H$ and $E'_H = \binom{\$}{\$}^* E_H$. Then $(L_H, E'_H) \cong F_H$ is an automatic copy of F_H .

4 Upper Bound

We provide a simple decision procedure for the theory of F_{H+2} that runs in alternating H -fold exponential time while making only polynomially many alternations. We found it more convenient to first prove this result in the realm of order trees: An *order tree* is a finite partial order (V, \leq) with a minimal element such that, for any $v \in V$, the set $\{w \in V \mid w \leq v\}$ is finite and linearly ordered by \leq . An *order forest* is a disjoint union of order trees. The *length* of an order forest is the maximal size of a linearly ordered subset, its *height* is the predecessor of its length.

Let oF_h denote the order version of the forest F_h , i.e., the disjoint union of infinitely many copies of any order tree of height $\leq H$. The theory of this order forest can be decided as follows: We determine from a sentence φ of quantifier rank r a finite order forest satisfying

φ iff $\text{oF}_{H+2} \models \varphi$. The size of this order forest can be bounded since, as we show below, every finite order tree of height $\leq H + 2$ is r -equivalent to an order tree of size at most $\exp_{H+1}(r + 1, \text{poly}(n + 1))$. The elements of this finite order forest have encodings by words of length $\leq \exp_{H+1}(r + 1, \text{poly}(n + 1))$. Then, the standard alternating model checking algorithm is applied to this forest (without computing it explicitly). The result on the forest F_{H+2} follows because of a polynomial-time reduction of the theory of the forest F_{H+2} to that of the order forest oF_{H+2} .

The following lemma on order forests prepares the construction of a “small” equivalent order tree.

► **Lemma 4.** *Let $(S_i)_{i \in I}$ and $(T_j)_{j \in J}$ be nonempty (possibly infinite) families of order trees such that*

$$|\{i \in I \mid S_i \in \tau\}| =_r |\{j \in J \mid T_j \in \tau\}| \quad (1)$$

holds for any \equiv_r -equivalence class τ . Then

$$\biguplus_{i \in I} S_i \equiv_r \biguplus_{j \in J} T_j. \quad (2)$$

Proof. We show that Duplicator has a winning strategy in the r -round Ehrenfeucht-Fraïssé-game on the forests $S = \biguplus_{i \in I} S_i$ and $T = \biguplus_{j \in J} T_j$. More precisely we show that Duplicator can maintain the following invariant after $\ell \in \{0, 1, \dots, r\}$ rounds (when the current position is (\bar{a}, \bar{b})):

$$\text{For all } i \in I, \text{ there exists } j \in J \text{ such that for all } k \in \{1, 2, \dots, \ell\}, \text{ we have } a_k \in S_i \iff b_k \in T_j \text{ and } (S_i, \bar{a}|_{S_i}) \equiv_{r-\ell} (T_j, \bar{b}|_{T_j}).$$

Since no edge connects distinct trees in a forest, every position $(a_1, \dots, a_r, b_1, \dots, b_r)$ satisfying this invariant describes a partial isomorphism $a_i \mapsto b_i$. Therefore it remains to be shown that Duplicator can maintain this invariant.

So let $0 \leq \ell < r$, $a_1, \dots, a_\ell \in S$, and $b_1, \dots, b_\ell \in T$ such that the invariant holds. Note that the invariant is equivalent to its dual:

$$\text{For all } j \in J, \text{ there exists } i \in I \text{ such that for all } k \in \{1, 2, \dots, \ell\}, \text{ we have } b_k \in T_j \iff a_k \in S_i \text{ and } (T_j, \bar{b}|_{T_j}) \equiv_{r-\ell} (S_i, \bar{a}|_{S_i}).$$

Hence, by symmetry, we can assume that Spoiler chooses an element $a_{\ell+1}$ of S in round $\ell + 1 \leq r$. Then there is $i \in I$ such that $a_{\ell+1}$ is a node from S_i . We distinguish two cases: either there is $k \in \{1, 2, \dots, \ell\}$ with $a_k \in S_i$ or there is no such k .

First, assume $a_k \in S_i$ for some $1 \leq k \leq \ell$. By the induction hypothesis, there exists $j \in J$ with $b_k \in T_j$ and $(S_i, \bar{a}|_{S_i}) \equiv_{r-\ell} (T_j, \bar{b}|_{T_j})$. Hence, there is $b_{\ell+1} \in T_j$ with $(S_i, \bar{a}a_{\ell+1}|_{S_i}) \equiv_{r-\ell-1} (T_j, \bar{b}b_{\ell+1}|_{T_j})$. Choosing this element $b_{\ell+1}$, Duplicator can move the play into a position that satisfies the invariant.

Now consider the second case, $a_k \notin S_i$ for all $1 \leq k \leq \ell$. Let $I' = \{i' \in I \mid S_i \equiv_r S_{i'}\}$ and, similarly, $J' = \{j' \in J \mid S_i \equiv_r T_{j'}\}$. If $|I'| = |J'|$, the invariant implies the existence of $j \in J'$ such that no element b_k belongs to T_j . Otherwise, we have $|J'| \geq r$ by (1). Since only $\ell < r$ many nodes b_k have been chosen so far, also in this case there exists $j \in J'$ such that no element b_k belongs to T_j . Because of $S_i \equiv_r T_j$, the tree T_j has some element $b_{\ell+1}$ with $(S_i, a_{\ell+1}) \equiv_{r-1} (T_j, b_{\ell+1})$ (and therefore also $(S_i, a_{\ell+1}) \equiv_{r-\ell-1} (T_j, b_{\ell+1})$). Thus, also in this case, Duplicator can move the play into a position that satisfies the invariant. ◀

► **Lemma 5.** *Let $r, h \in \mathbb{N}$. There exists a polynomial function $p_h: \mathbb{N} \rightarrow \mathbb{N}$ such that the following holds: For any order tree S of height $\leq h$, there exists an \equiv_r -equivalent order tree T of height $\leq h$ and size*

$$\leq \begin{cases} p_h(r+1) & \text{if } h \leq 2 \\ \exp_{h-2}(r+1, p_h(r+1)) & \text{if } h > 2. \end{cases}$$

Proof. For each $h, r \in \mathbb{N}$, we let \equiv_r^h denote the restriction of the relation \equiv_r to order trees of height $\leq h$.

By induction on h , we prove in addition

$$\text{index}(\equiv_r^h) \leq \begin{cases} 1 & \text{if } h = 0 \\ \exp_{h-1}(r+1, r+1) & \text{if } h \geq 1. \end{cases}$$

For $h = 0$, there is only one order tree of height h and this tree has size 1, hence we set $p_0(x) = 1$. Furthermore, $\text{index}(\equiv_r^0) = 1$ is obvious.

Now let $h > 0$ and let S be some order tree of height h . Let I denote the set of nodes of depth 1 and, for $i \in I$, let S_i denote the subtree of S rooted at i . By the induction hypothesis, any \equiv_r^{h-1} -equivalence class τ contains some order tree T_τ of size $\leq p_{h-1}(r+1)$ (if $h \leq 3$) and $\leq \exp_{h-3}(r+1, p_{h-1}(r+1))$ otherwise. For $i \in I$, let $T_i = T_{[S_i]}$ be the representative of the \equiv_r^{h-1} -class of S_i . Let $J \subseteq I$ such that

$$\min(r, |\{i \in I \mid S_i \in \tau\}|) = |\{j \in J \mid T_j \in \tau\}|$$

for any \equiv_r -equivalence class τ . Then (1) from Lemma 4 holds, implying $\biguplus_{i \in I} S_i \equiv_r \biguplus_{j \in J} T_j$ by Lemma 4. Let the order tree T arise from the order forest $\biguplus_{j \in J} T_j$ by the addition of a root that is smaller than any other node. Note that T is quantifier free definable in the disjoint sum of $\biguplus_{j \in J} T_j$ and a single node.¹ Since S arises in the same way from the order forest $\biguplus_{i \in I} S_i$, we get $S \equiv_r T$ [7].

Next, we prove the upper bound for the size of the order tree T . Note that this size is at most $|J|$ multiplied with the maximal size of an order tree T_j . Since J contains at most r elements per \equiv_r^{h-1} -equivalence class, we obtain

$$\begin{aligned} |J| &\leq r \cdot \text{index}(\equiv_r^{h-1}) \\ &\leq r \cdot \begin{cases} 1 & \text{if } h = 1 \\ r+1 & \text{if } h = 2 \\ \exp_{h-2}(r+1, r+1) & \text{if } h \geq 3. \end{cases} \end{aligned}$$

Since the size of the order trees T_j is bounded as described above, the size of the order tree T is

$$\begin{aligned} &\leq r \cdot \begin{cases} 1 \cdot p_0(r+1) & \text{if } h = 1 \\ (r+1) \cdot p_1(r+1) & \text{if } h = 2 \\ \exp_1(r+1, r+1) \cdot p_2(r+1) & \text{if } h = 3 \\ \exp_{h-2}(r+1, r+1) \cdot \exp_{h-3}(r+1, p_{h-1}(r+1)) & \text{if } h > 3 \end{cases} \\ &\leq \begin{cases} p_h(r+1) & \text{if } h \leq 2 \\ \exp_{h-2}(r+1, p_h(r+1)) & \text{if } h \geq 3 \end{cases} \end{aligned}$$

for a suitably chosen polynomial function p_h . This proves the claim from the lemma.

¹ Here we need order trees since this does not hold for successor trees (V, E) .

It remains to prove the additional inductive invariant on the number of equivalence classes of \equiv_r^h . Note that the order tree T constructed above is completely given by a mapping from the \equiv_r^{h-1} -equivalence classes into the set of numbers $\{0, 1, \dots, r\}$. Hence, the number of distinct order trees T that can arise in the above way, is

$$\begin{aligned} &\leq (r+1)^{\text{index}(\equiv_r^{h-1})} \\ &\leq \begin{cases} (r+1) & \text{if } h = 1 \\ (r+1)^{\text{exp}_{h-2}(r+1, r+1)} & \text{if } h > 1 \end{cases} \\ &= \text{exp}_{h-1}(r+1, r+1). \end{aligned} \quad \blacktriangleleft$$

For $r, k \in \mathbb{N}$, we let $\text{oF}_h^{r,k}$ denote the disjoint union of r copies of every order tree of height $\leq h$ and size $\leq k$.

► **Proposition 6.** *Let $r, h \in \mathbb{N}$. There exists a polynomial function $p_h: \mathbb{N} \rightarrow \mathbb{N}$ such that $\text{oF}_h \equiv_r \text{oF}_h^{r,k}$ with*

$$k = \begin{cases} p_h(r+1) & \text{if } h \leq 2 \\ \text{exp}_{h-2}(r+1, p_h(r+1)) & \text{if } h > 2. \end{cases}$$

Proof. Let τ be some \equiv_r -equivalence class containing some order tree S of height $\leq h$. The order forest of oF_h contains infinitely many copies of S . By Lemma 5, there exists an order tree T in $\text{oF}_h^{r,k}$ with $T \in \tau$. More precisely, there are $\geq r$ such order trees (possibly isomorphic). From Lemma 4, we obtain $F_h \equiv_r F_{r,k}^h$. ◀

► **Corollary 7.** *For $H \in \mathbb{N}$, the theory of oF_{H+2} belongs to $\text{STA}(*, \text{exp}_H(2, \text{poly}(n)), \text{poly}(n))$.*

Proof. Let φ be a sentence of size n . Without loss of generality, we assume φ to be in prenex normal form. Let furthermore p be the polynomial p_{H+2} from Proposition 6.

The quantifier rank of φ is $\leq n$. Hence, by Proposition 6, it suffices to decide whether φ holds in the finite order forest $\text{oF}_{H+2}^{n,k}$ with $k = \text{exp}_H(r+1, p(r+1))$. Using the encoding of F_{H+2} as automatic structure, the elements of $\text{oF}_{H+2}^{n,k}$ can be encoded as strings of length $\mathcal{O}(n+k)$. Hence the standard alternating model-checking algorithm for first-order logic uses time $\mathcal{O}(\text{poly}(n+k))$ and $\leq n$ alternations. Note that this algorithm does not calculate the order forest $\text{oF}_{H+2}^{n,k}$ explicitly, but only handles words of length $\mathcal{O}(n+k)$. ◀

As a consequence, we get the following result about the forest F_{H+2} .

► **Theorem 8.** *For $H \in \mathbb{N}$, the theory of F_{H+2} belongs to $\text{STA}(*, \text{exp}_H(2, \text{poly}(n)), \text{poly}(n))$.*

Proof. We reduce this theory to the theory of the ordered forest oF_{H+2} : Let φ be a sentence in the signature of trees. In φ , replace every occurrence of the atomic formula $E(x, y)$ by

$$x < y \wedge \neg \exists z: x < z < y$$

and call the resulting sentence φ' . Then $F_{H+2} \models \varphi \iff \text{oF}_{H+2} \models \varphi'$. Since φ' can be computed from φ in polynomial time, the claim follows from Corollary 7. ◀

5 Lower Bound

Let $H \geq 1$ be fixed throughout this section. We want to show that the theory of the forest F_{H+2} is hard for the class $\text{STA}(*, \exp_H(2, \text{poly}(n)), \text{poly}(n))$.

We will reduce an arbitrary language $L \subseteq \Sigma^*$ from the said complexity class to the theory of the forest F_{H+2} in two steps: First, we reduce L to the theory of the free monoid Δ^* . In this reduction, we can restrict quantification to words of length $\leq \exp_H(2, \text{poly}(|x|))$. In a second step, we reduce this bounded theory of the free monoid to the theory of the forest F_{H+2} .

Let φ be a formula and $k \geq 1$. Then $\exists^{\geq k} y: \varphi$ abbreviates the formula

$$\exists y_1, y_2, \dots, y_k: \bigwedge_{1 \leq i < j \leq k} y_i \neq y_j \wedge \forall y: \left(\left(\bigvee_{1 \leq i \leq k} y = y_i \right) \rightarrow \varphi \right)$$

and $\exists^=k y: \varphi$ stands for $\exists^{\geq k} y \varphi \wedge \neg \exists^{\geq k+1} y \varphi$. Note that the size of these formulas is $O(k^2 + |\varphi|)$.

5.1 Reduction to the theory of the bounded free monoid

Let $N \geq 0$ and let Δ be an alphabet. The N -bounded free monoid is the structure

$$(\Delta^{\leq \exp_H(N, N)}, \cdot, (a)_{a \in \Delta})$$

where $\Delta^{\leq \exp_H(N, N)}$ is the set of words over Δ of length $\leq \exp_H(N, N)$, \cdot is the concatenation of such words (considered as a ternary relation such that the product of two “long” words is not defined), and any letter $a \in \Delta$ serves as a constant.

An alternating Turing machine is a tuple $M = (Q, \Sigma, \Gamma, \delta, \iota, \square, \text{tp}, F)$ where Q is the finite set of states, $\Sigma \subseteq \Gamma$ are the input- and tape-alphabets, $\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{-1, 0, 1\}$ is the transition relation, $\iota \in Q$ is the initial state, $\square \in \Gamma \setminus \Sigma$ is the blank symbol, $\text{tp}: Q \rightarrow \{\forall, \exists\}$ is the type function with $\text{tp}(\iota) = \exists$, and $F \subseteq Q$ is the set of final states. We assume the tape of M to be infinite on the right, only. We write Δ for the set $\Gamma \cup Q \cup \{\triangleleft, \triangleright\}$ (assuming these three sets to be mutually disjoint).

A *configuration* is a word from $\triangleright \Gamma^* Q \Gamma^* \triangleleft$. We write $c \vdash c'$ for configurations c and c' if the machine can move from c to c' in one step. The *type* of a configuration is the type of its state. A *computation* is a finite sequence of configurations $(c_i)_{0 \leq i \leq n}$ for some $n \in \mathbb{N}$ with $c_i \vdash c_{i+1}$ for all $0 \leq i < n$. We say that it is a computation from c_0 to c_n . It is *existential* if all configurations are existential; it is *homogeneous* if

- the types of c_0, c_1, \dots, c_{n-1} are the same and
- the types of c_0 and c_n are different.

For configurations c and c' , we write

$$c \vdash_{\exists} c' \text{ and } c \vdash_{\text{hom}} c'$$

if there exists an existential and a homogeneous computation, respectively, from c to c' . Note that the latter implies that c and c' have distinct types.

Let $f: \mathbb{N} \rightarrow \mathbb{N}$ be a function. The alternating Turing machine is $f(n)$ -time bounded if any computation $(c_i)_{0 \leq i \leq N}$ with first configuration in $\triangleright \iota w \square^* \triangleleft$ and $w \in \Sigma^*$ makes $\leq f(|w|)$ steps, i.e., satisfies $N + 1 \leq f(|w|)$.

Now let $a \in \mathbb{N}$ be odd and $w \in \Sigma^*$. Then x is accepted by M with a alternations if there exists a configuration $c_0 \in \triangleright \iota w \square^* \triangleleft$ such that the following holds:

$$\begin{aligned}
& \exists \text{ configuration } c_1 \text{ with } c_0 \vdash_{\text{hom}} c_1 \\
& \quad \forall \text{ configurations } c_2 \text{ with } c_1 \vdash_{\text{hom}} c_2 \\
& \quad \quad \exists \text{ configuration } c_3 \text{ with } c_2 \vdash_{\text{hom}} c_3 \\
& \quad \quad \quad \forall \text{ configurations } c_4 \text{ with } c_3 \vdash_{\text{hom}} c_4 \\
& \quad \quad \quad \quad \vdots \\
& \quad \quad \quad \quad \quad \exists \text{ configuration } c_{a-2} \text{ with } c_{a-3} \vdash_{\text{hom}} c_{a-2} \\
& \quad \quad \quad \quad \quad \quad \forall \text{ configuration } c_{a-1} \text{ with } c_{a-2} \vdash_{\text{hom}} c_{a-1} \\
& \quad \quad \quad \quad \quad \quad \quad \exists \text{ accepting configuration } c_a : c_{a-1} \vdash_{\exists} c_a
\end{aligned} \tag{3}$$

For our reduction, fix a language $L \in \text{STA}(*, \exp_H(2, \text{poly}(n)), \text{poly}(n))$. Then there exist an alternating Turing machine M and polynomial functions $p, q: \mathbb{N} \rightarrow \mathbb{N}$ such that M is $\exp_H(2, p(n))$ -time bounded and L is the set of words w that are accepted by M with $q(|w|)$ alternations. For notational simplicity, we assume $q(n)$ to be odd for all $n \in \mathbb{N}$.

Let $w \in \Sigma^*$. Furthermore, let $N = p(|w|)^2$. We want to express the acceptance of w by M by a formula of polynomial size over

$$\mathcal{M}_{p(n)^2} = (\Delta^{\leq \exp_H(p(n)^2, p(n)^2)}, \cdot, (a)_{a \in \Delta}).$$

To achieve this, first note the following:

- A word c is an existential configuration if it satisfies

$$\begin{aligned}
\text{conf}_{\exists}(c) = \exists x, y \forall z_1, z_2: & \quad \bigwedge_{a \in Q \cup \{\triangleleft, \triangleright\}} (x \neq z_1 a z_2 \wedge y \neq z_1 a z_2) \\
& \quad \wedge \bigvee_{q \in Q, \text{tp}(q) = \exists} c = \triangleright x q y \triangleleft.
\end{aligned}$$

Universal and accepting configurations are described similarly by formulas $\text{conf}_{\forall}(c)$ and $\text{conf}_{\text{acc}}(c)$, respectively. Let $\text{conf} = \text{conf}_{\exists} \vee \text{conf}_{\forall}$.

- A word c is an initial configuration with input w , i.e., $c \in \triangleright \iota w \square^* \triangleleft$, iff it satisfies

$$\text{init}_w(c) = \exists y \left(c = \triangleright \iota w y \triangleleft \wedge \forall z_1, z_2: \bigwedge_{a \in \Delta \setminus \{\square\}} y \neq z_1 a z_2 \right).$$

- $c \vdash_M c'$ iff they satisfy

$$\text{step}(c, c') = \text{conf}(c) \wedge \text{conf}(c') \wedge \exists x, y: \bigvee_{(\ell, r) \in R} (c = x \ell y \wedge c' = x r y)$$

where R is some finite subset of $\Delta^3 \times \Delta^3$.

► **Lemma 9.** *There is a formula $\text{comp}_{\text{hom}}(\bar{x}, \bar{y})$ such that for any configurations c and c' , we have $\mathcal{M}_{p(n)^2} \models \text{comp}_{\text{hom}}(c, c')$ if, and only if, there exists a homogeneous computation*

$$c = c_0 \vdash c_1 \vdash c_2 \vdash \dots \vdash c_K = c'$$

with

$$\sum_{0 \leq i \leq K} |c_i| \leq \exp_H(p(n)^2, p(n)^2). \tag{4}$$

Similarly, there is a formula comp_{\exists} expressing the existence of an existential computation with the same length bound.

3:10 Elementary Complexity Classes with Theories of Automatic Structures

Proof. We will express the existence of a word $W = c_0 c_1 c_2 \dots c_K$ such that

- $c = c_0$,
- $c_i \vdash_M c_{i+1}$ for all $0 \leq i < K$,
- $c_K = c'$,
- and all configurations c_i for $i < K$ have the type of c_0 .

Note that this is the case iff there exists a word W such that

- c is a prefix of W ,
- c' is a suffix of W ,
- any factor x of W that is a configuration is either a suffix of W or followed by a factor y which is a configuration satisfying $x \vdash_M y$. In the latter case, its type is that of c .

If we consider this formula in the free monoid Δ^* , then it expresses the existence of a homogeneous computation from c to c' of arbitrary length. In the structure $\mathcal{M}_{p(n)^2}$, the length of the word W is bounded by $\exp_H(p(n)^2, p(n)^2)$. Hence we get (4). ◀

► **Proposition 10.** *From $w \in \Sigma^*$ with $|w| = n$, we can compute in polynomial time a sentence φ_w such that $w \in L$ if, and only if, $\mathcal{M}_{p(n)^2} \models \varphi_w$.*

Proof. Let φ_w be the following sentence:

$$\begin{aligned}
 \exists c_0 : \text{init}_w(c_0) \\
 \wedge \exists c_1 : \text{conf}(c_1) \wedge \text{comp}_{\text{hom}}(c_0, c_1) \\
 \wedge \forall c_2 : \text{conf}(c_2) \wedge \text{comp}_{\text{hom}}(c_1, c_2) \\
 \rightarrow \exists c_3 : \text{conf}(c_3) \wedge \text{comp}_{\text{hom}}(c_2, c_3) \\
 \wedge \forall c_4 : \text{conf}(c_4) \wedge \text{comp}_{\text{hom}}(c_3, c_4) \\
 \vdots \\
 \exists c_{q(n)-2} : \text{conf}(c_{q(n)-2}) \wedge \text{comp}_{\text{hom}}(c_{q(n)-3}, c_{q(n)-2}) \\
 \wedge \forall c_{q(n)-1} : \text{conf}(c_{q(n)-1}) \wedge \text{comp}_{\text{hom}}(c_{q(n)-2}, c_{q(n)-1}) \\
 \rightarrow \exists c_{q(n)} : \quad \text{conf}_{\text{acc}}(c_{q(n)}) \\
 \wedge \quad \text{comp}_{\exists}(c_{q(n)-1}, c_{q(n)})
 \end{aligned}$$

Since this is the direct translation of the acceptance condition by alternating Turing machines (3), we obtain that $\mathcal{M}_{p(n)^2} \models \varphi_w$ implies $w \in L$.

Conversely, suppose $w \in L$, i.e., (3) holds. Since M is $\exp(2, p(n))$ -time bounded, any computation starting from a configuration $c_0 \in \triangleright \iota w \square^* \triangleleft$ has length $\leq \exp_H(2, p(n))$; in particular, the machine's head can only move $\exp_H(2, p(n))$ cells to the right. Since (3) quantifies over reachable configurations, only, we can restrict quantification in (3) to configurations of length $\leq \exp_H(2, p(n))$. Furthermore, (3) quantifies over computations (hidden in the statements $c_i \vdash_{\text{hom}} c_{i+1}$ and $c_{a-1} \vdash_{\exists} c_a$). Since these computations start in reachable configurations, their length is at most $\exp_H(2, p(n))$ and all intermediate configurations are reachable and therefore of length $\leq \exp_H(2, p(n))$. Note that

$$(\exp_H(2, p(n)) + 1) \cdot \exp_H(2, p(n)) \leq \exp_H(p(n)^2, p(n)^2).$$

Hence, statements of the form $c_i \vdash_{\text{hom}} c_{i+1}$ can be replaced by statements of the form $\mathcal{M}_{p(n)^2} \models \text{comp}_{\text{hom}}(c_i, c_{i+1})$ (and similarly for $c_{q(n)-1} \vdash_{\exists} c_{q(n)}$). Thus, in summary, we get $\mathcal{M}_{p(n)^2} \models \varphi_w$. ◀

5.2 Interpretation of the bounded free monoid in F_{H+2}

To complete the reduction of L to the theory of the forest F_{H+2} , it remains to provide an interpretation of the theory of $\mathcal{M}_{p(n)^2}$ in F_{H+2} . This interpretation has to be computable in time polynomial in $N = p(n)^2$. This reduction requires to express certain numerical properties. Therefore, we first show how to encode numbers by nodes from F_{H+2} and how to do some restricted form of arithmetic.

5.2.1 Nodes as numbers

Let $N \geq 3$. We define the number $\llbracket v \rrbracket_N$ for any node v of the forest F_{H+2} . Let v_1, \dots, v_ℓ be the children of v (if v is of height 0, then there is no such child, i.e., $\ell = 0$). For $k \in \mathbb{N}$, let t_k denote the number of children v_i with $\llbracket v_i \rrbracket_N = k$, i.e.,

$$t_k = |\{i \mid 1 \leq i \leq \ell, \llbracket v_i \rrbracket_N = k\}|.$$

Note that $t_k = 0$ for almost all k since any node of F_{H+2} has only finitely many children. We want to consider the number t_k as k -th digit in a base- N -representation of some natural number. Therefore, we normalize this number to

$$d_k = \min(t_k, N - 1)$$

such that $d_k \in \{0, 1, \dots, N - 1\}$. Let $\chi_N(v) = (d_k)_{k \in \mathbb{N}}$ denote the *characteristic* of v and define

$$\llbracket v \rrbracket_N = \sum_{k \in \mathbb{N}} d_k \cdot b^k.$$

Note that the sequence $\chi_N(v)$ is the base- N -representation of the number $\llbracket v \rrbracket_N$.²

► **Example 11.** The number 0 is represented by all nodes of height 0, i.e., all leaves in F_{H+2} . A number $i \in \{1, 2, \dots, N - 2\}$ is represented by all nodes of height 1 with precisely i children. Any height-1-node with $\geq N - 1$ children represents the number $N - 1$. If $a_m \in \{0, 1, \dots, N - 1\}$ for $0 \leq m < n$, then $a = \sum_{0 \leq m < N} a_m b^m$ is represented, e.g., by a height-2-node v such that a_m children v have m children, i.e., represent the number m (for all $0 \leq m < N$). If $a_m = N - 1$, then we can even add further children representing m without changing $\llbracket v \rrbracket_N$.

By induction, one obtains for any node v of height h :

$$\begin{array}{ll} \llbracket v \rrbracket_N = 0 & \text{if } h = 0 \\ \exp_{h-2}(N, N) \leq \llbracket v \rrbracket_N < \exp_{h-1}(N, N) & \text{if } h \geq 1 \end{array}$$

Conversely (for $h \leq H + 2$), any $a < \exp_{h-1}(N, N)$ is represented by some node of height $\leq h$.

We next show that the relations $\llbracket v_1 \rrbracket_N < \llbracket v_2 \rrbracket_N$ and $\llbracket v_1 \rrbracket_N = \llbracket v_2 \rrbracket_N$ can be defined by first-order formulas.

² For $N = 2$, this is a simple variation of the encoding from [9]. For this case, Flum and Grohe also prove Lemma 12i, but neither Lemma 12ii nor Lemma 13. In contrast to them, we measure the size of our formulas in terms of N while H is considered a constant.

► **Lemma 12.** *From $N \in \mathbb{N}$, one can compute formulas $\text{eq}_N(x_1, x_2)$ and $\text{less}_N(x_1, x_2)$ in time polynomial in N such that for any two nodes v_1 and v_2 in F_{H+2} the following hold:*

- (i) $(F_{H+2}, v_1, v_2) \models \text{eq}_N$ if, and only if, $\llbracket v_1 \rrbracket_N = \llbracket v_2 \rrbracket_N$ and
- (ii) $(F_{H+2}, v_1, v_2) \models \text{less}_N$ if, and only if, $\llbracket v_1 \rrbracket_N < \llbracket v_2 \rrbracket_N$.

Proof. For $0 \leq h \leq H+2$, we can construct a formula in time $O(h)$ expressing that the height of a node is at most h : $\neg \exists x_0, x_1, \dots, x_{h+1} : x = x_0 \wedge \bigwedge_{0 \leq i \leq h} E(x_i, x_{i+1})$. We abbreviate this formula by $\text{hgt}_{\leq h}(x)$.

Let v_1 and v_2 be nodes of F_{H+2} . Then $\llbracket v_1 \rrbracket_N = \llbracket v_2 \rrbracket_N$ if, and only if, $\chi_N(v_1) = \chi_N(v_2)$. But this is the case if, and only if, for all children v of v_1 or v_2 , the number of children v'_1 of v_1 with $\llbracket v \rrbracket_N = \llbracket v'_1 \rrbracket_N$ equals the number of children v'_2 of v_2 with $\llbracket v \rrbracket_N = \llbracket v'_2 \rrbracket_N$ or both numbers are $\geq N-1$. Thus, to build the formula eq_N , we have to apply the same formula to nodes of smaller height. Therefore, we first construct formulas eq_N^h that satisfy i at least for all nodes v_1 and v_2 of height at most h (for $0 \leq h \leq H+2$). The first claim then follows with $\text{eq}_N = \text{eq}_N^{H+2}$.

The formula $\text{eq}_N^0 = (x_1 = x_1)$ satisfies i for nodes of height ≤ 0 since, whenever v_1 and v_2 are nodes of height 0, they both represent 0. We define eq_N^{h+1} as follows:

$$\text{eq}_N^{h+1} = \forall y : \left((E(x_1, y) \vee E(x_2, y)) \rightarrow \bigwedge_{1 \leq i < N} \left(\begin{array}{l} \exists \geq^i y_1 : E(x_1, y_1) \wedge \text{eq}_N^h(y, y_1) \\ \leftrightarrow \\ \exists \geq^i y_2 : E(x_2, y_2) \wedge \text{eq}_N^h(y, y_2) \end{array} \right) \right)$$

By the above explanation and by induction, this formula satisfies i for all nodes of height $\leq h+1$. This completes the definition of the formula $\text{eq}_N = \text{eq}_N^{H+2}$.

By induction, there are constants c_1, c_2, \dots, c_{H+2} such that, for sufficiently large n , we have $|\text{eq}_N^{h+1}| \leq c_{h+1}(n^3 + |\text{eq}_N^h|)$. Consequently,

$$|\text{eq}_N^{H+1}| \in O(N^{3 \cdot (H+2)}).$$

Since H was fixed from the beginning, the formula $\text{eq}_N^{H+2} = \text{eq}_N$ can be constructed from N in time polynomial in N .

Similarly, we construct formulas less_N^h that satisfy ii at least for all nodes v_1 and v_2 of height at most h (for $0 \leq h \leq H+2$). The second claim then follows with $\text{less}_N = \text{less}_N^{H+2}$.

Let $\chi_N(v_i) = (d_k^i)_{k \in \mathbb{N}}$ for $i \in \{1, 2\}$ be the characteristic of v_i . Then $\llbracket v_1 \rrbracket_N < \llbracket v_2 \rrbracket_N$ if, and only if, $\chi_N(v_1)$ is lexicographically properly smaller than $\chi_N(v_2)$. This means that there is some $k \in \mathbb{N}$ with $d_k^1 < d_k^2$ and $d_i^1 \leq d_i^2$ for all $i < k$. Since, in particular, $d_k^2 > 0$, there is a child v' of v_2 with $\llbracket v' \rrbracket_N = k$.

The formula $\text{less}_N^0 = (x_1 = x_1)$ satisfies the required property. Let less_N^{h+1} denote the following formula:

$$\begin{aligned} \exists y : & \quad E(x_2, y) \wedge \bigvee_{1 \leq i < N} \left(\begin{array}{l} \neg \exists \geq^i y_1 : E(x_1, y_1) \wedge \text{eq}_N^h(y, y_1) \\ \wedge \\ \exists \geq^i y_2 : E(x_2, y_2) \wedge \text{eq}_N^h(y, y_2) \end{array} \right) \\ \wedge & \quad \bigwedge_{1 \leq i < N} \forall z : \left(\begin{array}{l} (E(x_1, z) \wedge \text{less}_N^h(z, y) \wedge \exists \geq^i z_1 : E(x_1, z_1) \wedge \text{eq}_N^h(z, z_1)) \\ \rightarrow \\ \exists \geq^i z_2 : E(x_2, z_2) \wedge \text{eq}_N^h(z, z_2) \end{array} \right) \end{aligned}$$

By induction, there are constants c_1, c_2, \dots, c_{H+2} such that, for sufficiently large N , we have $|\text{less}_N^{h+1}| \leq c_{h+1}(N^3 + |\text{eq}_N^h| + |\text{less}_N^h|)$. Consequently,

$$|\text{less}_N^{H+1}| \in O(N^{3 \cdot (H+2)}).$$

Since H was fixed from the beginning, the formula $\text{less}_N^{H+2} = \text{less}_N$ can be constructed from N in time polynomial in N . ◀

Using the two formulas from above, we are now able to also define addition:

► **Lemma 13.** *From $N \in \mathbb{N}$, one can compute a formula $\text{add}_N(x_1, x_2, x_3)$ in time polynomial in N such that for any three nodes v_1, v_2 , and v_3 in F_{H+2} , the following holds:*

$$(F_{H+2}, v_1, v_2, v_3) \models \text{add}_N \text{ if, and only if, } \llbracket v_1 \rrbracket_N + \llbracket v_2 \rrbracket_N = \llbracket v_3 \rrbracket_N.$$

Proof. In the following explanations, let $t = \exp_H(N, N)$.

Let v_1, v_2 , and v_3 be nodes from F_{H+2} , and let $\chi_N(v_i) = (d_k^i)_{k \in \mathbb{N}}$ for all $1 \leq i \leq 3$. Then $d_k^i = 0$ for all $k \geq t$ since the height of v_i is $\leq H + 2$, i.e, its children (being of height $\leq H + 1$) represent numbers $< t$. Since $(d_k^i)_{0 \leq k < t}$ is the base- N -representation of $\llbracket v_i \rrbracket_N$, the following are equivalent:

- $\llbracket v_1 \rrbracket_N + \llbracket v_2 \rrbracket_N = \llbracket v_3 \rrbracket_N$
- There exist $e_k \in \{0, 1\}$ (the carry bits) for $0 \leq k < t$ such that
 - (a) $e_0 = 0$,
 - (b) $d_k^3 + N \cdot e_{k+1} = d_k^1 + d_k^2 + e_k$ for $0 \leq k < t - 1$, and
 - (c) $d_{t-1}^3 = d_{t-1}^1 + d_{t-1}^2 + e_{t-1}$.

We will translate this description into the formula add_N . Note that nodes of height $H + 2$ have characteristics of length t (more precisely: from the entry number t on, they are constantly zero). Hence any sequence $(e_0, e_1, \dots, e_{t-1}, 0, 0, \dots)$ of bits is the characteristics of some node y . Furthermore note that we have to quantify over numbers k with $0 \leq k < t - 1$ but these are precisely the values of nodes of height $\leq H + 1$. Therefore, the following formulas succ_N and max_N will become useful.

The formula

$$\text{succ}_N(z, z') = \text{hgt}_{\leq H+1}(z) \wedge \text{hgt}_{\leq H+1}(z') \wedge \text{less}_N(z, z') \wedge \neg \exists z'' : \text{less}_N(z, z'') \wedge \text{less}_N(z'', z')$$

expresses that z and z' are two nodes of height $\leq H + 1$ satisfying $\llbracket z \rrbracket_N + 1 = \llbracket z' \rrbracket_N$. Furthermore, the formula

$$\text{max}_N(z) = \text{hgt}_{\leq H+1}(z) \wedge \neg \exists z' : \text{hgt}_{\leq H+1}(z') \wedge \text{less}_N(z, z')$$

expresses that z is a node of height at most $H + 1$ that represents the maximal possible value for such a node, i.e., $\llbracket z \rrbracket_N = t - 1$.

Let I denote the set of quintuples $(a_1, a_2, b_1, a_3, b_2)$ of natural numbers from $\{0, 1, \dots, n - 1\}$ with $a_1 + a_2 + b_1 = a_3 + N \cdot b_2$. Finally, for $i \in \{0, 1, \dots, N - 1\}$ set

$$Q^i x \varphi = \begin{cases} \exists^{=i} x \varphi & \text{if } i < N - 1 \\ \exists^{\geq N-1} x \varphi & \text{if } i = N - 1. \end{cases}$$

Now consider the following formula $\text{add}_N(x_1, x_2, x_3)$:

$$\begin{aligned} \exists y \forall z, z' : & (E(y, z) \wedge E(y, z') \wedge \text{eq}_N(z, z')) \rightarrow (z = z' \wedge \exists y' : E(z, y')) \\ \wedge \text{succ}(z, z') \rightarrow & \bigvee_{(a_1, a_2, b_1, a_3, b_2) \in I} \left(\begin{array}{l} Q^{a_1} x'_1 : E(x_1, x'_1) \wedge \text{eq}_N(x'_1, z) \\ \wedge Q^{a_2} x'_2 : E(x_2, x'_2) \wedge \text{eq}_N(x'_2, z) \\ \wedge Q^{b_1} y' : E(y, y') \wedge \text{eq}_N(y', z) \\ \wedge Q^{a_3} x'_3 : E(x_3, x'_3) \wedge \text{eq}_N(x'_3, z) \\ \wedge Q^{b_2} y' : E(y, y') \wedge \text{eq}_N(y', z') \end{array} \right) \\ \wedge \text{max}(z) \rightarrow & \bigvee_{(a_1, a_2, b_1, a_3, 0) \in I} \left(\begin{array}{l} Q^{a_1} x'_1 : E(x_1, x'_1) \wedge \text{eq}_N(x'_1, z) \\ \wedge Q^{a_2} x'_2 : E(x_2, x'_2) \wedge \text{eq}_N(x'_2, z) \\ \wedge Q^{b_1} y' : E(y, y') \wedge \text{eq}_N(y', z) \\ \wedge Q^{a_3} x'_3 : E(x_3, x'_3) \wedge \text{eq}_N(x'_3, z) \end{array} \right) \end{aligned}$$

Let y be some node of F_{H+2} such that the formula starting with $\forall z$ holds. Let furthermore $(e_k)_{k \in \mathbb{N}}$ be the characteristic of the node y . Since the height of y is $\leq H + 2$, we get $e_k = 0$ for all $k \geq t$. The first conjunct expresses $e_k \in \{0, 1\}$ (since no two distinct children of y represent the same number) and $e_0 = 0$ (since no child of y has height 0, i.e., represents 0). Having said this, it is clear that the second and third conjunct ensure properties (b) and (c) from above. Thus, indeed, the formula add_N expresses the relation $\llbracket v_1 \rrbracket_N + \llbracket v_2 \rrbracket_N = \llbracket v_3 \rrbracket_N$.

Furthermore note that $|I| \leq N^5$. Hence, using Lemma 12, the formula add_N can be constructed in polynomial time from N . \blacktriangleleft

5.2.2 Tuples of nodes as words

In the previous section, we agreed how to consider a node v of depth ≥ 1 (and therefore of height $\leq H + 1$) as a number $\llbracket v \rrbracket_N$ between 0 and $\exp_H(N, N) - 1$. Now, we want to consider a tuple $\bar{v} = (v_a)_{a \in \Delta}$ of nodes as word $\text{word}_N(\bar{v})$ over the alphabet Δ . To this aim, let

$$P_a = \{\llbracket v'_a \rrbracket_N \mid (v_a, v'_a) \in E\}$$

denote the set of numbers represented by children of the node v_a (for $a \in \Delta$). The word $\text{word}_N(\bar{v})$ is defined only in case these sets of numbers are mutually disjoint and the union of these sets is an initial segment of the natural numbers. Let $\ell = \sup(\bigcup_{a \in \Delta} P_a)$. Then $\text{word}_N(\bar{v})$ is the word

$$a_0 a_1 a_2 \dots a_\ell$$

with $a_k = a \iff k \in P_a \iff k = \llbracket v'_a \rrbracket_N$ for some child v'_a of v_a . Thus, the children of the node v_a represent the positions of the letter a in $\text{word}_N(\bar{v})$. Since children of nodes have height $\leq H + 1$, the word $\text{word}_N(\bar{v})$ has length $\leq \exp_H(N, N)$. Conversely, any word of this length can be represented by a tuple of nodes $\text{word}_N(\bar{v})$.

► **Lemma 14.** *From $N \in \mathbb{N}$, one can compute in polynomial time formulas $\text{is_word}_N(\bar{x})$ and $\text{prod}(\bar{x}, \bar{y}, \bar{z})$, such that, for any Δ -tuples \bar{u}, \bar{v} , and \bar{w} of nodes, the following hold:*

- $(F_{H+2}, \bar{v}) \models \text{is_word}_N$ if, and only if, the tuple $\text{word}_N(\bar{v})$ is defined.
- $(F_{H+2}, \bar{u}, \bar{v}, \bar{w}) \models \text{prod}$ if, and only if, $\text{word}_N(\bar{u})$, $\text{word}_N(\bar{v})$, and $\text{word}_N(\bar{w})$ are defined and $\text{word}_N(\bar{u}) \text{ word}_N(\bar{v}) = \text{word}_N(\bar{w})$.

Proof. The formula is_word_N looks as follows:

$$\forall x, y \left(\bigvee_{a \in \Delta} E(x_a, y) \wedge \text{less}_N(x, y) \right) \rightarrow \exists x' \left(\bigvee_{b \in \Delta} E(x_b, x') \wedge \text{eq}_N(x, x') \right) \\ \wedge \bigwedge_{a, b \in \Delta, a \neq b} \left((E(x_a, x) \wedge E(x_b, y)) \rightarrow \neg \text{eq}_N(x, y) \right)$$

The first line expresses that $\bigcup_{a \in \Delta} P_a$ is an initial segment of (\mathbb{N}, \leq) , the second one ensures that the sets P_a are mutually disjoint.

Note that the length of the word $\text{word}_N(\bar{v})$ is the successor of the maximal number represented by any of the children of nodes v_a from the tuple \bar{v} . Therefore, the following formula ensures that the length of $\text{word}_N(\bar{x})$ equals $\llbracket \ell \rrbracket_N$:

$$\forall x: \bigwedge_{a \in \Delta} (E(x_a, x) \rightarrow \text{less}_N(x, \ell)) \\ \wedge \exists x: \bigvee_{a \in \Delta} E(x_a, x) \wedge \neg \exists y: \text{less}_N(x, y) \wedge \text{less}_N(y, \ell)$$

We just remark that representable words have length $\leq \exp_H(N, N)$. Hence, their length is always represented by some node of height $\leq H + 2$.

We denote the above formula by $|\text{word}_N(\bar{x})| = \ell$. Now the formula prod looks as follows:

$$\begin{aligned} \exists \ell_x, \ell_y, \ell_z: & \text{is_word}_N(\bar{x}) \wedge \text{is_word}_N(\bar{y}) \wedge \text{is_word}_N(\bar{z}) \\ & \wedge |\text{word}_N(\bar{x})| = \ell_x \wedge |\text{word}_N(\bar{y})| = \ell_y \wedge |\text{word}_N(\bar{z})| = \ell_z \\ & \wedge \text{add}_N(\ell_x, \ell_y, \ell_z) \\ & \wedge \bigwedge_{a \in \Delta} \forall x \exists z: E(x_a, x) \rightarrow E(z_a, z) \wedge \text{eq}_N(x, z) \\ & \wedge \bigwedge_{a \in \Delta} \forall y \exists z: E(y_a, y) \rightarrow E(z_a, z) \wedge \text{add}_N(y, \ell_x, z) \end{aligned} \quad \blacktriangleleft$$

► **Observation 15.** *From $N \in \mathbb{N}$ and $a \in \Delta$, one can construct in polynomial time a formula $\text{is_letter}_{N,a}(\bar{x})$ such that, for any Δ -tuple \bar{u} of nodes, we have*

$$(F_{H+2}, \bar{u}) \models \text{is_letter}_{N,a}(\bar{x}) \iff \text{word}_N(\bar{u}) \text{ is defined and equals } a.$$

This is obtained by the formula

$$\text{is_word}_N(\bar{u}) \wedge \bigwedge_{b \neq a} \forall y \neg E(x_b, y) \wedge \exists^{=1} y E(x_a, y).$$

This finishes the construction of an interpretation of the bounded free monoid \mathcal{M}_N in the forest F_{H+2} . Since all the formulas is_word_N , prod_N , and $\text{is_letter}_{N,a}$ can be computed in polynomial time, we can reduce the theory of the bounded free monoid \mathcal{M}_N in polynomial time to the theory of F_{H+2} . Together with Proposition 10, this finishes the proof of the following theorem:

► **Theorem 16.** *The theory of the forest F_{H+2} is hard for the class*

$$\text{STA}(*, \exp_H(2), \text{poly}(n)), \text{poly}(n).$$

6 Conclusion

We have shown that for every h there is an automatic structure, whose theory is complete for the Berman complexity class $\text{STA}(*, \exp_h(2), \text{poly}(n)), \text{poly}(n)$. Therefore theories of automatic structures are distributed across all stages of elementary complexity. The variants F_H^1 and F_H^∞ of our structure F_H that we mentioned in the beginning might be interesting in their own right. A careful analysis of our proof reveals without much effort that the theories of these two structures have the same complexity as the theory of F_H .

► **Theorem 17.** *The theories of F_H^1 and F_H^∞ are complete for*

$$\text{STA}(*, \exp_H(2), \text{poly}(n)), \text{poly}(n).$$

Finally let us mention a related problem from parameterized complexity theory.

► **Conjecture 18.** *There is no algorithm that determines correctly for every tree T of height at most H and every first-order sentence φ whether $T \models \varphi$ in time $\exp_{H-3}(2, \text{poly}(|\varphi|)) \cdot \text{poly}(|T|)$.*

An upper bound this problem is given in [12]. It might be possible to prove Conjecture 18 (under suitable complexity theoretic assumptions) with a similar strategy as it was used in [10] for the class of all finite trees. The formulas that we defined for our lower bound might be useful in this case.

References


- 1 V. Bárány, E. Grädel, and S. Rubin. Automata-based presentations of infinite structures. In *Finite and Algorithmic Model Theory*, pages 1–76. Cambridge University Press, 2011.
- 2 L. Berman. The complexity of logical theories. *Theoretical Computer Science*, 11:71–77, 1980.
- 3 A. Blumensath. Automatic structures. Technical report, RWTH Aachen, 1999.
- 4 A. Blumensath and E. Grädel. Automatic Structures. In *LICS'00*, pages 51–62. IEEE Computer Society Press, 2000.
- 5 Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49(2):129–141, 1961. URL: <http://eudml.org/doc/213582>.
- 6 C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Am. Math. Soc.*, 98:21–51, 1961.
- 7 S. Feferman and R.L. Vaught. The first order properties of algebraic systems. *Fund. Math.*, 47:57–103, 1959.
- 8 J. Ferrante and Ch. Rackoff. *The Computational Complexity of Logical Theories*. Lecture Notes in Mathematics vol. 718. Springer, 1979.
- 9 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, Heidelberg, 2006.
- 10 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004.
- 11 Ch. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. *Theor. Comput. Sci.*, 108:45–82, 1993.
- 12 Jakub Gajarský and Petr Hliněný. Faster deciding MSO properties of trees of fixed height, and some consequences. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, pages 112–123, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.112.
- 13 B.R. Hodgson. On direct products of automaton decidable theories. *Theoretical Computer Science*, 19:331–335, 1982.
- 14 B. Khossainov and M. Minnes. Three lectures on automatic structures. In *Proceedings of Logic Colloquium*, pages 132–176, 2007.
- 15 B. Khossainov and A. Nerode. Automatic presentations of structures. In *Logic and Computational Complexity*, Lecture Notes in Comp. Science vol. 960, pages 367–392. Springer, 1995.
- 16 B. Khossainov, S. Rubin, and F. Stephan. Definability and regularity in automatic structures. In *STACS'04*, Lecture Notes in Comp. Science vol. 2996, pages 440–451. Springer, 2004.
- 17 Bakhadyr Khossainov, Jiamou Liu, and Mia Minnes. Unary automatic graphs: an algorithmic perspective. *Mathematical Structures in Computer Science*, 19(1):133–152, 2009.
- 18 D. Kuske. Theories of automatic structures and their complexity. In *CAI 2009*, Lecture Notes in Comp. Science vol. 5725, pages 81–98. Springer, 2009.
- 19 D. Kuske and M. Lohrey. Some natural decision problems in automatic graphs. *Journal of Symbolic Logic*, 75(2):678–710, 2010.
- 20 D. Kuske and M. Lohrey. Automatic structures of bounded degree revisited. *Journal of Symbolic Logic*, 76(4):1352–1380, 2011.
- 21 P. Lindner. Theorien automatischer Strukturen in der Exponentialzeithierarchie. Master's thesis, TU Ilmenau, 2017.
- 22 D.C. Oppen. A $2^{2^{cn}}$ upper bound on the complexity of Presburger arithmetic. *Journal of Computer and System Sciences*, 16:323–332, 1978.
- 23 S. Rubin. Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic*, 14:169–209, 2008.
- 24 F. Stephan. Automatic structures – recent results and open questions. *Journal of Physics: Conference Series*, 632:012013, 2015.

High-Level Signatures and Initial Semantics

Benedikt Ahrens

University of Birmingham, UK


B.Ahrens@cs.bham.ac.uk

 <https://orcid.org/0000-0002-6786-4538>

André Hirschowitz

Université Nice Sophia Antipolis, France

ah@unice.fr

 <https://orcid.org/0000-0003-2523-1481>

Ambroise Lafont

IMT Atlantique

Inria, LS2N CNRS, France


ambroise.lafont@inria.fr

 <https://orcid.org/0000-0002-9299-641X>

Marco Maggesi¹

Università degli Studi di Firenze, Italy

marco.maggesi@unifi.it

 <https://orcid.org/0000-0003-4380-7691>

Abstract

We present a device for specifying and reasoning about syntax for datatypes, programming languages, and logic calculi. More precisely, we consider a general notion of “signature” for specifying syntactic constructions. Our signatures subsume classical algebraic signatures (i.e., signatures for languages with variable binding, such as the pure lambda calculus) and extend to much more general examples.

In the spirit of Initial Semantics, we define the “syntax generated by a signature” to be the initial object – if it exists – in a suitable category of models. Our notions of signature and syntax are suited for compositionality and provide, beyond the desired algebra of terms, a well-behaved substitution and the associated inductive/recursive principles.

Our signatures are “general” in the sense that the existence of an associated syntax is not automatically guaranteed. In this work, we identify a large and simple class of signatures which do generate a syntax.

This paper builds upon ideas from a previous attempt by Hirschowitz-Maggesi, which, in turn, was directly inspired by some earlier work of Ghani-Uustalu-Hamana and Matthes-Uustalu.

The main results presented in the paper are computer-checked within the UniMath system.

2012 ACM Subject Classification Theory of computation → Algebraic language theory

Keywords and phrases initial semantics, signatures, syntax, monadic substitution, computer-checked proofs

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.4

Supplement Material Computer-checked proofs with compilation instructions on <https://github.com/amlafont/largecatmodules>

¹ Supported by GNSAGA-INdAM and MIUR.



Funding This work has partly been funded by the CoqHoTT ERC Grant 637339. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0363.

Acknowledgements We would like to thank the anonymous referees for their helpful comments. Their constructive criticism led us to a deep revision of our presentation.

1 Introduction

1.1 Initial Semantics

The concept of characterizing data through an initiality property is standard in computer science, where it is known under the terms *Initial Semantics* and *Algebraic Specification* [21], and has been popularized by the movement of *Algebra of Programming* [5].

This concept offers the following methodology to define a *formal language*²:

1. Introduce a notion of signature.
2. Construct an associated notion of model (suitable as domain of interpretation of the syntax generated by the signature). Such models should form a category.
3. Define the *syntax generated by a signature* to be its initial model, when it exists³.
4. Find a satisfactory sufficient condition for a signature to generate a syntax.

For a notion of signature to be satisfactory, it should satisfy the following conditions:

- it should extend the notion of algebraic signature, and
- complex signatures should be built by assembling simpler ones, thereby opening room for compositionality properties.

In the present work we consider a general notion of signature – together with its associated notion of model – which is suited for the specification of untyped programming languages with variable binding. On one hand, our signatures are fairly more general than those introduced in some of the seminal papers on this topic [10, 15, 11], which are essentially given by a family of lists of natural numbers indicating the number of variables bound in each subterm of a syntactic construction (we call them “algebraic signatures” below). On the other hand, the existence of an initial model in our setting is not automatically guaranteed.

The main result of this paper is a sufficient condition on a signature to ensure such an existence. Our condition is still satisfied far beyond the algebraic signatures mentioned above. Specifically, our signatures form a cocomplete category and our condition is preserved by colimits (Section 7). Examples are given in Section 8.

Our notions of signature and syntax enjoy modularity in the sense introduced by [13]: indeed, we define a “total” category of models where objects are pairs consisting of a signature together with one of its models; and in this total category of models, merging two extensions of a syntax corresponds to building an amalgamated sum.

The present work improves a previous attempt [18] in two main ways: firstly, it gives a much simpler condition for the existence of an initial model, secondly, it provides computer-checked proofs for all the main statements.

² Here, the word “language” encompasses data types, programming languages and logic calculi, as well as languages for algebraic structures as considered in Universal Algebra.

³ In the literature, the word signature is often reserved for the case where such sufficient condition is automatically ensured.

1.2 Computer-checked formalization

The intricate nature of our main result made it desirable to provide a mechanically checked proof of that result, in conjunction with a human-readable summary of the proof.

Our computer-checked proof is based on the UniMath library [26], which itself is based on the proof assistant Coq [25]. The main reasons for our choice of proof assistant are twofold: firstly, the logical basis of the Coq proof assistant, dependent type theory, is well suited for abstract algebra, in particular, for category theory. Secondly, a suitable library of category theory, ready for use by us, had already been developed [2].

The formalization consists of about 8,000 lines of code, and can be consulted on <https://github.com/amblafont/largecatmodules>. A guide is given in the README.

Here below, we give in `teletype font` the name of the corresponding result in the computer-checked library, when available – often in the format `filename:identifier`.

1.3 Related work

The idea that the notion of monad is suited for modeling substitution concerning syntax (and semantics) has been retained by many contributions on the subject (see e.g. [6, 13, 24, 4]).

Matthes, Uustalu [24], followed by Ghani, Uustalu, and Hamana [13], are the first to consider a form of colimits (namely coends) of signatures. Their treatment rests on the technical device of *strength*⁴ and so did our preliminary version of the present work [18]. Notably, the present version simplifies the treatment by avoiding the consideration of strengths.

We should mention several other mathematical approaches to syntax (and semantics).

Fiore, Plotkin, Turi [10] develop a notion of substitution monoid. Following [3], this setting can be rephrased in terms of relative monads and modules over them [1]. Accordingly, our present contribution could probably be customized for this “relative” approach.

The work by Fiore with collaborators [10, 8, 9] and the work by Uustalu with collaborators [24, 13] share two traits: firstly, the modelling of variable binding by *nested abstract syntax*, and, secondly, the reliance on tensorial strengths in the specification of substitution. In the present work, variable binding is modelled using nested abstract syntax; however, we do without strengths.

Gabbay and Pitts [11] employ a different technique for modelling variable binding, based on nominal sets. We do not see yet how our treatment of more general syntax carries over to nominal techniques.

Yet another approach to syntax is based on Lawvere Theories. This is clearly illustrated in the paper [20], where Hyland and Power also outline the link with the language of monads and put in an historical perspective.

Finally, let us mention the classical approach based on Cartesian closed categories recently revisited and extended by T. Hirschowitz [19].

1.4 Organisation of the paper

Section 2 gives a succinct account of modules over a monad. Our categories of signatures and models are described in Sections 3 and 4 respectively. In Section 5 we give our definition of a syntax, and we show our modularity result about merging extensions of syntax. In Section 6

⁴ A (tensorial) strength for a functor $F : V \rightarrow V$ is given by a natural transformation $\beta_{v,w} : v \otimes Fw \rightarrow F(v \otimes w)$ commuting suitably with the associator and the unitor of the monoidal structure on V .

we show through examples how recursion can be recovered from initiality. Our notions of presentable signature and presentable syntax appear in Section 7. Finally, in Section 8, we give examples of presentable signatures and syntaxes.

2 Categories of modules over monads

2.1 Modules over monads

We recall only the definition and some basic facts about modules over a monad in the specific case of the category \mathbf{Set} of sets, although most definitions are generalizable. See [17] for a more extensive introduction on this topic.

A monad (over \mathbf{Set}) is a monoid in the category $\mathbf{Set} \rightarrow \mathbf{Set}$ of endofunctors of \mathbf{Set} , i.e., a triple $R = (R, \mu, \eta)$ given by a functor $R: \mathbf{Set} \rightarrow \mathbf{Set}$, and two natural transformations $\mu: R \cdot R \rightarrow R$ and $\eta: I \rightarrow R$ such that the following equations hold:

$$\mu \cdot \mu R = \mu \cdot R\mu, \quad \mu \cdot \eta R = 1_R, \quad \mu \cdot R\eta = 1_R.$$

Let R be a monad.

► **Definition 1 (Modules).** A left R -module is given by a functor $M: \mathbf{Set} \rightarrow \mathbf{Set}$ equipped with a natural transformation $\rho: M \cdot R \rightarrow M$, called *module substitution*, which is compatible with the monad composition and identity:

$$\rho \cdot \rho R = \rho \cdot M\mu, \quad \rho \cdot M\eta = 1_M.$$

There is an obvious corresponding definition of right R -modules that we do not need to consider in this paper. From now on, we will write “ R -module” instead of “left R -module” for brevity.

► **Example 2.**

- Every monad R is a module over itself, which we call the *tautological* module.
- For any functor $F: \mathbf{Set} \rightarrow \mathbf{Set}$ and any R -module $M: \mathbf{Set} \rightarrow \mathbf{Set}$, the composition $F \cdot M$ is an R -module (in the evident way).
- For every set W we denote by $\underline{W}: \mathbf{Set} \rightarrow \mathbf{Set}$ the constant functor $\underline{W} := X \mapsto W$. Then \underline{W} is trivially an R -module since $\underline{W} = \underline{W} \cdot R$.
- Let M_1, M_2 be two R -modules. Then the product functor $M_1 \times M_2$ is an R -module (see Proposition 4 for a general statement).

► **Definition 3 (Linearity).** We say that a natural transformation of R -modules $\tau: M \rightarrow N$ is *linear*⁵ if it is compatible with module substitution on either side:

$$\tau \cdot \rho^M = \rho^N \cdot \tau R.$$

We take linear natural transformations as morphisms among modules. It can be easily verified that we obtain in this way a category that we denote $\mathbf{Mod}(R)$.

⁵ Given a monoidal category \mathcal{C} , there is a notion of (left or right) module over a monoid object in \mathcal{C} (see <https://ncatlab.org/nlab/show/module+over+a+monoid> for details). The term “module” comes from the case of rings: indeed, a ring is just a monoid in the monoidal category of Abelian groups. Similarly, our monads are just the monoids in the monoidal category of endofunctors on \mathbf{Set} , and our modules are just modules over these monoids. Accordingly, the term “linear(ity)” for morphisms among modules comes from the paradigmatic case of rings.

Limits and colimits in the category of modules can be constructed point-wise:

► **Proposition 4.** *Mod(R) is complete and cocomplete.*

See `LModule_Colims_of_shape` and `LModule_Lims_of_shape` in `Prelims/LModuleColims` for the formalized proofs.

2.2 The total category of modules

We already introduced the category $\text{Mod}(R)$ of modules with fixed base R . It is often useful to consider a larger category which collects modules with different bases. To this end, we need first to introduce the notion of pullback.

► **Definition 5** (Pullback). Let $f: R \rightarrow S$ be a morphism of monads⁶ and M an S -module. The module substitution $M \cdot R \xrightarrow{Mf} M \cdot S \xrightarrow{\rho} M$ defines an R -module which is called *pullback* of M along f and noted f^*M .⁷

► **Definition 6** (The total module category). We define the *total module category* $\int_R \text{Mod}(R)$ as follows⁸:

- its objects are pairs (R, M) of a monad R and an R -module M .
- a morphism from (R, M) to (S, N) is a pair (f, m) where $f: R \rightarrow S$ is a morphism of monads, and $m: M \rightarrow f^*N$ is a morphism of R -modules.

The category $\int_R \text{Mod}(R)$ comes equipped with a forgetful functor to the category of monads, given by the projection $(R, M) \mapsto R$.

► **Proposition 7.** *The forgetful functor $\int_R \text{Mod}(R) \rightarrow \text{Mon}$ given by the first projection is a Grothendieck fibration with fibre $\text{Mod}(R)$ over a monad R . In particular, any monad morphism $f: R \rightarrow S$ gives rise to a functor*

$$f^*: \text{Mod}(S) \rightarrow \text{Mod}(R)$$

given on objects by Definition 5.

The formal proof is available as `Prelims/modules:cleaving_bmod`.

► **Proposition 8.** *For any monad morphism $f: R \rightarrow S$, the functor f^* preserves limits and colimits.*

See `pb_LModule_colim_iso` and `pb_LModule_lim_iso` in `Prelims/LModuleColims` for the formalized proofs.

2.3 Derivation

For our purposes, important examples of modules are given by the following general construction. Let us denote the final object of Set as $*$.

⁶ An explicit definition of morphism of monads can be found in [17].

⁷ The term “pullback” is standard in the terminology of Grothendieck fibrations (see Proposition 7).

⁸ Our notation for the total category is modelled after the category of elements of a presheaf, and, more generally, after the Grothendieck construction of a pseudofunctor. It overlaps with the notation for categorical ends.

► **Definition 9** (Derivation). For any R -module M , the *derivative* of M is the functor $M' := X \mapsto M(X + *)$. It is an R -module with the substitution $\rho': M' \cdot R \rightarrow M'$ defined as in the diagram

$$\begin{array}{ccc}
 M(R(X) + *) & \xrightarrow{\rho'_X} & M(X + *) \\
 \downarrow M(R(i_X) + \eta_{X+*}) & \nearrow \rho_{X+*} & \\
 M(R(X + *)) & &
 \end{array} \tag{1}$$

where $i_X: X \rightarrow X + *$ and $\underline{*}: * \rightarrow X + *$ are the obvious maps.

Derivation is a cartesian endofunctor on the category $\text{Mod}(R)$ of modules over a fixed monad R . In particular, derivation can be iterated: we denote by $M^{(k)}$ the k -th derivative of M .

► **Definition 10.** Given a list of non negative integers $(a) = (a_1, \dots, a_n)$ and a left module M over a monad R , we denote by $M^{(a)} = M^{(a_1, \dots, a_n)}$ the module $M^{(a_1)} \times \dots \times M^{(a_n)}$. Observe that, when $(a) = ()$ is the empty list, we have $M^{()} = *$ the final module.

► **Proposition 11.** Derivation yields an endofunctor of $\int_R \text{Mod}(R)$ which commutes with any functor f^* induced by a monad morphism f (Proposition 7).

See `LModule_deriv_is_functor` in `Prelims/DerivationIsFunctorial` and `pb_deriv_to_deriv_pb_iso` in `Prelims/LModPbCommute` for the formalized proofs.

We have a natural *substitution morphism* $\sigma: M' \times R \rightarrow M$ defined by $\sigma_X = \rho_X \circ w_X$, where $w_X: M(X + *) \times R(X) \rightarrow M(R(X))$ is the map

$$w_X: (a, b) \mapsto M(\eta_X + \underline{b}), \quad \underline{b}: * \mapsto b.$$

► **Lemma 12.** The transformation σ is linear.

See `Prelims/derivadj:substitution_laws` for the formalized proof.

The substitution σ allows us to interpret the derivative M' as the “module M with one formal parameter added”.

Abstracting over the module turns the substitution morphism into a natural transformation that is the unit of the following adjunction:

► **Proposition 13.** The endofunctor of $\text{Mod}(R)$ mapping M to the R -module $M \times R$ is left adjoint to the derivation endofunctor, the unit being the substitution morphism σ .

See `Prelims/derivadj:deriv_adj` for the formalized proof.

3 The category of signatures

In this section, we give our notion of signature. The destiny of a signature is to have actions in monads. An action of a signature Σ in a monad R should be a morphism from a module $\Sigma(R)$ to the tautological one R . For instance, in the case of the signature Σ of a binary operation, we have $\Sigma(R) := R^2 = R \times R$. Hence a signature assigns, to each monad R , a module over R in a functorial way.

► **Definition 14.** A *signature* is a section of the forgetful functor from the category $\int_R \text{Mod}(R)$ to the category Mon .

Now we give our basic examples of signatures.

- ▶ **Example 15.** The assignment $R \mapsto R$ is a signature, which we denote by Θ .
- ▶ **Example 16.** For any functor $F: \mathbf{Set} \rightarrow \mathbf{Set}$ and any signature Σ , the assignment $R \mapsto F \cdot \Sigma(R)$ yields a signature which we denote $F \cdot \Sigma$.
- ▶ **Example 17.** The assignment $R \mapsto *_R$, where $*_R$ denotes the final module over R , is a signature which we denote by $*$.
- ▶ **Example 18.** Given two signatures Σ and Υ , the assignment $R \mapsto \Sigma(R) \times \Upsilon(R)$ is a signature which we denote by $\Sigma \times \Upsilon$. In particular, $\Theta^2 = \Theta \times \Theta$ is the signature of any (first-order) binary operation, and, more generally, Θ^n is the signature of n -ary operations.
- ▶ **Example 19.** Given two signatures Σ and Υ , the assignment $R \mapsto \Sigma(R) + \Upsilon(R)$ is a signature which we denote by $\Sigma + \Upsilon$. In particular, $\Theta^2 + \Theta^2$ is the signature of a pair of binary operations.

This example explains why we do not need to distinguish here between “arities” – usually used to specify a single syntactic construction – and “signatures” – usually used to specify a family of syntactic constructions; our signatures allow us to do both (via Proposition 23 for families that are not necessarily finitely indexed).

▶ **Definition 20.** For each sequence of non-negative integers $s = (s_1, \dots, s_n)$, the assignment $R \mapsto R^{(s_1)} \times \dots \times R^{(s_n)}$ (see Definition 10) is a signature, which we denote by $\Theta^{(s)}$, or by Θ' in the specific case of $s = 1$. Signatures of this form are said *elementary*.

▶ **Remark 21.** The product of two elementary signatures is elementary.

▶ **Definition 22.** A *morphism between two signatures* $\Sigma_1, \Sigma_2: \mathbf{Mon} \rightarrow \int_R \mathbf{Mod}(R)$ is a natural transformation $m: \Sigma_1 \rightarrow \Sigma_2$ which, post-composed with the projection $\int_R \mathbf{Mod}(R) \rightarrow \mathbf{Mon}$, becomes the identity. Signatures form a subcategory \mathbf{Sig} of the category of functors from \mathbf{Mon} to $\int_R \mathbf{Mod}(R)$.

Limits and colimits of signatures can be easily constructed point-wise:

▶ **Proposition 23.** *The category of signatures is complete and cocomplete. Furthermore, it is distributive: for any signature Σ and family of signatures $(S_o)_{o \in O}$, the canonical morphism $\coprod_{o \in O} (S_o \times \Sigma) \rightarrow (\coprod_{o \in O} S_o) \times \Sigma$ is an isomorphism.*

See `Sig_Lims_of_shape` and `Sig_Colims_of_shape` in `Signatures/SignaturesColims`, and `Sig_isDistributive` in `Signatures/PresentableSignatureBinProdR` for the formalized proofs.

▶ **Definition 24.** An *algebraic signature* is a (possibly infinite) coproduct of elementary signatures.

These signatures are those which appear in [10]. For instance, the algebraic signature of the lambda-calculus is $\Sigma_{\text{LC}} = \Theta^2 + \Theta'$.

4 Categories of models

We define the notion of action of a signature in a monad.

► **Definition 25.** Given a monad R over Set , we define an *action*⁹ of the signature Σ in R to be a module morphism from $\Sigma(R)$ to R .

► **Example 26.** The usual $\text{app}: \text{LC}^2 \rightarrow \text{LC}$ is an action of the elementary signature Θ^2 into the monad LC of syntactic lambda calculus. The usual $\text{abs}: \text{LC}' \rightarrow \text{LC}$ is an action of the elementary signature Θ' into the monad LC . Then $\text{app} + \text{abs}$ is an action of the algebraic signature of the lambda-calculus $\Theta^2 + \Theta'$ into the monad LC .

► **Definition 27.** Given a signature Σ , we build the category Mon^Σ of *models* of Σ as follows. Its objects are pairs (R, r) of a monad R equipped with an action $r: \Sigma(R) \rightarrow R$ of Σ . A morphism from (R, r) to (S, s) is a morphism of monads $m: R \rightarrow S$ compatible with the actions in the sense that the following diagram of R -modules commutes:

$$\begin{array}{ccc} \Sigma(R) & \xrightarrow{r} & R \\ \Sigma(m) \downarrow & & \downarrow m \\ m^*(\Sigma(S)) & \xrightarrow{m^*s} & m^*S \end{array}$$

This is equivalent to asking that the square of underlying natural transformations commutes, i.e., $m \circ r = s \circ \Sigma(m)$. Here, the horizontal arrows come from the actions, the left vertical arrow comes from the functoriality of signatures, and $m: R \rightarrow m^*S$ is the morphism of monads seen as morphism of R -modules.

► **Proposition 28.** *These morphisms, together with the obvious composition, turn Mon^Σ into a category which comes equipped with a forgetful functor to the category of monads.*

In the formalization, this category is recovered as the fiber category over Σ of the displayed category [2] of models, see `Signatures/Signature:rep_disp`.

► **Definition 29 (Pullback).** Let $f: \Sigma \rightarrow \Upsilon$ be a morphism of signatures and $\mathcal{R} = (R, r)$ a model of Υ . The linear morphism $\Sigma(R) \xrightarrow{f} \Upsilon(R) \xrightarrow{r} R$ defines an action of Σ in R . The induced model of Σ is called *pullback*¹⁰ of \mathcal{R} along f and noted $f^*\mathcal{R}$.

5 Syntax

We are primarily interested in the existence of an initial object in the category Mon^Σ of models of a signature Σ . We call this object *the syntax generated by Σ* .

5.1 Representability

► **Definition 30.** Given a signature Σ , a *representation* of Σ is an initial object in Mon^Σ . If such an object exists, we call it *the syntax generated by Σ* and denote it by $\hat{\Sigma}$. In this case, we also say that $\hat{\Sigma}$ *represents* Σ , and we call the signature Σ *representable*¹¹.

► **Theorem 31.** *Algebraic signatures are representable.*

⁹ This terminology is borrowed from the vocabulary of algebras over a monad: an algebra over a monad T on a category \mathcal{C} is an object X of \mathcal{C} with a morphism $\nu: T(X) \rightarrow X$ that is compatible with the multiplication of the monad. This morphism is sometimes called an action.

¹⁰ Following the terminology introduced in Definition 5, the term “pullback” is justified by Lemma 33.

¹¹ For an algebraic signature Σ without binding constructions, the map assigning to any monad R its set of Σ -actions can be upgraded into a functor which is corepresented by the initial model.

This result is proved in a previous work [16, Theorems 1 and 2]. The proof goes as follows: an algebraic signature induces an endofunctor on the category of endofunctors on **Set**. Its initial algebra (constructed as the colimit of the initial chain) is given the structure of a monad with an action of the algebraic signature, and then a routine verification shows that it is actually initial in the category of models. As part of the present work, we provide a computer-checked proof as `algebraic_sig_representable` in the file `Signatures/BindingSig`.

In the following we present a more general representability result: Theorem 35 states that *representable* signatures, which form a superclass of algebraic signatures, are representable.

5.2 Modularity

In this section, we study the problem of how to merge two syntax extensions. Our answer, a “modularity” result (Theorem 32), was stated already in the preliminary version [18, Section 6], there without proof.

Suppose that we have a pushout square of representable signatures,

$$\begin{array}{ccc} \Sigma_0 & \longrightarrow & \Sigma_1 \\ \downarrow & & \downarrow \\ \Sigma_2 & \longrightarrow & \Sigma \end{array}$$

Intuitively, the signatures Σ_1 and Σ_2 specify two extensions of the signature Σ_0 , and Σ is the smallest extension containing both these extensions. Modularity means that the corresponding diagram of representations,

$$\begin{array}{ccc} \hat{\Sigma}_0 & \longrightarrow & \hat{\Sigma}_1 \\ \downarrow & & \downarrow \\ \hat{\Sigma}_2 & \longrightarrow & \hat{\Sigma} \end{array}$$

is a pushout as well – but we have to take care to state this in the “right” category. The right category for this purpose is the following total category $\int_{\Sigma} \text{Mon}^{\Sigma}$ of models:

- An object of $\int_{\Sigma} \text{Mon}^{\Sigma}$ is a triple (Σ, R, r) where Σ is a signature, R is a monad, and r is an action of Σ in R .
- A morphism in $\int_{\Sigma} \text{Mon}^{\Sigma}$ from (Σ_1, R_1, r_1) to (Σ_2, R_2, r_2) consists of a pair (i, m) of a signature morphism $i : \Sigma_1 \rightarrow \Sigma_2$ and a morphism m of Σ_1 -models from (R_1, r_1) to $(R_2, i^*(r_2))$.
- It is easily checked that the obvious composition turns $\int_{\Sigma} \text{Mon}^{\Sigma}$ into a category.

Now for each signature Σ , we have an obvious inclusion from the fiber Mon^{Σ} into $\int_{\Sigma} \text{Mon}^{\Sigma}$, through which we may see the syntax $\hat{\Sigma}$ of any representable signature as an object in $\int_{\Sigma} \text{Mon}^{\Sigma}$. Furthermore, a morphism $i : \Sigma_1 \rightarrow \Sigma_2$ of representable signatures yields a morphism $i_* := \hat{\Sigma}_1 \rightarrow \hat{\Sigma}_2$ in $\int_{\Sigma} \text{Mon}^{\Sigma}$. Hence our pushout square of representable signatures as described above yields a square in $\int_{\Sigma} \text{Mon}^{\Sigma}$.

► **Theorem 32.** *Modularity holds in $\int_{\Sigma} \text{Mon}^{\Sigma}$, in the sense that given a pushout square of representable signatures as above, the associated square in $\int_{\Sigma} \text{Mon}^{\Sigma}$ is a pushout again.*

In particular, the binary coproduct of two signatures Σ_1 and Σ_2 is represented by the binary coproduct of the representations of Σ_1 and Σ_2 .

Our computer-checked proof of modularity is available as `pushout_in_big_rep` in the file `Signatures/Modularity`. The proof uses, in particular, the following fact:

4:10 High-Level Signatures and Initial Semantics

► **Lemma 33.** *The projection $\pi : \int_{\Sigma} \text{Mon}^{\Sigma} \rightarrow \text{Sig}$ is a Grothendieck fibration.*

See `rep_cleaving` in `Signatures.Signature` for the formalized proof.

6 Recursion

We now show through examples how certain forms of recursion can be derived from initiality.

6.1 Example: Translation of intuitionistic logic into linear logic

We start with an elementary example of translation of syntaxes using initiality, namely the translation of second-order intuitionistic logic into second-order linear logic [14, page 6]. The syntax of second-order intuitionistic logic can be defined with one unary operator \neg , three binary operators \vee , \wedge and \Rightarrow , and two binding operators \forall and \exists . The associated (algebraic) signature is $\Sigma_{LK} = \Theta + (3 \times \Theta^2) + (2 \times \Theta')$. As for linear logic, there are four constants \top , \perp , 0 , 1 , two unary operators $!$ and $?$, five binary operators $\&$, \wp , \otimes , \oplus , \multimap and two binding operators \forall and \exists . The associated (algebraic) signature is $\Sigma_{LL} = (4 \times *) + (2 \times \Theta) + (5 \times \Theta^2) + (2 \times \Theta')$.

By universality of the coproduct, a model of Σ_{LK} is given by a monad R with module morphisms:

- $r_{\neg} : R \rightarrow R$
- $r_{\forall}, r_{\exists} : R' \rightarrow R$
- $r_{\wedge}, r_{\vee}, r_{\Rightarrow} : R \times R \rightarrow R$

and similarly, we can decompose an action of Σ_{LL} into as many components as there are operators.

The translation will be a morphism of monads between the initial models (i.e. the syntaxes) $o : \hat{\Sigma}_{LK} \rightarrow \hat{\Sigma}_{LL}$ that further satisfies the properties of a morphism of Σ_{LK} -models, for example $o(r_{\exists}(t)) = r_{\exists}(o(t))$. The strategy is to use the initiality of $\hat{\Sigma}_{LK}$. Indeed, equipping $\hat{\Sigma}_{LL}$ with an action $r'_{\alpha} : \alpha(\hat{\Sigma}_{LL}) \rightarrow \hat{\Sigma}_{LL}$ for each operator α of intuitionistic logic ($\top, \perp, \vee, \wedge, \Rightarrow, \forall, \exists, \in$ and $=$) yields a morphism of monads $o : \hat{\Sigma}_{LK} \rightarrow \hat{\Sigma}_{LL}$ such that $o(r_{\alpha}(t)) = r'_{\alpha}(o(t))$ for each α .

The definition of r'_{α} is then straightforward to devise, following the recursive clauses given on the right:

$$\begin{array}{ll}
 r'_{\neg} = r_{\multimap} \circ (r_! \times r_0) & (\neg A)^{\circ} := (!A) \multimap 0 \\
 r'_{\wedge} = r_{\&} & (A \wedge B)^{\circ} := A^{\circ} \& B^{\circ} \\
 r'_{\vee} = r_{\oplus} \circ (r_! \times r_!) & (A \vee B)^{\circ} := !A^{\circ} \oplus !B^{\circ} \\
 r'_{\Rightarrow} = r_{\multimap} \circ (r_! \times id) & (A \Rightarrow B)^{\circ} := !A^{\circ} \multimap B^{\circ} \\
 r'_{\exists} = r_{\exists} \circ r_! & (\exists x A)^{\circ} := \exists x !A^{\circ} \\
 r'_{\forall} = r_{\forall} & (\forall x A)^{\circ} := \forall x A^{\circ}
 \end{array}$$

The induced action of Σ_{LK} in the monad $\hat{\Sigma}_{LL}$ yields the desired translation morphism $o : \hat{\Sigma}_{LK} \rightarrow \hat{\Sigma}_{LL}$. Note that variables are automatically preserved by the translation because o is a monad morphism.

6.2 Example: Computing the set of free variables

We denote by $P(X)$ the power set of X . The union gives us a composition operator $P(P(X)) \rightarrow P(X)$ defined by $u \mapsto \bigcup_{s \in u} s$, which yields a monad structure on P .

We now define an action of the signature of lambda calculus Σ_{LC} in the monad P . We take union operator $\cup: P \times P \rightarrow P$ as action of the application signature $\Theta \times \Theta$; this is a module morphism since binary union distributes over union of sets. Next, given $s \in P(X + *)$ we define $\text{Maybe}^{-1}(s) = s \cap X$. This defines a morphism of modules $\text{Maybe}^{-1}: P' \rightarrow P$; a small calculation using a distributivity law of binary intersection over union of sets shows that this natural transformation is indeed linear. It can hence be used to model the abstraction signature Θ' in P .

Associated to this model of Σ_{LC} in P we have an initial morphism $\text{free}: \text{LC} \rightarrow P$. Then, for any $t \in \text{LC}(X)$, the set $\text{free}(t)$ is the set of free variables occurring in t .

6.3 Example: Computing the size of a term

We now consider the problem of computing the “size” of a λ -term, that is, for any set X , a function $s_X: \text{LC}(X) \rightarrow \mathbb{N}$ such that

$$\begin{aligned} s_X(x) &= 0 & (x \in X \text{ variable}) \\ s_X(\text{abs}(t)) &= 1 + s_{X+*}(t) \\ s_X(\text{app}(t, u)) &= 1 + s_X(t) + s_X(u) \end{aligned}$$

This problem (and many similar other ones) does not fit directly in our vision because this computation does not commute with substitution, hence does not correspond to a (potentially initial) morphism of monads.

Instead of computing the size of a term (which is 0 for a variable), we compute a generalized size gs which depends on arbitrary (formal) sizes attributed to variables. We have

$$gs: \forall X: \text{Set}, \text{LC}(X) \rightarrow (X \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$$

Here, we recognize the continuation monad (see also [22])

$$\text{Cont}_{\mathbb{N}} := X \mapsto (X \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$$

with multiplication $\lambda f. \lambda g. f(\lambda h. h(g))$. The sets $\text{Cont}_A(\emptyset)$ and A are in natural bijection and we will identify them in what follows.

Now we can define gs through initiality by endowing the monad $\text{Cont}_{\mathbb{N}}$ of a structure of Σ_{LC} -model as follows.

The function $\alpha(m, n) = 1 + m + n$ induces a natural transformation

$$\alpha_+: \text{Cont}_{\mathbb{N}} \times \text{Cont}_{\mathbb{N}} \rightarrow \text{Cont}_{\mathbb{N}}$$

thus an action for the application signature $\Theta \times \Theta$ in the monad $\text{Cont}_{\mathbb{N}}$.

Next, given $f \in \text{Cont}_{\mathbb{N}}(X + *)$, define $f' \in \text{Cont}_{\mathbb{N}}(X)$ by $f'(x) = 1 + f(x)$ for all $x \in X$ and $f'(*) = 0$. This induces a natural transformation

$$\begin{array}{ccc} \beta: \text{Cont}'_{\mathbb{N}} & \longrightarrow & \text{Cont}_{\mathbb{N}} \\ f & \longmapsto & f' \end{array}$$

which is the desired action of the abstraction signature Θ' .

Altogether, we have the desired action of Σ_{LC} in $\text{Cont}_{\mathbb{N}}$ and thus an initial morphism, i.e., a natural transformation $\iota: \text{LC} \rightarrow \text{Cont}_{\mathbb{N}}$ which respects the Σ_{LC} -model structure. Now let 0_X be the identically zero function on X . Then the sought “size” map is given by $s_X(x) = \iota_X(x, 0_X)$.

6.4 Example: Counting the number of redexes

We now consider an example of recursive computation: a function r such that $r(t)$ is the number of redexes of the λ -term t of $\text{LC}(X)$. Informally, the equations defining r are

$$\begin{aligned} r(x) &= 0, & (x \text{ variable}) \\ r(\text{abs}(t)) &= r(t), \\ r(\text{app}(t, u)) &= \begin{cases} 1 + r(t) + r(u) & \text{if } u \text{ is an abstraction} \\ r(t) + r(u) & \text{otherwise} \end{cases} \end{aligned}$$

Here the (standard) recipe is to make the desired function appear as a projection of an iterative function with values in a product. Concretely, we will proceed by first defining a Σ_{LC} -action on the monad product $W := \text{Cont}_{\mathbb{N}} \times \text{LC}$. First, consider the linear morphism $\beta: \text{Cont}'_{\mathbb{N}} \rightarrow \text{Cont}_{\mathbb{N}}$ given by $\beta(f)(x) = f(x)$ for all $f \in \text{Cont}_{\mathbb{N}}(X + *)$ and $x \in X$. Since we have $W' = \text{Cont}'_{\mathbb{N}} \times \text{LC}'$, the product

$$\beta \times \text{abs}: W' \longrightarrow W$$

is an action of the abstraction signature Θ' in W .

Next we specify the action of the application signature $\Theta \times \Theta$. Given $((u, s), (v, t)) \in W(X) \times W(X)$ and $k: X \rightarrow A$ we define

$$c((u, s), (v, t)) := \begin{cases} (1 + u(k) + v(k))(k) & \text{if } t \text{ is an abstraction} \\ (u(k) + v(k))(k) & \text{otherwise} \end{cases}$$

and

$$a((u, s), (v, t)) := \text{app}(s, t)$$

The pair map $(c, a): W \times W \rightarrow W$ is our action of app in W .

From this Σ_{LC} -action, we get an initial morphism $\iota: \text{LC} \rightarrow \text{Cont}_{\mathbb{N}} \times \text{LC}$. The second component of ι is nothing but the identity morphism. By taking the projection on the first component, we find a module morphism $\pi_1 \cdot \iota: \text{LC} \rightarrow \text{Cont}_{\mathbb{N}}$. Finally, if 0_X is the constant function $X \rightarrow \mathbb{N}$ returning zero, then $\pi_1(\iota(0_X)): \text{LC}(X) \rightarrow \mathbb{N}$ is the desired function r .

7 Presentable signatures and syntaxes

In this section, we identify a superclass of algebraic signatures that are still representable: we call them *presentable* signatures.

► **Definition 34.** A signature Σ is *presentable*¹² if there is an algebraic signature Υ and an epimorphism of signatures $p: \Upsilon \rightarrow \Sigma$.

► **Remark.** By definition, any construction which can be encoded through a presentable signature can alternatively be encoded through the “presenting” algebraic signature. The former encoding is finer than the latter in the sense that terms which are different in the latter encoding can be identified by the former. In other words, a certain amount of semantics is integrated into the syntax.

¹²In algebra, a presentation of a group G is an epimorphism $F \rightarrow G$ where F is free (together with a generating set of relations among the generators).

The main desired property of our presentable signatures is that, thanks to the following theorem, they are representable:

► **Theorem 35.** *Any presentable signature is representable.*

A sketch of the proof is available in Appendix A.

See `PresentableIsRepresentable` in `Signatures/PresentableSignature` for the formalized proof.

► **Definition 36.** We call a syntax *presentable* if it is generated by a presentable signature.

Next, we give important examples of presentable signatures:

► **Theorem 37.** *The following hold:*

1. *Any algebraic signature is presentable.*
2. *Any colimit of presentable signatures is presentable.*
3. *The product of two presentable signatures is presentable.*
(`Signatures/PresentableSignatureBinProdR:har_binprodR_isPresentable` in the case when one of them is Θ).

Proof. Items 1–2 are easy to prove. For Item 3, if Σ_1 and Σ_2 are presented by $\coprod_i \Upsilon_i$ and $\coprod_j \Phi_j$ respectively, then $\Sigma_1 \times \Sigma_2$ is presented by $\coprod_{i,j} \Upsilon_i \times \Phi_j$. ◀

► **Corollary 38.** *Any colimit of algebraic signatures is representable.*

8 Examples of presentable signatures

In this section we present various constructions which, thanks to Theorem 35, can be “safely” added to a presentable syntax. *Safely* here means that the resulting signature is still presentable.

8.1 Example: Adding a syntactic binary commutative operator

Here we present a signature that could be used to formalize a binary commutative operator, for example the addition of two numbers. The elementary signature $\Theta \times \Theta$ already provides a way to extend the syntax with a constructor with two arguments. By quotienting this signature, we can enforce commutativity. To this end, consider the signature $\mathcal{S}_2 \cdot \Theta$ (see Example 16) where \mathcal{S}_2 is the endofunctor that assigns to each set X the set of its unordered pairs. It is presentable because the epimorphism between the square endofunctor $\Delta = X \mapsto X \times X$ and \mathcal{S}_2 yields an epimorphism from $\Delta \cdot \Theta \cong \Theta \times \Theta$ to $\mathcal{S}_2 \cdot \Theta$. This signature could alternatively be defined as the coequalizer of the identity morphism and the signature morphism `swap` : $\Theta \times \Theta \rightarrow \Theta \times \Theta$ that exchanges the first and the second projection.

An action of the signature $\mathcal{S}_2 \cdot \Theta$ in a monad R is given by an operation on unordered pairs of elements of $R(X)$ for any set X , or equivalently, thanks to the universal property of the quotient, by a module morphism $m : R^2 \rightarrow R$ such that, for any set X and $a, b \in R(X)$, $m_X(a, b) = m_X(b, a)$.

8.2 Example: Adding a syntactic closure operator

Given a quantification construction (e.g., abstraction, universal or existential quantification), it is often useful to take the associated closure operation. One well-known example is the universal closure of a logic formula. Such a closure is invariant under permutation of the

fresh variables. A closure can be syntactically encoded in a rough way by iterating the closure with respect to one variable at a time. Here our framework allows a refined syntactic encoding which we explain below.

Let us start with binding a fixed number k of fresh variables. The elementary signature $\Theta^{(k)}$ already specifies an operation that binds k variables. However, this encoding does not reflect invariance under variable permutation. To enforce this invariance, it suffices to quotient the signature $\Theta^{(k)}$ with respect to the action of the group S_k of permutations of the set k , that is, to consider the colimit of the following one-object diagram:

$$\begin{array}{c} \Theta^{(\sigma)} \\ \curvearrowright \\ \Theta^{(k)} \end{array}$$

where σ ranges over the elements of S_k . We denote by $\mathcal{S}^{(k)}\Theta$ the resulting (presentable) signature. By universal property of the quotient, a model of it consists of a monad R with an action $m : R^{(k)} \rightarrow R$ that satisfies the required invariance.

Now, we want to specify an operation which binds an arbitrary number of fresh variables, as expected from a closure operator. One rough solution is to consider the coproduct $\coprod_k \mathcal{S}^{(k)}\Theta$. However, we encounter a similar inconvenience as for $\Theta^{(k)}$. Indeed, for each $k' > k$, each term already encoded by the signature $\mathcal{S}^{(k)}\Theta$ may be considered again, encoded (differently) through $\mathcal{S}^{(k')}\Theta$.

Fortunately, a finer encoding is provided by the following simple colimit of presentable signatures. The crucial point here is that, for each k , all natural injections from $\Theta^{(k)}$ to $\Theta^{(k+1)}$ induce the same canonical injection from $\mathcal{S}^{(k)}\Theta$ to $\mathcal{S}^{(k+1)}\Theta$. We thus have a natural colimit for the sequence $k \mapsto \mathcal{S}^{(k)}\Theta$ and thus a signature $\text{colim}_k \mathcal{S}^{(k)}\Theta$ which, as a colimit of presentable signatures, is presentable (Theorem 37, item 2).

Accordingly, we define a total closure on a monad R to be an action of the signature $\text{colim}_k \mathcal{S}^{(k)}\Theta$ in R . It can easily be checked that a model of this signature is a monad R together with a family of module morphisms $(e_k : R^{(k)} \rightarrow R)_{k \in \mathbb{N}}$ compatible in the sense that for each injection $i : k \rightarrow k'$ the following diagram commutes:

$$\begin{array}{ccc} R^{(k)} & \xrightarrow{R^{(i)}} & R^{(k')} \\ & \searrow e_k & \downarrow e_{k'} \\ & & R \end{array}$$

8.3 Example: Adding an explicit substitution

In this section, we explain how we can extend any presentable signature with an *explicit substitution* construction. In fact we will show three solutions, differing in the amount of “coherence” which is handled at the syntactic level (e.g., invariance under permutation and weakening). We follow the approach initiated by Ghani, Uustalu, and Hamana in [13].

Let R be a monad. We have already considered (see Lemma 12) the (unary) substitution $\sigma_R : R' \times R \rightarrow R$. More generally, we have the sequence of substitution operations

$$\text{subst}_p : R^{(p)} \times R^p \longrightarrow R. \tag{2}$$

We say that subst_p is the p -substitution in R ; it simultaneously replaces the p extra variables in its first argument with the p other arguments, respectively. (Note that subst_1 is the original σ_R).

We observe that, for fixed p , the group S_p of permutations on p elements has a natural action on $R^{(p)} \times R^p$, and that subst_p is invariant under this action.

Thus, if we fix an integer p , there are two ways to internalize subst_p in the syntax: we can choose the elementary signature $\Theta^{(p)} \times \Theta^p$, which is rough in the sense that the above invariance is not reflected; and alternatively, if we want to reflect the permutation invariance syntactically, we can choose the quotient Q_p of the above signature by the action of S_p .

By universal property of the quotient, a model of our quotient Q_p is given by a monad R with an action $m : R^{(p)} \times R^p \rightarrow R$ satisfying the desired invariance.

Before turning to the encoding of the entire series $(\text{subst}_p)_{p \in \mathbb{N}}$, we recall how, as noticed already in [13], this series enjoys further coherence. In order to explain this coherence, we start with two natural numbers p and q and the module $R^{(p)} \times R^q$. Pairs in this module are almost ready for substitution: what is missing is a map $u : I_p \rightarrow I_q$. But such a map can be used in two ways: letting u act covariantly on the first factor leads us into $R^{(q)} \times R^q$ where we can apply subst_q ; while letting u act contravariantly on the second factor leads us into $R^{(p)} \times R^p$ where we can apply subst_p . The good news is that we obtain the same result. More precisely, the following diagram is commutative:

$$\begin{array}{ccc}
 R^{(p)} \times R^q & \xrightarrow{R^{(p)} \times R^u} & R^{(p)} \times R^p \\
 \begin{array}{c} R^{(u)} \times R^p \\ \downarrow \end{array} & & \begin{array}{c} \downarrow \text{subst}_p \\ R \end{array} \\
 R^{(q)} \times R^q & \xrightarrow{\text{subst}_q} & R
 \end{array} \tag{3}$$

Note that in the case where p equals q and u is a permutation, we recover exactly the invariance by permutation considered earlier.

Abstracting over the numbers p, q and the map u , this exactly means that our series factors through the coend $\int^{p:\mathbb{N}} R^{(p)} \times R^{\bar{p}}$, where covariant (resp. contravariant) occurrences of the bifunctor have been underlined (resp. overlined), and the category \mathbb{N} is the full subcategory of Set whose objects are natural numbers. Thus we have a canonical morphism

$$\text{isubst}_R : \int^{p:\mathbb{N}} R^{(\underline{p})} \times R^{\bar{p}} \longrightarrow R.$$

Abstracting over R , we obtain the following:

► **Definition 39.** The *integrated substitution*

$$\text{isubst} : \int^{p:\mathbb{N}} \Theta^{(\underline{p})} \times \Theta^{\bar{p}} \longrightarrow \Theta$$

is the signature morphism obtained by abstracting over R the linear morphisms isubst_R .

Thus, if we want to internalize the whole sequence $(\text{subst}_p)_{p \in \mathbb{N}}$ in the syntax, we have at least three solutions: we can choose the algebraic signature $\coprod_{p:\mathbb{N}} \Theta^{(p)} \times \Theta^p$, which is rough in the sense that the above invariance and coherence is not reflected; we can choose the presentable signature $\coprod_{p:\mathbb{N}} Q_p$, which reflects the invariance by permutation, but not more; and finally, if we want to reflect the whole coherence syntactically, we can choose the presentable signature $\int^{p:\mathbb{N}} \Theta^{(\underline{p})} \times \Theta^{\bar{p}}$.

Thus, whenever a signature is presentable, we can safely extend it by adding one or the other of the three above signatures, for a (more or less coherent) explicit substitution.

Ghani, Uustalu, and Hamana already studied this problem in [13]. Our solution proposed here does not require the consideration of a *strength*.

8.4 Example: Adding a coherent fixed point operator

In the same spirit as in the previous section, we define, in this section,

- for each $n \in \mathbb{N}$, a notion of n -ary fixed point operator in a monad;
- a notion of *coherent fixed point operator* in a monad, which assigns, in a “coherent” way, to each $n \in \mathbb{N}$, an n -ary fixed point operator.

We furthermore explain how to safely extend any presentable syntax with a syntactic coherent fixed point operator.

There is one fundamental difference between the integrated substitution of the previous section and our coherent fixed points: while every monad has a canonical integrated substitution, this is not the case for coherent fixed point operators.

Let us start with the unary case.

► **Definition 40.** A *unary fixed point operator* for a monad R is a module morphism f from R' to R that makes the following diagram commute,

$$\begin{array}{ccc} R' & \xrightarrow{(id_{R'}:f)} & R' \times R \\ & \searrow f & \swarrow \sigma \\ & & R \end{array}$$

where σ is the substitution morphism defined in Lemma 12.

Accordingly, the signature for a syntactic unary fixpoint operator is Θ' , ignoring the commutation requirement (which we plan to address in a future work by extending our framework with equations).

Let us digress here and examine what the unary fixpoint operators are for the lambda calculus, more precisely, for the monad $\text{LC}_{\beta\eta}$ of the lambda-calculus modulo β - and η -equivalence. How can we relate the above notion to the classical notion of fixed-point combinator? Terms are built out of two constructions, $\text{app} : \text{LC}_{\beta\eta} \times \text{LC}_{\beta\eta} \rightarrow \text{LC}_{\beta\eta}$ and $\text{abs} : \text{LC}'_{\beta\eta} \rightarrow \text{LC}_{\beta\eta}$. A fixed point combinator is a term Y satisfying, for any (possibly open) term t , the equation

$$\text{app}(t, \text{app}(Y, t)) = \text{app}(Y, t).$$

Given such a combinator Y , we define a module morphism $\hat{Y} : \text{LC}'_{\beta\eta} \rightarrow \text{LC}_{\beta\eta}$. It associates, to any term t depending on an additional variable $*$, the term $\hat{Y}(t) := \text{app}(Y, \text{abs } t)$. This term satisfies $t[\hat{Y}(t)/*] = \hat{Y}(t)$, which is precisely the diagram of Definition 40 that \hat{Y} must satisfy to be a unary fixed point operator for the monad $\text{LC}_{\beta\eta}$. Conversely, we have:

► **Proposition 41.** Any fixed point combinator in $\text{LC}_{\beta\eta}$ comes from a unique fixed point operator.

The proof can be found in Appendix B.

After this digression, we now turn to the n -ary case.

► **Definition 42.**

- A *rough n -ary fixed point operator* for a monad R is a module morphism $f : (R^{(n)})^n \rightarrow R^n$ making the following diagram commute:

$$\begin{array}{ccc} (R^{(n)})^n & \xrightarrow{id_{(R^{(n)})^n}.f, \dots, f} & (R^{(n)})^n \times (R^n)^n \\ f \downarrow & & \parallel \\ R^n & \xleftarrow{(\text{subst}_n)^n} & (R^{(n)} \times R^n)^n \end{array}$$

where subst_n is the n -substitution as in Section 8.3.

- An *n*-ary fixed point operator is just a rough *n*-ary fixed point operator which is furthermore invariant under the natural action of the permutation group S_n .

The type of f above is canonically isomorphic to

$$(R^{(n)})^n + (R^{(n)})^n + \dots + (R^{(n)})^n \rightarrow R,$$

which we abbreviate to¹³ $n \times (R^{(n)})^n \rightarrow R$.

Accordingly, a natural signature for encoding a syntactic rough *n*-ary fixpoint operator is $n \times (\Theta^{(n)})^n$.

Similarly, a natural signature for encoding a syntactic *n*-ary fixpoint operator is $(n \times (\Theta^{(n)})^n)/S_n$ obtained by quotienting the previous signature by the action of S_n .

Now we let n vary and say that a *total fixed point operator* on a given monad R assigns to each $n \in \mathbb{N}$ an *n*-ary fixpoint operator on R . Obviously, the natural signature for the encoding of a syntactic total fixed point operator is $\coprod_n (\Theta^{(n)})^n/S_n$. Alternatively, we may wish to discard those total fixed point operators that do not satisfy some coherence conditions analogous to what we encountered in Section 8.3, which we now introduce.

Let R be a monad with a sequence of module morphisms $\text{fix}_n : n \times (R^{(n)})^n \rightarrow R$. We call this family *coherent* if, for any $p, q \in \mathbb{N}$ and $u : p \rightarrow q$, the following diagram commutes:

$$\begin{array}{ccc} p \times (R^{(p)})^q & \xrightarrow{p \times (R^{(p)})^u} & p \times (R^{(p)})^p \\ u \times (R^{(u)})^q \downarrow & & \downarrow \text{fix}_p \\ q \times (R^{(q)})^q & \xrightarrow{\text{fix}_q} & R \end{array} \quad (4)$$

These conditions have an interpretation in terms of a coend, just as we already encountered in Section 8.3. This leads us to the following

► **Definition 43.** Given a monad R , we define a *coherent fixed point operator on R* to be a module morphism from $\int^{n:\mathbb{N}} \underline{n} \times (R^{(\underline{n})})^{\bar{n}}$ to R where, for every $n \in \mathbb{N}$, the n -th component is a (rough)¹⁴ *n*-ary fixpoint operator.

Now, the natural signature for a syntactic coherent fixed point operator is $\int^{n:\mathbb{N}} \underline{n} \times (\Theta^{(\underline{n})})^{\bar{n}}$. Thus, given a presentable signature Σ , we can safely extend it with a syntactic coherent fixed point operator by adding the presentable signature $\int^{n:\mathbb{N}} \underline{n} \times (\Theta^{(\underline{n})})^{\bar{n}}$ to Σ .

9 Conclusions and future work

We have presented notions of *signature* and *model of a signature*. A signature is said to be *representable* when its category of models has an initial model. We have defined a class of *presentable* signatures, which contains traditional algebraic signatures, and which is closed under various operations, including colimits. Our main result says that any presentable signature is representable.

One difference to other work on Initial Semantics, e.g., [24, 12, 7, 9], is that we do not rely on the notion of strength. However, a signature endofunctor with strength as used in the aforementioned articles can be translated to a high-level signature as presented in this work. In future work, we will show that this translation extends faithfully to models of signatures, and preserves initiality.

¹³In the following, we similarly write n instead of I_n in order to make equations more readable.

¹⁴As in Section 8.3, the invariance follows from the coherence.

Furthermore, we plan to generalize our representability criterion to encompass *explicit join* (see [24]); to generalize our notions of signature and models to (*simply-*)*typed* syntax; and to provide a systematic approach to *equations* for our notion of signature and models.

References

- 1 Benedikt Ahrens. Modules over relative monads for syntax and semantics. *Mathematical Structures in Computer Science*, 26:3–37, 2016. doi:10.1017/S0960129514000103.
- 2 Benedikt Ahrens and Peter LeFanu Lumsdaine. Displayed Categories. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction*, volume 84 of *Leibniz International Proceedings in Informatics*, pages 5:1–5:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSCD.2017.5.
- 3 Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. *Logical Methods in Computer Science*, 11(1), 2015. doi:10.2168/LMCS-11(1:3)2015.
- 4 Thorsten Altenkirch and Bernhard Reus. Monadic presentations of lambda terms using generalized inductive types. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 1999. doi:10.1007/3-540-48168-0_32.
- 5 Richard S. Bird and Oege de Moor. *Algebra of programming*. Prentice Hall International series in computer science. Prentice Hall, 1997.
- 6 Richard S. Bird and Ross Paterson. Generalised folds for nested datatypes. *Formal Asp. Comput.*, 11(2):200–222, 1999. doi:10.1007/s001650050047.
- 7 Marcelo P. Fiore. Second-order and dependently-sorted abstract syntax. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 57–68. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.38.
- 8 Marcelo P. Fiore and Chung-Kil Hur. Second-order equational logic (extended abstract). In Anuj Dawar and Helmut Veith, editors, *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2010. doi:10.1007/978-3-642-15205-4_26.
- 9 Marcelo P. Fiore and Ola Mahmoud. Second-order algebraic theories - (extended abstract). In Petr Hliněný and Antonín Kucera, editors, *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 368–380. Springer, 2010. doi:10.1007/978-3-642-15155-2_33.
- 10 Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 193–202, 1999. doi:10.1109/LICS.1999.782615.
- 11 Murdoch J. Gabbay and Andrew M. Pitts. A New Approach to Abstract Syntax Involving Binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224, Washington, DC, USA, 1999. IEEE Computer Society Press. doi:10.1109/LICS.1999.782617.
- 12 Neil Ghani and Tarmo Uustalu. Explicit substitutions and higher-order syntax. In *MERLIN '03: Proceedings of the 2003 ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding*, pages 1–7, New York, NY, USA, 2003. ACM Press.
- 13 Neil Ghani, Tarmo Uustalu, and Makoto Hamana. Explicit substitutions and higher-order syntax. *Higher-Order and Symbolic Computation*, 19(2-3):263–282, 2006. doi:10.1007/s10990-006-8748-4.
- 14 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50(1):1–102, 1987. doi:10.1016/0304-3975(87)90045-4.
- 15 Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, jan 1993. doi:10.1145/138027.138060.

- 16 André Hirschowitz and Marco Maggesi. Modules over monads and linearity. In D. Leivant and R. J. G. B. de Queiroz, editors, *WoLLIC*, volume 4576 of *Lecture Notes in Computer Science*, pages 218–237. Springer, 2007. doi:10.1007/978-3-540-73445-1_16.
- 17 André Hirschowitz and Marco Maggesi. Modules over monads and initial semantics. *Information and Computation*, 208(5):545–564, May 2010. Special Issue: 14th Workshop on Logic, Language, Information and Computation (WoLLIC 2007). doi:10.1016/j.ic.2009.07.003.
- 18 André Hirschowitz and Marco Maggesi. Initial semantics for strengthened signatures. In Dale Miller and Ésik Zoltán, editors, *Proceedings of the 8th Workshop on Fixed Points in Computer Science*, pages 31–38, 2012. doi:10.4204/EPTCS.77.
- 19 Tom Hirschowitz. Cartesian closed 2-categories and permutation equivalence in higher-order rewriting. *Logical Methods in Computer Science*, 9(3):10, 2013. 19 pages. doi:10.2168/LMCS-9(3:10)2013.
- 20 Martin Hyland and John Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electronic Notes in Theoretical Computer Science*, 172:437–458, April 2007. doi:10.1016/j.entcs.2007.02.019.
- 21 J.W. Thatcher J.A. Goguen and E.G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology, IV: Data Structuring*, pages 80–144. Prentice-Hall, 1978.
- 22 Patricia Johann and Neil Ghani. Initial algebra semantics is enough! In *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, pages 207–222, 2007. doi:10.1007/978-3-540-73228-0_16.
- 23 Saunders Mac Lane. *Categories for the working mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1998.
- 24 Ralph Matthes and Tarmo Uustalu. Substitution in non-wellfounded syntax with variable binding. *Theor. Comput. Sci.*, 327(1-2):155–174, 2004. doi:10.1016/j.tcs.2004.07.025.
- 25 The Coq development team. *The Coq Proof Assistant, version 8.8.0*, 2018. Version 8.8. URL: <http://coq.inria.fr>.
- 26 Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. UniMath — a computer-checked library of univalent mathematics. Available at <https://github.com/UniMath/UniMath>.

A Proof of Theorem 35

The results of this section, as well as Theorem 35 for which these results are used, are mechanically checked in our library; the reader may thus prefer to check the formalized statements in the library rather than their proofs in this section.

The proof of Theorem 35 rests on the more technical Lemma 48 below, which requires the notion of *epi-signature*:

► **Definition 44.** An *epi-signature* is a signature Σ that preserves the epimorphicity in the category of endofunctors on Set : for any monad morphism $f : R \rightarrow S$, if $U(f)$ is an epi of functors, then so is $U(\Sigma(f))$. Here, we denote by U the forgetful functor from monads resp. modules to the underlying endofunctors.

► **Example 45.** Any algebraic signature is an epi-signature.

This example is formalized in `Signatures/BindingSig:BindingSigAreEpiSig`.

► **Proposition 46.** *Epimorphisms of signatures are pointwise epimorphisms.*

4:20 High-Level Signatures and Initial Semantics

Proof. The proof is formalized in `Signatures/EpiArePointwise:epiSig_is_pwEpi`. In any category, a morphism $f : a \rightarrow b$ is an epimorphism if and only if the following diagram is a pushout diagram ([23, exercise III.4.4]) :

$$\begin{array}{ccc} a & \xrightarrow{f} & b \\ f \downarrow & & \downarrow id \\ b & \xrightarrow{id} & b \end{array}$$

Using this characterization of epimorphisms, the proof follows from the fact that colimits are computed pointwise in the category of signatures. \blacktriangleleft

Another important ingredient will be the following quotient construction for monads. Let R be a monad, and let \sim be a “compatible” family of relations on (the functor underlying) R , that is, for any $X : \mathbf{Set}_0$, \sim_X is an equivalence relation on RX such that, for any $f : X \rightarrow Y$, the function $R(f)$ maps related elements in RX to related elements in RY . Taking the pointwise quotient, we obtain a quotient $\pi : R \rightarrow \bar{R}$ in the functor category, satisfying the usual universal property. We want to equip \bar{R} with a monad structure that upgrades $\pi : R \rightarrow \bar{R}$ into a quotient in the category of monads. In particular, this means that we need to fill in the square

$$\begin{array}{ccc} R \cdot R & \xrightarrow{\mu} & R \\ \pi \cdot \pi \downarrow & & \downarrow \pi \\ \bar{R} \cdot \bar{R} & \xrightarrow{\bar{\mu}} & \bar{R} \end{array}$$

with a suitable $\bar{\mu} : \bar{R} \cdot \bar{R} \rightarrow \bar{R}$ satisfying the monad laws. But since π , and hence $\pi \cdot \pi$, is epi, this is possible when any two elements in RRX that are mapped to related elements by $\pi \cdot \pi$ (the left vertical morphism) are also mapped to related elements by $\pi \circ \mu$ (the top-right composition). It turns out that this is the only extra condition needed for the upgrade. We summarize the construction in the following lemma:

► **Lemma 47.** *Given a monad R , and a compatible relation \sim on R such that for any set X and $x, y \in RRX$, we have that if $(\pi \cdot \pi)_X(x) \sim (\pi \cdot \pi)_X(y)$ then $\pi(\mu(x)) \sim \pi(\mu(y))$. Then we can construct the quotient $\pi : R \rightarrow \bar{R}$ in the category of monads, satisfying the usual universal property.*

We are now in a position to state and prove the main technical lemma:

► **Lemma 48.** *Let Υ be a representable signature. Let $F : \Upsilon \rightarrow \Sigma$ be a morphism of signatures. Suppose that Υ is an epi-signature and F is an epimorphism. Then Σ is representable.*

Sketch of the proof. We denote by R the initial Υ -model, as well as $-$ by abuse of notation $-$ its underlying monad. For each set X , we consider the equivalence relation \sim_X on $R(X)$ defined as follows: for all $x, y \in R(X)$ we stipulate that $x \sim_X y$ if and only if $i_X(x) = i_X(y)$ for each (initial) morphism of Υ -models $i : R \rightarrow F^*S$ with S a Σ -model and F^*S the Υ -model induced by $F : \Upsilon \rightarrow \Sigma$.

Per Lemma 47 we obtain the quotient monad, which we call R/F , and the epimorphic projection $\pi : R \rightarrow R/F$. We now equip R/F with a Σ -action, and show that the induced model is initial, in four steps:

- (i) We equip R/F with a Σ -action, i.e., with a morphism of R/F -modules $m_{R/F} : \Sigma(R/F) \rightarrow R/F$. We define $u : \Upsilon(R) \rightarrow \Sigma(R/F)$ as $u = F_{R/F} \circ \Upsilon(\pi)$. Then u is epimorphic, by composition of epimorphisms and by using Corollary 46. Let $m_R : \Upsilon(R) \rightarrow R$ be the action of the initial model of Υ . We define $m_{R/F}$ as the unique morphism making the following diagram commute in the category of endofunctors on Set:

$$\begin{array}{ccc} \Upsilon(R) & \xrightarrow{m_R} & R \\ u \downarrow & & \downarrow \pi \\ \Sigma(R/F) & \xrightarrow{m_{R/F}} & R/F \end{array}$$

Uniqueness is given by the pointwise surjectivity of u . Existence follows from the compatibility of m_R with the congruence \sim_X . The diagram necessary to turn $m_{R/F}$ into a module morphism on R/F is proved by pre-composing it with the epimorphism $\pi \cdot (\Sigma(\pi) \circ F_S)$ and unfolding the definitions.

- (ii) Now, π can be seen as a morphism of Υ -models between R and F^*R/F , by naturality of F and using the previous diagram. It remains to show that $(R/F, m_{R/F})$ is initial in the category of Σ -models.
- (iii) Given a Σ -model (S, m_s) , the initial morphism of Υ -models $i_S : R \rightarrow F^*S$ induces a monad morphism $\iota_S : R/F \rightarrow S$. We need to show that the morphism ι is a morphism of Σ -models. Pre-composing the involved diagram by the epimorphism $\Sigma(\pi)F_R$ and unfolding the definitions shows that $\iota_S : R/F \rightarrow S$ is a morphism of Σ -models.
- (iv) We show that ι_S is the only morphism $R/F \rightarrow S$. Let g be such a morphism. Then $g \circ \pi : R \rightarrow S$ defines a morphism in the category of Υ -models. Uniqueness of i_S yields $g \circ \pi = i_S$, and by uniqueness of the diagram defining ι_S it follows that $g = i'_S$. ◀

In the formalization, this result is derived from the existence of a left adjoint to the pullback functor F^* from Σ -models to Υ -models. The right adjoint is constructed in `is_right_adjoint_functor_of_reps_from_pw_epi` in `Signatures/EpiSigRepresentability`, and transfer of representability is shown in `push_initiality` in the same file.

Proof of Thm. 35. Let Σ be presentable. We need to show that Σ is representable. By hypothesis, we have a presenting algebraic signature Υ and an epimorphism of signatures $e : \Upsilon \rightarrow \Sigma$.

As the signature Υ is algebraic, it is representable (by Theorem 31) and is an epi-signature (by Example 45). We can thus instantiate Lemma 48 to deduce representability of Σ . ◀

B Miscellaneous

Proof of Prop. 41. We construct a bijection between the set $\text{LC}_{\beta\eta}\emptyset$ of closed terms on the one hand and the set of module morphisms from $\text{LC}'_{\beta\eta}$ to $\text{LC}_{\beta\eta}$ satisfying the fixed point property on the other hand.

A closed lambda term t is mapped to the morphism $u \mapsto \hat{t}u := \text{app}(t, \text{abs } u)$. We have already seen that if t is a fixed point combinator, then \hat{t} is a fixed point operator.

For the inverse function, note that a module morphism f from $\text{LC}'_{\beta\eta}$ to $\text{LC}_{\beta\eta}$ induces a closed term $Y_f := \text{abs}(f_1(\text{app}(*, **)))$ where $f_1 : \text{LC}_{\beta\eta}(\{*, **\}) \rightarrow \text{LC}_{\beta\eta}\{*\}$.

A small calculation shows that $Y \mapsto \hat{Y}$ and $f \mapsto Y_f$ are inverse to each other.

4:22 High-Level Signatures and Initial Semantics

It remains to be proved that if f is a fixed point operator, then Y_f satisfies the fixed point combinator equation. Let $t \in \text{LC}_{\beta\eta}X$, then we have

$$\text{app}(Y_f, t) = \text{app}(\text{abs } f_1(\text{app}(*, **)), t) \tag{5}$$

$$= f_X(\text{app}(t, **)) \tag{6}$$

$$= \text{app}(t, \text{app}(Y_f, t)) \tag{7}$$

where (6) comes from the definition of a fixed point operator. Equality (7) follows from the equality $\text{app}(Y_f, t) = f_X(\text{app}(t, **))$, which is obtained by chaining the equalities from (5) to (6). This concludes the construction of the bijection. ◀

The True Concurrency of Herbrand's Theorem

Aurore Alcolei

Univ Lyon, ENS de Lyon, CNRS, UCB Lyon 1, LIP
Lyon, France
Aurore.Alcolei@ens-lyon.fr

Pierre Clairambault

Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP
Lyon, France
Pierre.Clairambault@ens-lyon.fr

Martin Hyland

DPMMS, University of Cambridge
Cambridge, United Kingdom
M.Hyland@dpmms.cam.ac.uk

Glynn Winskel

Computer Laboratory, University of Cambridge
Cambridge, United Kingdom
Glynn.Winskel@cl.cam.ac.uk

Abstract

Herbrand's theorem, widely regarded as a cornerstone of proof theory, exposes some of the constructive content of classical logic. In its simplest form, it reduces the validity of a first-order purely existential formula to that of a finite disjunction. In the general case, it reduces first-order validity to propositional validity, by understanding the structure of the assignment of first-order terms to existential quantifiers, and the causal dependency between quantifiers.

In this paper, we show that Herbrand's theorem in its general form can be elegantly stated and proved as a theorem in the framework of concurrent games, a denotational semantics designed to faithfully represent causality and independence in concurrent systems, thereby exposing the concurrency underlying the computational content of classical proofs. The causal structure of concurrent strategies, paired with annotations by first-order terms, is used to specify the dependency between quantifiers implicit in proofs. Furthermore concurrent strategies can be composed, yielding a compositional proof of Herbrand's theorem, simply by interpreting classical sequent proofs in a well-chosen denotational model.

2012 ACM Subject Classification Theory of computation → Proof theory, Theory of computation → Denotational semantics

Keywords and phrases Herbrand's theorem, Game semantics, True concurrency

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.5

Acknowledgements We acknowledge support of the French LABEX MILYON (ANR-10-LABX-0070), the ERC Advanced Grant ECSYM and the Collegium de Lyon.



© Aurore Alcolei, Pierre Clairambault, Martin Hyland, and Glynn Winskel;
licensed under Creative Commons License CC-BY

27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 5; pp. 5:1–5:22

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

“What more do we know when we have proved a theorem
by restricted means than if we merely know it is true?”

Kreisel's question is the driving force for much modern Proof Theory. This paper is concerned with Herbrand's Theorem, perhaps the earliest result in that direction. It is a simple consequence of completeness and compactness in first-order logic. So it is an example of information being extracted from the bare fact of provability. Usually by contrast one thinks in terms of extracting information from the proofs themselves, typically - as in Kohlenbach's proof mining - via some form of functional interpretation. This has the advantage that information is extracted compositionally in the spirit of functional programming. Specifically information for $\vdash A$ and $\vdash A \rightarrow B$ can be composed to give information for $\vdash B$; or, in terms of the sequent calculus, we can interpret the cut rule.

It seems to be folklore that there is a problem for Herbrand's Theorem. That is made precise in Kohlenbach [17] which shows that one cannot hope directly to use collections of Herbrand terms for $\vdash A$ and $\vdash A \rightarrow B$ to give a collection for $\vdash B$. That leaves the possibility of making some richer data compositional, realised indirectly in Gerhardy and Kohlenbach [11] with data provided by Shoenfield's version [30] of Gödel's Dialectica Interpretation [14]. Now functional interpretations make no pretence to be faithful to the structure of proofs as encapsulated in systems like the sequent calculus: they explore in a sequential order terms proposed by a proof as witnesses for existential quantifiers, but this order is certainly not intrinsic to the proof. Thus it is compelling to seek some compositional form of Herbrand's Theorem faithful to the structure of proofs and to the dependency between terms; for cut-free proofs, Miller's *expansion trees* [24] capture precisely this “Herbrand content” (the information pertaining to quantifier instantiations), but not compositionally.

In this paper, we provide such a compositional form of Herbrand's theorem, presented as a game semantics for first-order classical logic. Our games have two players, both playing on the quantifiers of a formula φ . Eloïse, playing the existential quantifiers, defends the validity of φ . \forall bélar, playing the universal quantifiers, attempts to falsify it. This understanding of formulas as games is folklore in mathematical logic and computer science. However, like functional interpretations, such games are usually sequential [7, 19]. In contrast, our model captures the exact dependence and independence between quantifiers. To achieve that we build on *concurrent/asynchronous* games [23, 27, 4], which marry game semantics with the so-called *true concurrency* approach to models of concurrent systems, and avoid interleavings. So in a formal sense, our model highlights a parallelism inherent to classical proofs. In essence, our strategies are close to expansion trees enriched with an explicit acyclicity witness.

The computational content of classical logic is a longstanding active topic, with a wealth of related works, and it is hard to do it justice in this short introduction. There are, roughly speaking, two families of approaches. On the one hand, some (including the functional interpretations mentioned above) extract from proofs a sequential procedure, *e.g.* via translation to sequential calculi or by annotating a proof to sequentialize or determinize its behaviour under cut reduction [13, 8]. Other than that cited above, influential developments in this “polarized” approach include work by Berardi [2], Coquand [7], Parigot [26], Krivine [18], and others. Polarization yields better-behaved dynamics and a non-degenerate equational theory, but distorts the intent of the proof by an added unintended sequentiality. On the other hand, some works avoid polarization – including, of course, Gentzen's *Hauptsatz* [10]. This causes issues, notably unrestricted cut reduction yields a degenerate equational theory [13] and enjoys only *weak*, rather than *strong*, normalization [8]. Nevertheless, witness extraction

remains possible (though it is non-deterministic). Particularly relevant to our endeavour is a recent activity around the matter of enriching expansion trees so as to support cuts. This includes Heijltjes' *proof forests* [15], McKinley's *Herbrand nets* [21], and Hetzl and Weller's recent *expansion trees with cuts* [16]. In all three cases, a generalization of expansion trees allowing cuts is given along with a weakly normalizing cut reduction procedure. Intuitions from games are often mentioned, but the methods used are syntactic and based on rewriting.

Other related works include Laurent's model for the first-order $\lambda\mu$ -calculus [19], whose annotation of moves via first-order terms is similar to ours; and Mimram's categorical presentation of a games model for a linear first-order logic without propositional connectives [25].

Since our model avoids polarization, some phenomena from the proof theory of classical logic reflect in it: our semantics does not preserve cut reduction – if it did, it would be a boolean algebra [13]. Yet it preserves it in a sense for *first-order MLL* [12]. Likewise, just as classical proofs can lead to arbitrary large cut-free proofs [8], our semantics may yield *infinite* strategies, from which *finite* sub-strategies can nonetheless always be extracted. This reflects that non-polarized proof systems for classical logic are often only weakly normalizing.

In Section 2 we recall Herbrand's theorem, and introduce the game-theoretic language leading to our compositional reformulation of it. The rest of the paper describes the interpretation of proofs as winning strategies: in Section 3 we give the interpretation of propositional MLL, in Section 4 we deal with quantifiers, and finally, in Section 5, we add contraction and weakening and complete the interpretation.

2 From Herbrand to winning Σ -strategies

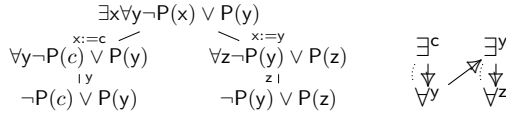
A **signature** is $\Sigma = (\Sigma_f, \Sigma_p)$, with Σ_f a countable set of **function symbols** (f, g, h , etc. range over function symbols), and Σ_p a countable set of **predicate symbols** (P, Q , etc. range over predicate symbols). There is an **arity function** $\text{ar} : \Sigma_f \uplus \Sigma_p \rightarrow \mathbb{N}$ where \uplus is the usual set-theoretic union, where argument sets are disjoint. For a relative gain in simplicity in some arguments and examples, we assume that Σ has at least one constant symbol, i.e., a function symbol of arity 0. We use a, b, c, \dots to range over constant symbols.

If \mathcal{V} is a set of **variable names**, we write $\text{Term}_\Sigma(\mathcal{V})$ for the set of first-order terms on Σ with free variables in \mathcal{V} . We use variables t, s, u, v, \dots to range over terms. **Literals** have the form $P(t_1, \dots, t_n)$ or $\neg P(t_1, \dots, t_n)$, where P is a n -ary predicate symbol and the t_i s are terms. **Formulas** are also closed under quantifiers, and the connectives \vee and \wedge . **Negation** is not considered a logical connective: the negation φ^\perp of φ is obtained by De Morgan rules. We write $\text{Form}_\Sigma(\mathcal{V})$ for the set of **first-order formulas** on Σ with free variables in \mathcal{V} , and use φ, ψ, \dots to range over them. We also write $\text{QF}_\Sigma(\mathcal{V})$ for the set of **quantifier-free** formulas. Finally, we write $\text{fv}(\varphi)$ or $\text{fv}(t)$ for the set of free variables in a formula φ or a term t . Formulas are considered up to α -conversion and satisfy Barendregt's convention.

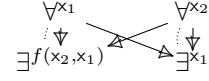
2.1 Herbrand's theorem

Intuitionistic logic has the *witness* property: if $\exists x \varphi$ holds intuitionistically, then there is some term t such that $\varphi(t)$ holds. While this fails in classical logic, Herbrand's theorem, in its popular form, gives a weakened classical version, a *finite disjunction property*.

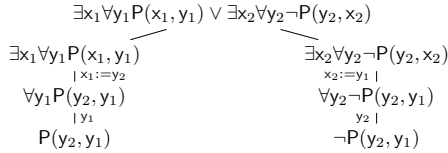
► **Theorem 1.** *Let \mathcal{T} be a theory finitely axiomatized by universal formulas. Let $\psi = \exists x_1 \dots \exists x_n \varphi(x_1, \dots, x_n)$ be a purely existential formula ($\varphi \in \text{QF}_\Sigma$). Then, $\mathcal{T} \models \psi$ iff there are closed terms $(t_{i,j})_{1 \leq i \leq p, 1 \leq j \leq n}$ such that $\mathcal{T} \models \bigvee_{i=1}^p \varphi(t_{i,1}, \dots, t_{i,n})$.*



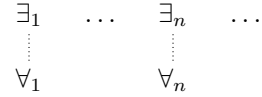
■ **Figure 1** An expansion tree and winning Σ -strategy for DF.



■ **Figure 2** A partially ordered winning Σ -strategy.



■ **Figure 3** An incorrect expansion tree.



■ **Figure 4** The arena $[[DF]]^3$.

► **Example 2.** Consider the formula $\psi = \exists x \neg P(x) \vee P(f(x))$ (where $f \in \Sigma_f$). A valid Herbrand disjunction for ψ is $(\neg P(c) \vee P(f(c))) \vee (\neg P(f(c)) \vee P(f(f(c))))$ where c is some constant symbol.

A similar disjunction property holds for general formulas, though it is harder to state. A common way to do so is by reduction to the above: a formula φ is converted to prenex normal form and universally quantified variables are replaced with new function symbols added to Σ , in a process called *Herbrandization* (dual to Skolemization). For instance, the *drinker’s formula* (DF): $\exists x \forall y \neg P(x) \vee P(y)$, yields by Herbrandization the formula ψ of Example 2.

Instead, to avoid prenexification and Skolemization and the corresponding distortion of the formula, one may adopt a representation of proofs that displays the instantiation of existential quantifiers with finitely many witnesses while staying structurally faithful to the original formula. To that end Miller proposes **expansion trees** [24]. They can be introduced via a game-theoretic metaphor, reminiscent of [7]. Two players, \exists loïse and \forall bélar, debate the validity of a formula. On a formula $\forall x \varphi$, \forall bélar provides a fresh variable x and the game keeps going on φ . On $\exists x \varphi$, \exists loïse provides a *term* t , possibly containing variables previously introduced by \forall bélar. \exists loïse, though, has a special power: at any time she can *backtrack* to a previous existential position, and propose a new term. Figure 1 (left) shows an expansion tree for DF. It may be read from top to bottom, and from left to right: \exists loïse plays c , then \forall bélar introduces y , then \exists loïse *backtracks* (we jump to the right branch) and plays y , and finally \forall bélar introduces z . \exists loïse wins: the disjunction of the leaves is a tautology.

However the metaphor has limits, it suggests a sequential ordering between branches, which expansion trees do not have in reality: the order is only implicit in the term annotations. Besides, the natural ordering between quantifiers induced by terms is not always sequential. It is, of course, always acyclic – on expansion trees this is ensured by an *acyclicity correctness criterion*, whose necessity is made obvious by the (incorrect) expansion tree of Figure 3 “proving” a falsehood. This acyclicity entails the existence of a sequentialization, but committing to one is an arbitrary choice not forced by the proof.

A partial order is much more faithful to the proof. In this paper, we show that expansion trees can be made compositional modulo a change of perspective: rather than derived we consider this order primitive, and only later decorate it with term annotations. For instance,

we display in Figure 2 the formal object, called a (sequential) *winning Σ -strategy*, matching in our framework the expansion tree for DF. Another winning Σ -strategy, displayed in Figure 2, illustrates that this order is not always naturally sequential. By lack of space we do not define expansion trees here, though they are captured in essence by our strategies.

2.2 Expansion trees as winning Σ -strategies

We now introduce our formulation of expansion trees as Σ -strategies. Although our definitions look superficially very different from Miller's, the only fundamental difference is the explicit display of the dependency between quantifiers. Σ -strategies will be certain partial orders, with elements either “ \forall events” or “ \exists events”. Events will carry terms, in a way that respects causal dependency. Σ -strategies will play on *games* representing the formulas. The first component of a game is its *arena*, that specifies the causal ordering between quantifiers.

► **Definition 3.** An **arena** is $A = (|A|, \leq_A, \text{pol}_A)$ where $|A|$ is a set of **events**, \leq_A is a partial order that is *forest-shaped*:

- (1) if $a_1 \leq_A a$ and $a_2 \leq_A a$, then either $a_1 \leq_A a_2$ or $a_2 \leq_A a_1$, and
- (2) for all $a \in |A|$, the branch $[a]_A = \{a' \in A \mid a' \leq_A a\}$ is finite.

Finally, $\text{pol}_A : |A| \rightarrow \{\forall, \exists\}$ is a **polarity function** which expresses if a move belongs to Eloïse or \forall bélar.

A **configuration** of an arena (or any partial order) is a down-closed set of events. We write $\mathcal{C}^\infty(A)$ for the set of configurations of A , and $\mathcal{C}(A)$ for the set of *finite* configurations.

The arena only describes the moves available to both players; it says nothing about terms or winning. Similarly to expansion trees where only \exists loïse can replicate her moves, our arenas will at first be biased towards \exists loïse: each \exists move exists in as many copies as she might desire, whereas \forall events are *a priori* not copied. Figure 4 shows the \exists -biased arena $\llbracket DF \rrbracket^\exists$ for DF. The order is drawn from top to bottom. Although only \exists loïse can replicate her moves, the universal quantifier is also copied as it depends on the existential quantifier.

Strategies on an arena A will be certain *augmentations* of prefixes of A . They carry causal dependency between quantifiers induced by term annotations, but not the terms themselves.

For any partial order A and $a_1, a_2 \in |A|$, we write $a_1 \rightarrow_A a_2$ (or $a_1 \rightarrow a_2$ if A is clear from the context) if $a_1 <_A a_2$ with no other event in between – this notation was used implicitly in Figures 1 and 2. We call \rightarrow **immediate causal dependency**.

► **Definition 4.** A **strategy** σ on arena A , written $\sigma : A$, is a partial order $(|\sigma|, \leq_\sigma)$ with $|\sigma| \subseteq |A|$, such that for all $a \in |\sigma|$, $[a]_\sigma$ is finite (an *elementary event structure*); subject to:

- (1) *Arena-respecting.* We have $\mathcal{C}^\infty(\sigma) \subseteq \mathcal{C}^\infty(A)$,
- (2) *Receptivity.* If $x \in \mathcal{C}(\sigma)$ s.t. $x \cup \{a^\forall\} \in \mathcal{C}(A)$, then $a \in |\sigma|$,
- (3) *Courtesy.* If $a_1 \rightarrow_\sigma a_2$ and $(\text{pol}(a_1) = \exists \text{ or } \text{pol}(a_2) = \forall)$, then $a_1 \rightarrow_A a_2$.

These strategies are essentially the *receptive ingenuous strategies* of Melliès and Mimram [23], though their formulation, with a direct handle on causality, is closer to Rideau and Winskel's later *concurrent strategies* [27]. Receptivity means that \exists loïse cannot refuse to acknowledge a move by \forall bélar, and courtesy that the only new causal constraints that she can enforce with respect to the game is that some existential quantifiers depend on some universal quantifiers. Ignoring terms, Figure 2 (right) displays a strategy on the arena of Figure 4 – in Figure 2 we also show via dotted lines the immediate dependency of the arena.

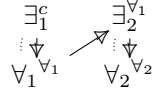
Let us now add terms, and define Σ -strategies.

► **Definition 5.** A Σ -strategy on arena A is a strategy $\sigma : A$, with a **labelling function** $\lambda_\sigma : |\sigma| \rightarrow \text{Tm}_\Sigma(|\sigma|)$, satisfying (with $[a]_\sigma^\forall = \{a' \in |\sigma| \mid a' \leq_\sigma a \ \& \ \text{pol}_A(a') = \forall\}$):

- (1) Σ -receptivity: $\forall a^\forall \in |\sigma|, \lambda_\sigma(a) = a$,
- (2) Σ -courtesy: $\forall a^\exists \in |\sigma|, \lambda_\sigma(a) \in \text{Tm}_\Sigma([a]_\sigma^\forall)$.

Rather than having \forall moves introduce fresh variables, we consider them as *variables* themselves. Hence, the \exists moves carry terms having as free variables the \forall moves in their causal history. For instance the diagram of Figure 1 (right) is meant formally to denote the one on the right (where superscripts are the terms given by λ). In the sequel we omit the (redundant) annotation of \forall bélard's events.

Besides the fact that they are not assumed finite, Σ -strategies are more general than expansion trees: they have an explicit causal ordering, which may be more constraining than that given by the terms. A Σ -strategy $\sigma : A$ is **minimal** iff whenever $a_1 \rightarrow_\sigma a_2$ such that $a_1 \notin \text{fv}(\lambda_\sigma(a_2))$, then $a_1 \rightarrow_A a_2$ as well. In a minimal Σ -strategy $\sigma : A$, the ordering \leq_σ is actually redundant and can be uniquely recovered from λ_σ and \leq_A .



Now, we adjoin *winning conditions* to arenas and define *winning Σ -strategies*. As in expansion trees, we aim to capture that the substitution (by terms from the strategies) of the expansion of the original formula is a tautology.

► **Definition 6.** A **game** \mathcal{A} is an arena A , with $\mathcal{W}_\mathcal{A} : (x \in \mathcal{C}^\infty(A)) \rightarrow \text{QF}_\Sigma^\infty(x)$ expressing **winning conditions**, where $\text{QF}_\Sigma^\infty(x)$ denotes the **infinitary quantifier-free formulas** – obtained from $\text{QF}_\Sigma(x)$ by adding infinitary connectives $\bigvee_{i \in I} \varphi_i$ and $\bigwedge_{i \in I} \varphi_i$, with I countable.

For a game interpreting a formula φ , the winning conditions associate configurations of the arena $\llbracket \varphi \rrbracket$ with the propositional part of the corresponding *expansion* of φ . For instance:

$$\begin{aligned} \mathcal{W}_{\llbracket DF \rrbracket^\exists}(\{\exists_3, \forall_3, \exists_6, \forall_6\}) &= (\neg P(\exists_3) \vee P(\forall_3)) \vee (\neg P(\exists_6) \vee P(\forall_6)) \\ \mathcal{W}_{\llbracket DF \rrbracket^\exists}(\{\exists_3, \forall_3, \exists_6\}) &= (\neg P(\exists_3) \vee P(\forall_3)) \vee \top \end{aligned}$$

recalling that the arena for DF appears in Figure 4. In the second clause, \top (the true formula) comes from \forall bélard not having played \forall_6 yet, yielding victory to \exists loïse on that copy. The winning conditions yield syntactic, uninterpreted formulas: we keep the second formula as-is although it is equivalent to \top . Finally, we can define *winning strategies*.

► **Definition 7.** If $\sigma : A$ is a Σ -strategy and $x \in \mathcal{C}^\infty(\sigma)$, we say that x is **tautological** in σ if the formula $\mathcal{W}_\mathcal{A}(x)[\lambda_\sigma]$ corresponding to the substitution of $\mathcal{W}_\mathcal{A}(x) \in \text{QF}_\Sigma^\infty(x)$ by $\lambda_\sigma : x \rightarrow \text{Tm}_\Sigma(x)$, is a (possibly infinite) tautology.

Then, a Σ -strategy $\sigma : A$ is **winning** if for any $x \in \mathcal{C}^\infty(\sigma)$ that is \exists -**maximal** (i.e., such that for all $a \in |\sigma|$ with $x \cup \{a\} \in \mathcal{C}^\infty(\sigma)$, $\text{pol}_A(a) = \forall$), x is tautological.

Finally, a Σ -strategy $\sigma : A$ is **top-winning** if $|\sigma| \in \mathcal{C}^\infty(\sigma)$ is tautological.

2.3 Constructions on games and Herbrand's theorem

To complete our statement of Herbrand's theorem with Σ -strategies, it remains to set the interpretation of formulas as games. To that end we introduce a few constructions on games, first at the level of arenas and then enriched with winning conditions. We write \emptyset for the **empty arena**. If A is an arena, A^\perp is its **dual**, with same events and causality but polarity reversed. We review some other constructions.

► **Definition 8.** The **simple parallel composition** $A_1 \parallel A_2$ of A_1 and A_2 has as events the tagged disjoint union $\{1\} \times |A_1| \uplus \{2\} \times |A_2|$, as causal order that given by $(i, a) \leq_{A_1 \parallel A_2} (j, a')$ iff $i = j$ and $a \leq_{A_i} a'$, and, as polarity $\text{pol}_{A_1 \parallel A_2}((i, a)) = \text{pol}_{A_i}(a)$.

$$\begin{array}{llll} \llbracket \top \rrbracket_{\mathcal{V}}^{\exists} = 1 & \llbracket \mathsf{P}(t_1, \dots, t_n) \rrbracket_{\mathcal{V}}^{\exists} = \mathsf{P}(t_1, \dots, t_n) & \llbracket \exists x \varphi \rrbracket_{\mathcal{V}}^{\exists} = ?\exists x. \llbracket \varphi \rrbracket_{\mathcal{V} \uplus \{x\}} & \llbracket \varphi_1 \vee \varphi_2 \rrbracket_{\mathcal{V}}^{\exists} = \llbracket \varphi_1 \rrbracket_{\mathcal{V}}^{\exists} \wp \llbracket \varphi_2 \rrbracket_{\mathcal{V}}^{\exists} \\ \llbracket \perp \rrbracket_{\mathcal{V}}^{\exists} = \perp & \llbracket \neg \mathsf{P}(t_1, \dots, t_n) \rrbracket_{\mathcal{V}}^{\exists} = \neg \mathsf{P}(t_1, \dots, t_n) & \llbracket \forall x \varphi \rrbracket_{\mathcal{V}}^{\exists} = \forall x. \llbracket \varphi \rrbracket_{\mathcal{V} \uplus \{x\}} & \llbracket \varphi_1 \wedge \varphi_2 \rrbracket_{\mathcal{V}}^{\exists} = \llbracket \varphi_1 \rrbracket_{\mathcal{V}}^{\exists} \otimes \llbracket \varphi_2 \rrbracket_{\mathcal{V}}^{\exists} \end{array}$$

■ **Figure 5** \exists -biased interpretation of formulas.

Configurations $x \in \mathcal{C}^{\infty}(A \parallel B)$ have the form $\{1\} \times x_A \cup \{2\} \times x_B$ with $x_A \in \mathcal{C}^{\infty}(A)$ and $x_B \in \mathcal{C}^{\infty}(B)$, which we write $x = x_A \parallel x_B$. This construction has a general counterpart $\parallel_{i \in I} A_i$ with I at most countable, defined likewise. In particular we will later use the uniform countably infinite parallel composition $\parallel_{\omega} A$. Another important construction is *prefixing*.

► **Definition 9.** For $\alpha \in \{\forall, \exists\}$ and A an arena, $\alpha.A$ has events $\{(1, \alpha)\} \cup \{2\} \times |A|$ and causality $(i, a) \leq (j, a')$ iff $i = j = 2$ and $a \leq_A a'$, or $(i, a) = (1, \alpha)$; i.e., $(1, \alpha)$ is the unique minimal event. Its polarity is $\text{pol}_{\alpha.A}((1, \alpha)) = \alpha$ and $\text{pol}_{\alpha.A}((2, a)) = \text{pol}_A(a)$.

Configurations $x \in \mathcal{C}^{\infty}(\alpha.A)$ are \emptyset , or $\{(1, \alpha)\} \cup \{2\} \times x_A$ ($x_A \in \mathcal{C}^{\infty}(A)$), written $\alpha.x_A$.

Now, let us enrich these with winning, yielding the constructions on games used for interpreting formulas. Importantly, the inductive interpretation of formulas requires us to consider formulas with free variables. For \mathcal{V} a finite set, a **\mathcal{V} -game** is defined as a game \mathcal{A} (Def. 6), except that winning may also depend on \mathcal{V} : for $x \in \mathcal{C}^{\infty}(A)$, $\mathcal{W}_{\mathcal{A}}(x) \in \mathbf{QF}_{\Sigma \uplus \mathcal{V}}^{\infty}(x)$.

We now define all our constructions, on \mathcal{V} -games rather than games. The duality $(-)^{\perp}$ extends to \mathcal{V} -games, simply by negating the winning conditions: for all $x \in \mathcal{C}^{\infty}(A)$, $\mathcal{W}_{\mathcal{A}^{\perp}}(x) = \mathcal{W}_{\mathcal{A}}(x)^{\perp}$. The \parallel of arenas gives rise to *two* constructions, \otimes and \wp , on \mathcal{V} -games:

► **Definition 10.** For \mathcal{A} and \mathcal{B} \mathcal{V} -games, we define two \mathcal{V} -games with arena $A \parallel B$ and winning conditions $\mathcal{W}_{\mathcal{A} \otimes \mathcal{B}}(x_A \parallel x_B) = \mathcal{W}_{\mathcal{A}}(x_A) \wedge \mathcal{W}_{\mathcal{B}}(x_B)$ and $\mathcal{W}_{\mathcal{A} \wp \mathcal{B}}(x_A \parallel x_B) = \mathcal{W}_{\mathcal{A}}(x_A) \vee \mathcal{W}_{\mathcal{B}}(x_B)$.

Note the implicit renaming so that $\mathcal{W}_{\mathcal{A}}(x_A), \mathcal{W}_{\mathcal{B}}(x_B)$ are in $\mathbf{QF}_{\Sigma \uplus \mathcal{V}}^{\infty}(x_A \parallel x_B)$ rather than $\mathbf{QF}_{\Sigma \uplus \mathcal{V}}^{\infty}(x_A), \mathbf{QF}_{\Sigma \uplus \mathcal{V}}^{\infty}(x_B)$ respectively – we will often keep such renamings implicit.

Observe that \otimes and \wp are De Morgan duals, i.e., $(\mathcal{A} \otimes \mathcal{B})^{\perp} = \mathcal{A}^{\perp} \wp \mathcal{B}^{\perp}$. We write these operations \otimes and \wp rather than \wedge and \vee , because they behave more like the connectives of linear logic [12] than those of classical logic; for each \mathcal{V} the \otimes and \wp will form the basis of a $*$ -autonomous structure and hence a model of multiplicative linear logic (see Section 3).

To interpret classical logic however, we will need *replication*.

► **Definition 11.** For \mathcal{V} -game \mathcal{A} , we define the \mathcal{V} -games $! \mathcal{A}, ? \mathcal{A}$ with arena $\parallel_{\omega} A$ and winning:

$$\mathcal{W}_{! \mathcal{A}}(\parallel_{i \in \omega} x_i) = \bigwedge_{i \in \omega} \mathcal{W}_{\mathcal{A}}(x_i) \quad \mathcal{W}_{? \mathcal{A}}(\parallel_{i \in \omega} x_i) = \bigvee_{i \in \omega} \mathcal{W}_{\mathcal{A}}(x_i)$$

Though $\mathcal{W}_{! \mathcal{A}}(x)$ (resp. $\mathcal{W}_{? \mathcal{A}}(x)$) is an infinite conjunction (resp. disjunction), it simplifies to a finite one when x visits finitely many copies (with cofinitely many copies of $\mathcal{W}_{\mathcal{A}}(\emptyset)$).

Next we show how \mathcal{V} -games support quantifiers.

► **Definition 12.** Let \mathcal{A} a $(\mathcal{V} \uplus \{x\})$ -game, we define the \mathcal{V} -game $\forall x. \mathcal{A}$ and its dual $\exists x. \mathcal{A}$ with arenas $\forall. A$ and $\exists. A$ respectively, with $\mathcal{W}_{\forall x. \mathcal{A}}(\emptyset) = \top$, $\mathcal{W}_{\exists x. \mathcal{A}}(\emptyset) = \perp$, and:

$$\mathcal{W}_{\forall x. \mathcal{A}}(\forall. x_A) = \mathcal{W}_{\mathcal{A}}(x_A)[\forall/x] \quad \mathcal{W}_{\exists x. \mathcal{A}}(\exists. x_A) = \mathcal{W}_{\mathcal{A}}(x_A)[\exists/x]$$

Finally, we regard a literal φ as a \mathcal{V} -game on arena \emptyset , with $\mathcal{W}_{\varphi}(\emptyset) = \varphi$. We write 1 and \perp for the unit \mathcal{V} -games on arena \emptyset with winning conditions respectively \top and \perp .

Putting these together, we give in Figure 5 the \exists -biased interpretation of a formula $\varphi \in \text{Form}_{\Sigma}(\mathcal{V})$ as a \mathcal{V} -game. Note the difference between the case of existential and universal

\mathcal{V}-MLL			
$\text{Ax} \frac{}{\vdash^{\mathcal{V}} \varphi^{\perp}, \varphi} \text{fv}(\varphi) \subseteq \mathcal{V}$	$\text{Cut} \frac{\vdash^{\mathcal{V}} \Gamma, \varphi \quad \vdash^{\mathcal{V}} \varphi^{\perp}, \Delta}{\vdash^{\mathcal{V}} \Gamma, \Delta}$	$\text{Ex} \frac{\vdash^{\mathcal{V}} \Gamma, \varphi, \psi, \Delta}{\vdash^{\mathcal{V}} \Gamma, \psi, \varphi, \Delta}$	
$\text{TI} \frac{}{\vdash^{\mathcal{V}} \top}$	$\perp\text{I} \frac{\vdash^{\mathcal{V}} \Gamma}{\vdash^{\mathcal{V}} \Gamma, \perp}$	$\wedge\text{I} \frac{\vdash^{\mathcal{V}} \Gamma, \varphi \quad \vdash^{\mathcal{V}} \psi, \Delta}{\vdash^{\mathcal{V}} \Gamma, \varphi \wedge \psi, \Delta}$	$\vee\text{I} \frac{\vdash^{\mathcal{V}} \Gamma, \varphi, \psi, \Delta}{\vdash^{\mathcal{V}} \Gamma, \varphi \vee \psi, \Delta}$

First-order MLL (MLL₁)	LK
$\forall\text{I} \frac{\vdash^{\mathcal{V}\{x\}} \Gamma, \varphi}{\vdash^{\mathcal{V}} \Gamma, \forall x. \varphi} \quad x \notin \text{fv}(\Gamma) \quad \exists\text{I} \frac{\vdash^{\mathcal{V}} \Gamma, \varphi[t/x]}{\vdash^{\mathcal{V}} \Gamma, \exists x. \varphi} \quad t \in \text{Term}_{\Sigma}(\mathcal{V})$	$\text{C} \frac{\vdash^{\mathcal{V}} \Gamma, \varphi, \varphi}{\vdash^{\mathcal{V}} \Gamma, \varphi} \quad \text{W} \frac{\vdash^{\mathcal{V}} \Gamma}{\vdash^{\mathcal{V}} \Gamma, \varphi}$

■ **Figure 6** Rules for the sequent calculus LK.

formulas, reflecting the bias towards \exists loïse. This is indeed compatible with the examples given previously. We can now state our concurrent version of Herbrand's theorem.

► **Theorem 13.** *For any $\varphi \in \text{Form}_{\Sigma}$, $\models \varphi$ iff there exists a finite, top-winning $\sigma : \llbracket \varphi \rrbracket^{\exists}$.*

Besides the game-theoretic language, the difference with expansion trees is superficial: on φ , expansion trees essentially coincide with the *minimal* top-winning Σ -strategies $\sigma : \llbracket \varphi \rrbracket^{\exists}$. The effort to change view point, from a syntactic construction to a (game) semantic one, will however pay off now, when we show how to *compose* Σ -strategies.

2.4 Compositional Herbrand's theorem

Unlike expansion trees, strategies can be *composed*. Whereas Theorem 13 above could be deduced via the connection with expansion trees, that proof would intrinsically rely on the admissibility of cut in the sequent calculus. Instead, we will give an alternative proof of Herbrand's theorem where the witnesses are obtained truly *compositionally* from any sequent proof, without first eliminating cuts. In other words, strategies will come naturally from the interpretation of the classical sequent calculus in a semantic model.

To compose Σ -strategies, we must restore the symmetry between \exists loïse and \forall bélar in the interpretation of formulas. The *non-biased* interpretation $\llbracket \varphi \rrbracket_{\mathcal{V}}$ of $\varphi \in \text{Form}_{\Sigma}(\mathcal{V})$ is defined as for $\llbracket \varphi \rrbracket_{\mathcal{V}}^{\exists}$, except for $\llbracket \forall x \varphi \rrbracket_{\mathcal{V}} = !\forall x. \llbracket \varphi \rrbracket_{\mathcal{V}\{x\}}$. Thus we lose finiteness: \exists loïse must be reactive to the infinite number of copies potentially opened by \forall bélar. But we can now state:

► **Theorem 14.** *For φ closed, the following are equivalent: (1) $\models \varphi$, (2) there exists a finite, top-winning Σ -strategy $\sigma : \llbracket \varphi \rrbracket^{\exists}$, (3) there exists a winning Σ -strategy $\sigma : \llbracket \varphi \rrbracket$.*

Proof. That (2) implies (1) is easy, as a finite top-winning $\sigma : \llbracket \varphi \rrbracket^{\exists}$ directly informs a proof.

That (3) implies (2) is more subtle: first, one may restrict a winning $\sigma : \llbracket \varphi \rrbracket$ to $\llbracket \varphi \rrbracket^{\exists}$ to obtain a finite top-winning strategy. However, this top-winning strategy *may not be finite*. Yet, it follows by compactness that there is always a finite top-winning sub-strategy that may be effectively computed from σ . See the Appendix C for details.

The proof that (1) implies (3) is our main contribution: a winning strategy will be computed from a proof using our denotational model of classical proofs. ◀

Our source sequent calculus (Figure 6) is fairly standard, one-sided, with rules presented in the multiplicative style. A notable variation is that sequents carry a set \mathcal{V} of free variables, that may appear freely in formulas. The introduction rule for \forall introduces a fresh variable, whereas the introduction rule for \exists provides a term whose free variables must be in \mathcal{V} .

What mathematical structure is required to interpret this sequent calculus? Ignoring the \mathcal{V} annotations, the first group is nothing but Multiplicative Linear Logic (MLL). Propositional (\mathcal{V} -)MLL can be interpreted in a $*$ -autonomous category [3]. Accordingly, in Section 3, we first construct a $*$ -autonomous category Ga of games and winning Σ -strategies. Then, in Section 4, we build the structure required for the interpretation of quantifiers, still ignoring contraction and weakening. For each set of variables \mathcal{V} we construct a $*$ -autonomous category $\mathcal{V}\text{-Ga}$, with a fibred structure to link the $\mathcal{V}\text{-Ga}$ together for distinct \mathcal{V} s and suitable structure to deal with quantifiers, obtaining a model of first-order MLL. Finally in Section 5 we complete the interpretation by adding the exponential modalities from linear logic to the interpretation of quantifiers, and get from that an interpretation of contraction and weakening.

3 A $*$ -autonomous category

The following theorem, on cut reduction for MLL, is folklore.

► **Theorem 15.** *There is a set of reduction rules on MLL sequent proofs, written $\rightsquigarrow_{\text{MLL}}$, such that for any proof π of a sequent $\vdash \Gamma$, there is a cut-free π' of Γ such that $\pi \rightsquigarrow_{\text{MLL}}^* \pi'$.*

The reduction $\rightsquigarrow_{\text{MLL}}$ comprises *logical* reductions, reducing a cut on a formula φ/φ^\perp , between two proofs starting with the introduction rule for the main connective of φ/φ^\perp ; and *structural* reductions, consisting in commutations between rules so as to reach the logical steps. We assume some familiarity with this process.

In this section we aim to give an interpretation of MLL proofs, which should be invariant under cut-elimination. Categorical logic tells us that this is essentially the same as producing a *$*$ -autonomous category*. We opt here for the equivalent formulation by Cockett and Seely as a *symmetric linearly distributive category with negation* [6].

► **Definition 16.** A **symmetric linearly distributive category** is a category \mathcal{C} with two symmetric monoidal structures $(\otimes, 1)$ and (\wp, \perp) which *distribute*: there is a natural $\delta_{A,B,C} : A \otimes (B \wp C) \xrightarrow{\mathcal{C}} (A \otimes B) \wp C$, the *linear distribution*, subject to coherence conditions [6].

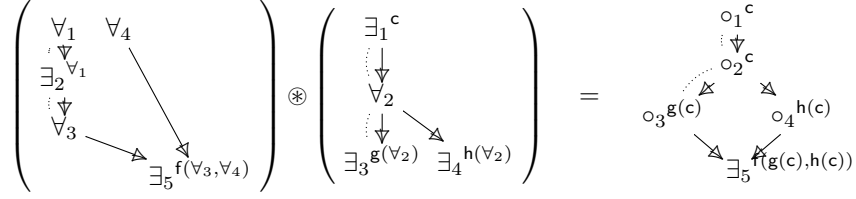
A symmetric linearly distributive category **with negation** also has a function $(-)^\perp$ on objects and families of maps $\eta_A : 1 \xrightarrow{\mathcal{C}} A^\perp \wp A$ and $\epsilon_A : A \otimes A^\perp \xrightarrow{\mathcal{C}} \perp$ such that the canonical composition $A \rightarrow A \otimes (A^\perp \wp A) \rightarrow (A \otimes A^\perp) \wp A \rightarrow A$, and its dual $A^\perp \rightarrow A^\perp$, are identities.

Note also the degenerate case of a **compact closed category**, which is a symmetric linearly distributive category where the monoidal structures $(\otimes, 1)$ and (\wp, \perp) coincide.

Abusing terminology, we will refer to *symmetric linearly distributive categories with negation* by the shorter *$*$ -autonomous categories*. This should not create any confusion in the light of their equivalence [6]. If \mathcal{C} a $*$ -autonomous category comes with a choice of $\llbracket P(t_1, \dots, t_n) \rrbracket$ (an object of \mathcal{C}) for all closed literal, then this interpretation can be extended to all closed quantifier-free formulas following Figure 5. For all such φ , we have $\llbracket \varphi^\perp \rrbracket = \llbracket \varphi \rrbracket^\perp$.

The interpretation of MLL proofs in a $*$ -autonomous category \mathcal{C} is standard [29]: a proof π of a *MLL sequent* $\vdash \varphi_1, \dots, \varphi_n$ is interpreted as a morphism $\llbracket \pi \rrbracket : 1 \xrightarrow{\mathcal{C}} \llbracket \varphi_1 \rrbracket \wp \dots \wp \llbracket \varphi_n \rrbracket$. This interpretation is sound *w.r.t.* provability: if φ is provable, then $1 \rightarrow_{\mathcal{C}} \llbracket \varphi \rrbracket$ is inhabited. Furthermore, the categorical laws make this interpretation invariant under cut reduction.

► **Theorem 17.** *If $\pi \rightsquigarrow_{\text{MLL}} \pi'$ are proofs of $\vdash \Gamma$, $\llbracket \pi \rrbracket = \llbracket \pi' \rrbracket$.*



■ **Figure 7** Interaction of $\sigma : 1^\perp \parallel (\exists_1 \forall_2 \exists_3 \parallel \exists_4)$ and $\tau : (\exists_1 \forall_2 \exists_3 \parallel \exists_4)^\perp \parallel \exists_5$.

So a proof has the same denotation as its cut-free form obtained by Theorem 15. In the rest of this section we construct a concrete $*$ -autonomous category of games and winning Σ -strategies; supporting the interpretation of MLL. This is done in three stages: first we focus on composition of Σ -strategies (without winning), then we extend this to a compact closed category. Finally, adding back winning, we split \parallel into two \otimes and \wp , and prove $*$ -autonomy.

3.1 Composition of Σ -strategies

We construct a category Ar_Σ having arenas as objects, and as morphisms from A to B the Σ -strategies $\sigma : A^\perp \parallel B$, also written $\sigma : A \xrightarrow{\text{Ar}_\Sigma} B$. The composition of $\sigma : A \xrightarrow{\text{Ar}_\Sigma} B$ and $\tau : B \xrightarrow{\text{Ar}_\Sigma} C$ will be computed in two stages: first, the *interaction* $\tau \otimes \sigma$ is obtained as the most general partial-order-with-terms satisfying the constraints given by both σ and τ – Figure 7 displays such an interaction. Then, we will obtain the *composition* $\tau \odot \sigma$ by hiding events in B . In the example of Figure 7 we get the single annotated event $\exists_5^{f(g(c), h(c))}$.

We fix some definitions on terms and substitutions. If $\mathcal{V}_1, \mathcal{V}_2$ are sets, a **substitution** $\gamma : \mathcal{V}_1 \xrightarrow{\Sigma} \mathcal{V}_2$ is a function $\gamma : \mathcal{V}_2 \rightarrow \text{Tm}_\Sigma(\mathcal{V}_1)$. For $t \in \text{Tm}_\Sigma(\mathcal{V}_2)$, we write $t[\gamma] \in \text{Tm}_\Sigma(\mathcal{V}_1)$ for the substitution operation. Substitutions form a category \mathcal{S} , which is *cartesian*: the empty set \emptyset is terminal, and the product of \mathcal{V}_1 and \mathcal{V}_2 is their disjoint union $\mathcal{V}_1 + \mathcal{V}_2$. From $\gamma : \mathcal{V}_1 \xrightarrow{\Sigma} \mathcal{V}_2$ and $\gamma' : \mathcal{V}'_1 \xrightarrow{\Sigma} \mathcal{V}_2$, we say that γ **subsumes** γ' , written $\gamma' \preceq \gamma$, if there is $\alpha : \mathcal{V}'_1 \xrightarrow{\Sigma} \mathcal{V}_2$ s.t. $\gamma \circ \alpha = \gamma'$ – giving a preorder on substitutions with codomain \mathcal{V}_2 .

Consider first the *closed* interaction of two Σ -strategies $\sigma : A$ and $\tau : A^\perp$. As they disagree on the polarities on A we drop them – $\tau \otimes \sigma$ will be a *neutral Σ -strategy* on a *neutral arena*:

► **Definition 18.** A **neutral arena** is an arena, without polarities. **Neutral strategies** $\sigma : A$, are defined as in Definition 4 without (2), (3). **Neutral Σ -strategies** additionally have $\lambda_\sigma : (s \in |\sigma|) \rightarrow \text{Tm}_\Sigma([s]_\sigma)$, and are **idempotent**: for all $a \in |a|$, $\lambda_\sigma(a)[\lambda_\sigma] = \lambda_\sigma(a)$.

Forgetting polarities, every Σ -strategy is a neutral one. Given σ and τ , $\tau \otimes \sigma$ is a *minimal strengthening* of σ and τ , regarding both the causal structure and term annotations, i.e., a *meet* for the partial order (*idempotence* above is required for it to be antisymmetric):

► **Definition 19.** For $\sigma, \tau : A$ neutral Σ -strategies, we write $\sigma \preceq \tau$ iff $|\sigma| \subseteq |\tau|$, $\mathcal{C}^\infty(\sigma) \subseteq \mathcal{C}^\infty(\tau)$, and for all $x \in \mathcal{C}(|\sigma|)$, $\lambda_\tau \upharpoonright x$ subsumes $\lambda_\sigma \upharpoonright x$ (regarded as substitutions $x \xrightarrow{\Sigma} x$).

Ignoring terms, any two σ and τ have a meet $\sigma \wedge \tau$; this is a simplification of the *pullback* in the category of event structures, exploiting the absence of conflict [31]. The partial order $(|\sigma \wedge \tau|, \leq_{\sigma \wedge \tau})$ has events all common moves of σ and τ with a causal history compatible with both \leq_σ and \leq_τ , and for $\leq_{\sigma \wedge \tau}$ the minimal causal order compatible with both.

However, two neutral Σ -strategies do not necessarily have a meet for \preceq (see Example 45 in Appendix A). Hence, we focus on the meets occurring from compositions of Σ -strategies and show that for $\sigma : A$ and $\tau : A^\perp$ dual Σ -strategies the meet *does* exist:

► **Lemma 20.** *Any two Σ -strategies $\sigma : A$ and $\tau : A^\perp$ have a meet $\sigma \wedge \tau$.*

Proof. We start with the causal meet $\sigma \wedge \tau$, which we enrich with $\lambda_{\sigma \wedge \tau}$ the *most general unifier* of $\lambda_\sigma \upharpoonright |\sigma \wedge \tau|$ and $\lambda_\tau \upharpoonright |\sigma \wedge \tau|$, obtained by well-founded induction on $\leq_{\sigma \wedge \tau}$:

$$\lambda_{\sigma \wedge \tau}(a) = \begin{cases} \lambda_\sigma(a)[\lambda_{\sigma \wedge \tau} \upharpoonright [a]] & \text{if } \text{pol}_A(a) = \exists \\ \lambda_\tau(a)[\lambda_{\sigma \wedge \tau} \upharpoonright [a]] & \text{if } \text{pol}_A(a) = \forall \end{cases}$$

where $[a] = \{a' \in A \mid a' <_{\sigma \wedge \tau} a\}$. It follows that this is indeed the *m.g.u.* – in particular, we exploit that from Σ -courtesy, if $a^\exists \in |\sigma|$ then $\lambda_\sigma(a) \in \text{Tm}_\Sigma([a]_\sigma)$. ◀

However this is not sufficient: for composable $\sigma : A^\perp \parallel B$ and $\tau : B^\perp \parallel C$, the games are not purely dual; we need to “pad out” σ and τ and compute instead $(\sigma \parallel C^\perp) \wedge (A \parallel \tau)$, where the parallel composition of Definition 8 is extended with terms in the obvious way, and where $\lambda_A(a) = a$ for all $a \in |A|$. Now $\sigma \parallel C^\perp : A^\perp \parallel B \parallel C^\perp$ and $A \parallel \tau : A \parallel B^\perp \parallel C$ are dual, but Σ -courtesy from Σ -strategies is relaxed to idempotence. Yet, Lemma 20 still holds since, from idempotence, if $a^\exists \in |\sigma|$ then either $\lambda_\sigma(a) \in \text{Tm}_\Sigma([a]_\sigma)$ or $\lambda_\sigma(a) = a$. Hence, we can define $\tau \otimes \sigma = (\sigma \parallel C^\perp) \wedge (A \parallel \tau) : A \parallel B \parallel C$.

Variables appearing in $\lambda_{\tau \otimes \sigma}$ cannot be events in B – they must be negative in $A^\perp \parallel C$. So we can define $\tau \odot \sigma = (\tau \otimes \sigma) \cap (A \parallel C)$ the restriction of $\tau \otimes \sigma$ to $A \parallel C$, with same causal order and term annotation. The pair $(|\tau \odot \sigma|, \leq_{\tau \odot \sigma})$ is a strategy, as an instance of the constructions in [4], and this extends to terms so that $\tau \odot \sigma : A^\perp \parallel C$ is a Σ -strategy, the **composition** of σ and τ . Because interaction is defined as a meet for \preceq , it follows that it is compatible with it, i.e., if $\sigma \preceq \sigma'$, then $\tau \otimes \sigma \preceq \tau \otimes \sigma'$. This is preserved by projection, and hence $\tau \odot \sigma \preceq \tau \odot \sigma'$ as well. This compatibility of composition with \preceq will be used later on, together with the easy fact that \preceq is more constrained on Σ -strategies:

► **Lemma 21.** *For $\sigma, \sigma' : A$ Σ -strategies, if $\sigma \preceq \sigma'$, then $\lambda_\sigma(s) = \lambda_{\sigma'}(s)$ for all $s \in |\sigma|$.*

To complete our category, we also define the *copycat strategy*.

► **Definition 22.** For an arena A , the **copycat Σ -strategy** $\alpha_A : A^\perp \parallel A$ has events $|\alpha_A| = A^\perp \parallel A$. Writing $(i, a) = (3 - i, a)$, its partial order \leq_{α_A} is the transitive closure of $\leq_{A^\perp \parallel A} \cup \{(c, \bar{c}) \mid c^\forall \in |A^\perp \parallel A|\}$ and its labelling function is $\lambda_{\alpha_A}(c^\forall) = c$, $\lambda_{\alpha_A}(c^\exists) = \bar{c}$.

The proof of categorical laws are variations on construction of the bicategory in [4].

► **Proposition 23.** *There is a poset-enriched category Ar_Σ with arenas as objects, and Σ -strategies as morphisms.*

3.2 Compact closed structure

We show that Ar_Σ is compact closed. The **tensor product** of arenas A and B is $A \parallel B$. For Σ -strategies $\sigma_1 : A_1^\perp \parallel B_1$ and $\sigma_2 : A_2^\perp \parallel B_2$, we have $\sigma_1 \parallel \sigma_2 : (A_1^\perp \parallel B_1) \parallel (A_2^\perp \parallel B_2)$, which is isomorphic to $(A_1 \parallel A_2)^\perp \parallel (B_1 \parallel B_2)$ – overloading notations, we also write $\sigma_1 \parallel \sigma_2 : (A_1 \parallel A_2)^\perp \parallel (B_1 \parallel B_2)$ for the obvious renaming. It is not difficult to prove:

► **Proposition 24.** *Simple parallel composition yields an enriched functor $\parallel : \text{Ar}_\Sigma \times \text{Ar}_\Sigma \rightarrow \text{Ar}_\Sigma$.*

For the compact closed structure, we elaborate the renaming used above. We write $f : A \cong B$ for an **isomorphism of arenas**, preserving and reflecting all structure.

► **Definition 25.** For $f : A \cong B$ and $\sigma : A$ a Σ -strategy, the **renaming** $f * \sigma : B$ has components $|f * \sigma| = f|\sigma|$, $\leq_{f*\sigma} = \{(f a_1, f a_2) \mid a_1 \leq_\sigma a_2\}$ and $\lambda_{f*\sigma}(f a) = \lambda_\sigma(a)[f]$.

In particular, if $f : A \cong B$, then the corresponding **copycat strategy** is $\alpha_f = (A^\perp \parallel f) * \alpha_A : A^\perp \parallel B$. We use this to define the structural morphisms for the symmetric monoidal structure of Ar_Σ . For instance, the iso $\alpha_{A,B,C} : (A \parallel B) \parallel C \cong A \parallel (B \parallel C)$ yields $\alpha_{\alpha_{A,B,C}} : (A \parallel B) \parallel C \xrightarrow{\text{Ar}_\Sigma} A \parallel (B \parallel C)$. The other structural morphisms arise similarly. Coherence and naturality then follows from the key *copycat lemma*:

► **Lemma 26.** For $\sigma : A^\perp \parallel B$ a Σ -strategy and $f : B \cong C$, $\alpha_f \odot \sigma = (A^\perp \parallel f) * \sigma : A^\perp \parallel C$.

As a corollary we get coherence for the structural morphisms (following from those on isomorphisms), and naturality. For all A we get $\eta_A : \emptyset \xrightarrow{\text{Ar}_\Sigma} A^\perp \parallel A$ and $\epsilon_A : A \parallel A^\perp \xrightarrow{\text{Ar}_\Sigma} \emptyset$ as the obvious renamings of copycat. Checking the law for compact closed categories is a variation of the idempotence of copycat. Overall:

► **Proposition 27.** Ar_Σ is a poset-enriched compact closed category.

3.3 A linearly distributive category with negation

Finally, we reinstate winning conditions. We first note:

► **Proposition 28.** There is a (poset-enriched) category Ga_Σ with objects the games (Definition 6) on Σ , and morphisms Σ -strategies $\sigma : \mathcal{A}^\perp \wp \mathcal{B}$, also written $\sigma : \mathcal{A} \xrightarrow{\text{Ga}_\Sigma} \mathcal{B}$.

That copycat is winning boils down to the excluded middle. That $\tau \odot \sigma : \mathcal{A}^\perp \wp \mathcal{C}$ is winning if $\sigma : \mathcal{A}^\perp \wp \mathcal{B}$ and $\tau : \mathcal{B}^\perp \wp \mathcal{C}$ are, is as in [5]: for $x \in \mathcal{C}(\tau \odot \sigma)$ \exists -maximal we find a witness $y \in \mathcal{C}(\tau \otimes \sigma)$ (i.e., $y \cap (A \parallel C) = x$) s.t. $y \cap (A \parallel B) \in \sigma$, $y \cap (B \parallel C) \in \tau$ are \exists -maximal; and apply transitivity of implication. The equations follow from Ar_Σ . Likewise:

► **Proposition 29.** The functor $\parallel : \text{Ar}_\Sigma \times \text{Ar}_\Sigma \rightarrow \text{Ar}_\Sigma$ splits into $\otimes, \wp : \text{Ga}_\Sigma \times \text{Ga}_\Sigma \rightarrow \text{Ga}_\Sigma$.

It suffices to check winning, which is straightforward. It remains to prove that all structural morphisms from Ar_Σ (copycat strategies) are winning, which boils down to the following sufficient conditions to hold: For \mathcal{A}, \mathcal{B} games, a **win-iso** $f : \mathcal{A} \rightarrow \mathcal{B}$ is an iso $f : A \cong B$ such that $(\mathcal{W}_A(x))^\perp \vee \mathcal{W}_B(f x)$ is a tautology, for all $x \in \mathcal{C}^\infty(A)$.

► **Lemma 30.** If $f : \mathcal{A} \rightarrow \mathcal{B}$ is a win-iso, then $\alpha_f : \mathcal{A}^\perp \wp \mathcal{B}$ is a winning Σ -strategy.

This easily entails that all structural morphisms (including linear distributivity) are winning. Finally $\eta_A : \mathbf{1} \xrightarrow{\text{Ga}_\Sigma} \mathcal{A}^\perp \wp \mathcal{A}$ and $\epsilon_A : \mathcal{A} \otimes \mathcal{A}^\perp \xrightarrow{\text{Ga}_\Sigma} \mathbf{1}$ are winning, which concludes:

► **Proposition 31.** Ga_Σ is a poset-enriched *-autonomous category.

4 A model of first-order MLL

We move on to MLL_1 , i.e., all rules except for contraction and weakening. Before developing the interpretation, we discuss cut elimination. There are three new cut reduction rules, displayed in Figure 8: the new *logical* reduction (\forall/\exists), and two for the propagation of cuts past introduction rules for \forall and \exists . Writing $\pi \rightsquigarrow_{\text{MLL}_1} \pi'$ for the reduction obtained with these new rules together with $\rightsquigarrow_{\text{MLL}}$:

► **Proposition 32.** Let π be any MLL_1 proof of $\vdash^\vee \Gamma$. Then, there is a cut-free proof π' of $\vdash^\vee \Gamma$ s.t. $\pi \rightsquigarrow_{\text{MLL}_1}^* \pi'$.

are games on the signature $\Sigma \uplus \mathcal{V}$, i.e., the \mathcal{V} -games of Section 2.3. *Morphisms* between \mathcal{V} -games \mathcal{A} and \mathcal{B} are winning $(\Sigma \uplus \mathcal{V})$ -strategies on $\mathcal{A}^\perp \wp \mathcal{B}$ regarded as a game on signature $\Sigma \uplus \mathcal{V}$ – also called **winning Σ -strategies on the \mathcal{V} -game $\mathcal{A}^\perp \wp \mathcal{B}$** .

Finally, for \mathcal{A} a \mathcal{V}_2 -game and $\gamma : \mathcal{V}_1 \rightarrow \mathcal{V}_2$ a substitution, the game $\mathcal{T}(\gamma)(\mathcal{A}) = \mathcal{A}[\gamma]$ is defined as having arena A , and, for $x \in \mathcal{C}^\infty(A)$, $\mathcal{W}_{\mathcal{A}[\gamma]}(x) = \mathcal{W}_{\mathcal{A}}(x)[\gamma] \in \mathbf{QF}_{\Sigma \uplus \mathcal{V}_1}^\infty(x)$. Likewise, given \mathcal{A} and \mathcal{B} two \mathcal{V} -games and $\sigma : \mathcal{A}^\perp \wp \mathcal{B}$, $\sigma[\gamma]$ has the same components as σ , but term annotations $\lambda_{\sigma[\gamma]}(s) = \lambda(s)[\gamma] \in \mathbf{Tm}_{\Sigma \uplus \mathcal{V}_1}(x)$. It is a simple verification to prove:

► **Proposition 34.** *For any $\gamma : \mathcal{V}_1 \xrightarrow{\mathcal{S}} \mathcal{V}_2$, $\mathcal{T}(\gamma) : \mathcal{T}(\mathcal{V}_2) \rightarrow \mathcal{T}(\mathcal{V}_1)$ is a strict $*$ -autonomous functor preserving the order.*

4.2 Quantifiers

Finally, we give the interpretation of $\forall I$ and $\exists I$. For now, we consider a *linear* interpretation $\llbracket - \rrbracket_{\mathcal{V}}^\ell$ of formulas defined like $\llbracket - \rrbracket_{\mathcal{V}}^\exists$ except for $\llbracket \exists x \varphi \rrbracket_{\mathcal{V}}^\ell = \exists x. \llbracket \varphi \rrbracket_{\mathcal{V}}^\ell$.

Besides preserving the $*$ -autonomous structure, substitution also propagates through quantifiers, from which we have:

► **Lemma 35.** *Let $\varphi \in \mathbf{Form}_\Sigma(\mathcal{V}_2)$ and $\gamma : \mathcal{V}_1 \rightarrow \mathcal{V}_2$ a substitution, then $\llbracket \varphi[\gamma] \rrbracket_{\mathcal{V}_1}^\ell = \llbracket \varphi \rrbracket_{\mathcal{V}_2}^\ell[\gamma]$.*

This will be used implicitly from now on. The definition of quantifiers on games of Definition 12 extends to functors $\forall_{\mathcal{V},x}, \exists_{\mathcal{V},x} : \mathcal{T}(\mathcal{V} \uplus \{x\}) \rightarrow \mathcal{T}(\mathcal{V})$. From $\sigma : \mathcal{A}^\perp \wp \mathcal{B}$, $\forall_{\mathcal{V},x}(\sigma) : (\forall x. \mathcal{A})^\perp \wp \forall x. \mathcal{B}$ plays copycat on the initial \forall , then plays as σ (similarly for $\exists_{\mathcal{V},x}(\sigma)$). Following Lawvere [20], one expects adjunctions $\exists_{\mathcal{V},x} \dashv \mathcal{T}(w_{\mathcal{V},x}) \dashv \forall_{\mathcal{V},x}$. Unfortunately, this fails – we present this failure later as the non-preservation of $\rightsquigarrow_{\text{CUT}/\forall}$.

We now interpret $\forall I$ and $\exists I$. First, we give a strategy introducing a witness t .

► **Definition 36.** The $(\Sigma \uplus \mathcal{V})$ -strategy $\exists_A^t : A^\perp \parallel \exists. A$ is $(|A^\perp \parallel \exists. A|, \leq_{\exists_A^t}, \lambda_{\exists_A^t})$ where $\leq_{\exists_A^t}$ includes \leq_{α_A} , plus dependencies $\{((2, \exists), (2, a)) \mid a \in A\} \uplus \{((2, \exists), (1, a)) \mid \exists a_0^\forall \in A. a_0 \leq_A a\}$ and term assignment that of α_A plus $\lambda_{\exists_A^t}((2, \exists)) = t$.

In other words, \exists_A^t plays \exists annotated with t , then proceeds as copycat on A . We have:

► **Proposition 37.** *Let \mathcal{A} be a \mathcal{V} -game, and $t \in \mathbf{Tm}_\Sigma(\mathcal{V})$. Then, $\exists_A^t : \mathcal{A}[t/x] \xrightarrow{\mathcal{V}\text{-Gas}} \exists x. \mathcal{A}$.*

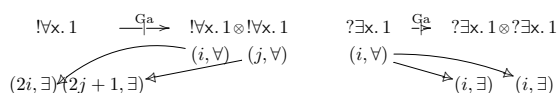
Indeed, any \exists -maximal $x_A \parallel \exists. x_A \in \mathcal{C}^\infty(\exists_A^t)$ corresponds to a tautology $\mathcal{W}_{\mathcal{A}[t/x]}(x_A)^\perp \vee \mathcal{W}_{\mathcal{A}}(x_A)[t/x]$. We interpret $\exists I$ by post-composing with \exists_A^t (as in Figure 10 without the last step). This validates $\rightsquigarrow_{\text{CUT}/\exists}$, by associativity of composition.

To a strategy σ , the operation interpreting $\forall I$ adds \forall as new minimal event, and sets it as a dependency for all events whose annotation comprise the distinguished variable x .

► **Definition 38.** For σ a $(\Sigma \uplus \mathcal{V} \uplus \{x\})$ -strategy on $A^\perp \parallel B$, the $(\Sigma \uplus \mathcal{V})$ -strategy $\forall_{A,B}^x(\sigma) : A^\perp \parallel \forall. B$ has events $|\sigma| \uplus \{(2, \forall)\}$, term assignment $\lambda((2, \forall)) = (2, \forall)$ and causality $\lambda(s) = \lambda_\sigma(s)[(2, \forall)/x]$ ($s \in |\sigma|$), and $\leq = \leq_\sigma \cup \{((2, \forall), s) \mid s \in \forall. B \quad \vee \quad \exists s' \leq_\sigma s, x \in \text{fv}(\lambda_\sigma(s'))\}$.

► **Proposition 39.** *If σ is winning on a $(\mathcal{V} \uplus \{x\})$ -game $\mathcal{A}[w_{\mathcal{V},x}] \wp \mathcal{B}$, then $\forall_{A,B}^x(\sigma)$ is winning on the \mathcal{V} -game $\mathcal{A} \wp \forall x. \mathcal{B}$.*

Indeed, if \forall bélarde does not play $(2, \forall)$ we get a tautology, otherwise the remaining configuration is in σ and so is tautological. This completes the interpretation of MLL_1 . This interpretation leaves $\rightsquigarrow_{\forall/\exists}$ invariant, but fails $\rightsquigarrow_{\text{CUT}/\forall}$. This stems from the fact that the *minimal* Σ -strategies are not stable under composition (see Example 46 in Appendix A). The interpretation of cut-free proofs yield minimal Σ -strategies. In contrast, in compositions



■ **Figure 9** Two examples of contraction.

interpreting cuts, causality may flow through the syntax tree of the cut formula, and create causal dependencies not reflected in the variables. Hence, cut reduction may weaken the causal structure.

► **Lemma 40.** *For $\sigma : A \xrightarrow{\text{Ar}\Sigma} B$ and $\tau : B \xrightarrow{\text{Ar}\Sigma_{\omega\{x\}}} C$, we have $\forall I_{A,C}^x(\tau \odot \sigma) \preceq \forall I_{B,C}^x(\tau) \odot \sigma$.*

By Lemma 21 these two have the same terms on common events. In fact, $\forall I_{A,C}^x(\tau \odot \sigma)$ and $\forall I_{B,C}^x(\tau) \odot \sigma$ also have the same *events* – they correspond to the same *expansion tree*, only the acyclicity witness differs. But the variant of \preceq with $|\sigma_1| = |\sigma_2|$ is not a congruence: relaxing causality of σ in $\tau \odot \sigma$ may unlock new events, previously part of causal loops.

As \preceq is preserved by all operations on Σ -strategies, we deduce:

► **Theorem 41.** *If $\pi \rightsquigarrow_{\text{MLL}_1} \pi'$, then $\llbracket \pi' \rrbracket \preceq \llbracket \pi \rrbracket$.*

For MLL_1 , we conjecture that “having the same expansion tree” (i.e., same events and term annotations) is actually a congruence, yielding a **-autonomous hyperdoctrine*. As this would not hold in the presence of contraction and weakening, we leave this for future work.

5 Contraction and weakening

In this section we reinstate $!$ and $?$ in the interpretation of quantifiers, i.e., $\llbracket \forall x. \varphi \rrbracket_{\mathcal{V}} = !\forall x. \llbracket \varphi \rrbracket_{\mathcal{V}_{\omega\{x\}}}$ and $\llbracket \exists x \varphi \rrbracket_{\mathcal{V}} = ?\exists x \llbracket \varphi \rrbracket_{\mathcal{V}_{\omega x}}$ – this is reminiscent of Mellies’ discussion on the interaction between quantifiers and exponential modalities in a polarized setting [22].

Unlike for MLL_1 , we only aim to map proofs to Σ -strategies on the appropriate game, with no preservation of reduction. We must interpret contraction and weakening, but also revisit the interpretation of rules for quantifiers as the interpretation of formulas has changed.

Weakening is easy: for any game \mathcal{A} , any Σ -strategy $\sigma : \mathcal{A} \rightarrow 1$ is winning; for definiteness, we use the minimal $e_{\mathcal{A}} : \mathcal{A} \rightarrow 1$, only closed under receptivity. *Contraction* is much more subtle. To illustrate the difficulty, we present in Figure 9 two simple instances of the contraction Σ -strategy (without term annotations). The first looks like the usual contraction of AJM games [1]. It can be used to interpret the contraction rule on existential formulas, where it has the effect of taking the union of the different witnesses proposed. But in LK, one can also use contraction on a universal formula, which will appeal to a strategy like the second. Any witness proposed by \forall bélard will then have to be propagated to both branches to ensure that we are winning (mimicking the effect of cut reduction).

In order to define this contraction Σ -strategy along with the tools to revisit the introduction rules for quantifiers, we will first study some properties of the exponential modalities.

Recall $!$ and $?$ from Definition 11, both based on arena $\llbracket_{\omega} A$. First, we examine their functorial action. Let $\sigma : A \xrightarrow{\text{Ar}\Sigma} B$. Then, $\llbracket_{\omega} \sigma : \llbracket_{\omega} (A^{\perp} \parallel B)$ which is isomorphic to $(\llbracket_{\omega} A)^{\perp} \parallel (\llbracket_{\omega} B)$; overloading notion we still write $\llbracket_{\omega} \sigma : \llbracket_{\omega} A \xrightarrow{\text{Ar}\Sigma} \llbracket_{\omega} B$.

► **Lemma 42.** *Let $\sigma : \mathcal{A} \xrightarrow{\text{Ga}\Sigma} \mathcal{B}$. Then, we have $!\sigma = \llbracket_{\omega} \sigma : !\mathcal{A} \xrightarrow{\text{Ga}\Sigma} !\mathcal{B}$ and $?\sigma = \llbracket_{\omega} \sigma : ?\mathcal{A} \xrightarrow{\text{Ga}\Sigma} ?\mathcal{B}$.*

Rather than defining directly the contraction, we build $co_{\varphi} : \llbracket \varphi \rrbracket_{\mathcal{V}} \xrightarrow{\text{Ga}\Sigma_{\omega\mathcal{V}}} !\llbracket \varphi \rrbracket_{\mathcal{V}}$ by induction on $\varphi \in \text{Form}_{\Sigma}(\mathcal{V})$. For φ quantifier-free, the empty $co_{\varphi} : \llbracket \varphi \rrbracket_{\mathcal{V}} \rightarrow !\llbracket \varphi \rrbracket_{\mathcal{V}}$ is winning. We

$$\begin{aligned}
 \left[\left[\frac{\pi}{\frac{\vdash^{\mathcal{V}} \Gamma, \varphi, \varphi}{\vdash^{\mathcal{V}} \Gamma, \varphi}} \right] \right]_{\mathcal{C}} &= \Gamma^{\perp} \xrightarrow{\mathcal{T}(\mathcal{V})} \llbracket \pi \rrbracket \varphi \wp \varphi \xrightarrow{\mathcal{T}(\mathcal{V})} \delta_{\varphi^{\perp}}^{\perp} \varphi \\
 \left[\left[\frac{\pi}{\frac{\vdash^{\mathcal{V} \uplus \{x\}} \Gamma, \varphi}{\vdash^{\mathcal{V}} \Gamma, \forall x. \varphi}} \right] \right]_{\forall \mathbf{I}} &= \Gamma^{\perp} \xrightarrow{\mathcal{T}(\mathcal{V})} \text{co}_{\Gamma^{\perp}} \uparrow \Gamma^{\perp} \xrightarrow{\mathcal{T}(\mathcal{V})} \uparrow (\forall \mathbf{I}(\llbracket \pi \rrbracket)) \uparrow \forall x. \varphi \\
 \left[\left[\frac{\pi}{\frac{\vdash^{\mathcal{V}} \Gamma, \varphi[t/x]}{\vdash^{\mathcal{V}} \Gamma, \exists x. \varphi}} \right] \right]_{\forall \mathbf{I}} &= \Gamma^{\perp} \xrightarrow{\mathcal{T}(\mathcal{V})} \llbracket \pi \rrbracket \varphi[t/x] \xrightarrow{\mathcal{T}(\mathcal{V})} \exists x. \varphi \xrightarrow{\mathcal{T}(\mathcal{V})} \exists x. \varphi \xrightarrow{\mathcal{T}(\mathcal{V})} ? \exists x. \varphi
 \end{aligned}$$

■ **Figure 10** Interpretation of the remaining rules of LK.

$$\begin{array}{ccccc}
 \! \mathcal{A} \rightarrow \! \! \mathcal{A} & \! \mathcal{A} \rightarrow \! \mathcal{A} \otimes \! \mathcal{A} & ? \mathcal{A} \rightarrow \! ? \mathcal{A} & \! \mathcal{A} \otimes \! \mathcal{B} \rightarrow \! (\mathcal{A} \otimes \mathcal{B}) & \! \mathcal{A} \wp \! \mathcal{B} \rightarrow \! (\mathcal{A} \wp \mathcal{B}) \\
 ((i, j), a) \mapsto (i, (j, a)) & (2i, a) \mapsto (1, (i, a)) & (i, (j, a)) \mapsto (j, (i, a)) & (j, (i, a)) \mapsto (i, (j, a)) & (j, (i, a)) \mapsto (i, (j, a)) \\
 & (2i + 1, a) \mapsto (2, (i, a)) & & &
 \end{array}$$

■ **Figure 11** Some win-isos with exponentials whose lifting are used in the interpretation.

get $\text{co}_{\forall x. \varphi} : \forall x. \llbracket \varphi \rrbracket_{\mathcal{V}} \rightarrow \! \! \forall x. \llbracket \varphi \rrbracket_{\mathcal{V}}$ as a particular case of $\! \mathcal{A} \rightarrow \! \! \mathcal{A}$ from Figure 11. We get $\text{co}_{\varphi \wedge \psi}$ and $\text{co}_{\varphi \vee \psi}$ by induction and composition with $\! \mathcal{A} \otimes \! \mathcal{B} \rightarrow \! (\mathcal{A} \otimes \mathcal{B})$, $\! \mathcal{A} \wp \! \mathcal{B} \rightarrow \! (\mathcal{A} \wp \mathcal{B})$.

Finally, $\text{co}_{? \exists x. \llbracket \varphi \rrbracket_x}$ is obtained analogously to the contraction on the right of Figure 9.

► **Lemma 43.** *For any $(\mathcal{V} \uplus \{x\})$ -game \mathcal{A} , there is a winning $\mu_{\mathcal{A}, x} : \exists x. \! \mathcal{A} \xrightarrow{\mathcal{V}\text{-Ga}} \! \exists x. \mathcal{A}$.*

Proof. After the unique minimal \forall move (on the left hand side), the strategy simultaneously plays all the (i, \exists) (on the right hand side) with annotation \forall ; then proceeds as $\omega_{\mathcal{A}}$. ◀

We get $\text{co}_{? \exists x. \llbracket \varphi \rrbracket_x}$ by induction, post-composition with $? \mu_{\llbracket \varphi \rrbracket, x}$ and distribution of $?$ over $\!$.

► **Proposition 44.** *For any $\varphi \in \text{Form}_{\Sigma}(\mathcal{V})$, there is a winning $\text{co}_{\llbracket \varphi \rrbracket_{\mathcal{V}}} : \llbracket \varphi \rrbracket_{\mathcal{V}} \xrightarrow{\mathcal{V}\text{-Ga}} \! \llbracket \varphi \rrbracket_{\mathcal{V}}$.*

Combining Proposition 44 with other primitives (including $\! \mathcal{A} \rightarrow \! \mathcal{A}$, playing copy-cat between \mathcal{A} and the 0th copy on the left, closed under receptivity), we get $\delta_{\llbracket \varphi \rrbracket_{\mathcal{V}}} : \llbracket \varphi \rrbracket_{\mathcal{V}} \rightarrow \! \llbracket \varphi \rrbracket_{\mathcal{V}} \otimes \llbracket \varphi \rrbracket_{\mathcal{V}}$ for $\varphi \in \text{Form}_{\Sigma}(\mathcal{V})$. We complete the interpretation in Figure 10, omitting W, which is by post-composition with $e_{\mathcal{A}}$ and silently using the isomorphism between winning Σ -strategies from 1 to $\Gamma \wp \mathcal{A}$ and from Γ^{\perp} to \mathcal{A} . This concludes the proof of Theorem 14.

6 Conclusion

For LK there is no hope of preserving unrestricted cut reduction without collapsing to a boolean algebra [13]. There are non-degenerate models for classical logic with an involutive negation, *e.g.* Führman and Pym's *classical categories* [9] with reduction only preserved in a lax sense; but our model does not preserve reduction even in this weaker sense. Besides, our semantics is infinitary: from the *structural dilemma* in [8] we obtained a proof of some $\exists x. \varphi$ with φ quantifier-free (no \forall bélard moves) yielding an infinite Σ -strategy (see Appendix B).

Both phenomena could be avoided by adopting a polarized model, abandoning however our faithfulness to the raw Herbrand content of proofs. It is a fascinating open question whether one can find a non-polarized model of classical first-order logic that remains finitary – this is strongly related to the actively investigated question of finding a strongly normalizing reduction strategy on syntaxes for expansion trees [15, 21, 16].

References

- 1 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.
- 2 Franco Barbanera and Stefano Berardi. A symmetric lambda calculus for "classical" program extraction. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software, International Conference TACS '94, Sendai, Japan, April 19-22, 1994, Proceedings*, volume 789 of *Lecture Notes in Computer Science*, pages 495–515. Springer, 1994. doi:10.1007/3-540-57887-0_112.
- 3 Michael Barr. *-autonomous categories and linear logic. *Mathematical Structures in Computer Science*, 1(2):159–178, 1991.
- 4 Simon Castellan, Pierre Clairambault, Silvain Rideau, and Glynn Winskel. Games and strategies as event structures. *Logical Methods in Computer Science*, 13(3), 2017.
- 5 Pierre Clairambault, Julian Gutierrez, and Glynn Winskel. The winning ways of concurrent games. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012*, pages 235–244. IEEE, 2012.
- 6 J.R.B. Cockett and R.A.G. Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133–173, 1997.
- 7 Thierry Coquand. A semantics of evidence for classical arithmetic. *J. Symb. Log.*, 60(1):325–337, 1995.
- 8 Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. A new deconstructive logic: Linear logic. *J. Symb. Log.*, 62(3):755–807, 1997.
- 9 Carsten Führmann and David J. Pym. On categorical models of classical logic and the geometry of interaction. *Mathematical Structures in Computer Science*, 17(5):957–1027, 2007.
- 10 Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische zeitschrift*, 39(1):176–210, 1935.
- 11 Philipp Gerhardy and Ulrich Kohlenbach. Extracting herbrand disjunctions by functional interpretation. *Arch. Math. Log.*, 44(5):633–644, 2005.
- 12 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
- 13 Jean-Yves Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.
- 14 Kurt Gödel. Über eine bisher noch nicht benützte erweiterung des finiten standpunktes. *dialectica*, 12(3-4):280–287, 1958.
- 15 Willem Heijltjes. Classical proof forestry. *Ann. Pure Appl. Logic*, 161(11):1346–1366, 2010.
- 16 Stefan Hetzl and Daniel Weller. Expansion trees with cut. *CoRR*, abs/1308.0428, 2013.
- 17 Ulrich Kohlenbach. On the no-counterexample interpretation. *J. Symb. Log.*, 64(4):1491–1511, 1999.
- 18 Jean-Louis Krivine. Realizability in classical logic. *Panoramas et synthèses*, 27:197–229, 2009.
- 19 Olivier Laurent. Game semantics for first-order logic. *Logical Methods in Computer Science*, 6(4), 2010.
- 20 F William Lawvere. Adjointness in foundations. *Dialectica*, 23(3-4):281–296, 1969.

- 21 Richard McKinley. Proof nets for Herbrand's theorem. *ACM Trans. Comput. Log.*, 14(1):5, 2013.
- 22 Paul-André Mellies. Categorical semantics of linear logic. *Panoramas et synthèses*, 27:15–215, 2009.
- 23 Paul-André Mellies and Samuel Mimram. Asynchronous games: Innocence without alternation. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *LNCS*, pages 395–411. Springer, 2007.
- 24 Dale A Miller. A compact representation of proofs. *Studia Logica*, 46(4):347–370, 1987.
- 25 Samuel Mimram. The structure of first-order causality. *Mathematical Structures in Computer Science*, 21(1):65–110, 2011.
- 26 Michel Parigot. Lambda-my-calculus: An algorithmic interpretation of classical natural deduction. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning, International Conference LPAR'92, St. Petersburg, Russia, July 15-20, 1992, Proceedings*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 1992. doi:10.1007/BFb0013061.
- 27 Silvain Rideau and Glynn Winskel. Concurrent strategies. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011*, pages 409–418, 2011.
- 28 Robert A. G. Seely. Hyperdoctrines, natural deduction and the Beck condition. *Math. Log.*, 29(10):505–542, 1983.
- 29 Robert AG Seely. *Linear logic, *-autonomous categories and cofree coalgebras*. Ste. Anne de Bellevue, Quebec: CEGEP John Abbott College, 1987.
- 30 J.R. Shoenfield. *Mathematical Logic*. Addison-Wesley, 1967.
- 31 Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets*, volume 255 of *LNCS*, pages 325–392. Springer, 1986.

A Counter-examples

In this section, we detail a few counter-examples referred to in the main text.

► **Example 45.** The neutral Σ -strategies $\sigma_1 = \begin{matrix} e_1^{e_1} & e_2^{e_2} \\ \forall & \exists \\ e_3^{f(e_1)} \end{matrix}$ and $\sigma_2 = \begin{matrix} e_1^{e_1} & e_2^{e_2} \\ \forall & \exists \\ e_3^{f(e_2)} \end{matrix}$, have no meet.

Assume they have a meet σ . Necessarily, since $e_1^{e_1} e_2^{e_2} \preceq \sigma_1, \sigma_2$, then σ must comprise the events-with-annotations $e_1^{e_1}$ and $e_2^{e_2}$. But we also have

$$\begin{matrix} e_1^c & & e_2^c \\ \forall & & \exists \\ e_3^{f(c)} \end{matrix} \preceq \sigma_1, \sigma_2$$

for any constant symbol c . Therefore, σ must also include event-with-annotation e_3^c . But t must be an instance of $f(e_1), f(e_2)$; and must instantiate to $f(c)$ for all constant symbol c . So t must have the form $f(e)$ for some $e \in [e_3]$, i.e., $e \in \{e_1, e_2, e_3\}$. It is direct to check that none of those options gives a neutral Σ -strategy that is below both σ_1 and σ_2 for \preceq .

► **Example 46.** Consider $\sigma : \forall_1 1 \multimap \forall_2 \forall_3 1$ and $\tau : \forall_2 \forall_3 1 \multimap \forall_4 1$ two Σ -strategies:

$$\begin{matrix} \forall_1 1 \multimap \forall_2 \forall_3 1 & \forall_2 \forall_3 1 \multimap \forall_4 1 \\ \forall_2 & \forall_4 \\ \forall_3 & \exists_2^{\forall_4} \\ \exists_1^{\forall_3} & \forall_5 \\ & \exists_3 \end{matrix}$$

where we omit the annotation of negative events, forced by Σ -receptivity.

Their composition has $\forall_4 \rightarrow \exists_1^c$, which is not a minimal strategy since c does not have \forall_4 as a free variable.

This counter-example also means that we do not have the adjunction expected from categorical logic $\exists_{\mathcal{V},x} \dashv \mathcal{T}(w_{\mathcal{V},x}) \dashv \forall_{\mathcal{V},x}$. More precisely, Lemma 40 cannot be strengthened into an equality. Indeed, note that $\tau = \mathbb{M}_{(\forall_2 \forall_3 1),1}^x (\exists_2^x \rightarrow \exists_3^c)$. On the other hand, $\tau \odot \sigma = \forall_4 \rightarrow \exists_1^c$, which cannot be of the form $\mathbb{M}_{\forall_1 1,1}^x$ – this construction would put no causal link from \forall_4 to \exists_1^c , since c does not involve the variable x .

The intuition behind this failure is that $\mathbb{M}_{A,B}^x$ only introduces causal links that follow occurrences of a variable x . However, after composition, we may end up with Σ -strategies that are not *minimal*, i.e., they have immediate causal links not reflecting directly a syntactic dependency. In other words, in order to get an adjunction as one would expect, only the term information would have to be retained – but our interpretation remembers more.

B Non-finiteness of the interpretation

From the infinitary primitives in the interpretation, it is natural to expect the interpretation to be infinitary. It was surprisingly difficult to find such an example, however one can do so by revisiting standard pathological examples in the proof theory of classical logic, having arbitrarily large normal forms.

More precisely, we construct an LK proof of the formula $\exists x. \top$ whose interpretation is infinite, despite the fact that there is no move by \forall bélard in the game.

Our starting point is the following proof:

$$\varpi = \frac{\frac{\text{Ax} \frac{}{\vdash \varphi, \varphi^\perp} \quad \text{Ax} \frac{}{\vdash \varphi, \varphi^\perp}}{\wedge \text{I} \frac{}{\vdash \varphi \wedge \varphi, \varphi^\perp, \varphi^\perp}} \quad \frac{\frac{\text{Ax} \frac{}{\vdash \varphi, \varphi^\perp} \quad \text{Ax} \frac{}{\vdash \varphi, \varphi^\perp}}{\wedge \text{I} \frac{}{\vdash \varphi, \varphi, \varphi^\perp \wedge \varphi^\perp}} \quad \text{C} \frac{}{\vdash \varphi, \varphi^\perp \wedge \varphi^\perp}}{\text{CUT} \frac{}{\vdash \varphi \wedge \varphi, \varphi^\perp \wedge \varphi^\perp}}}{\vdash \varphi \wedge \varphi, \varphi^\perp \wedge \varphi^\perp}$$

This proof is referred to in [8] as a *structural dilemma*. There are two ways to push the CUT beyond contraction, as the two proofs interact, and try to duplicate one another. This is an example of a proof where unrestricted cut reduction does not necessarily terminate; and which has infinitely large cut-free forms.

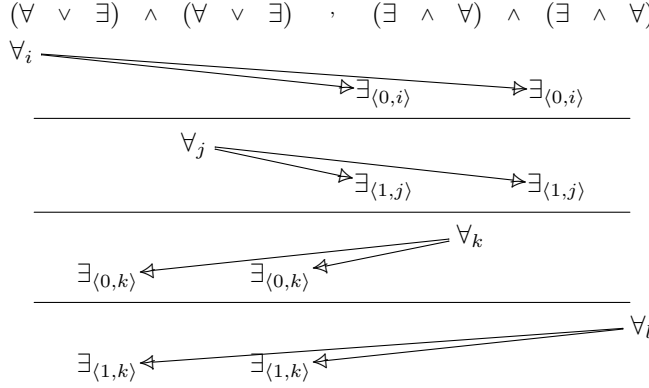
In order to construct a proof with an infinite interpretation, we will start with this proof, with $\varphi = \forall x. \perp \vee \exists y. \top$, which to shorten notations we will just write as $\forall \vee \exists$.

Omitting details, here is the interpretation of the left branch of ϖ (we omit term annotations, which always coincide with the unique predecessor for Eloïse's moves).

$$\left[\left[\frac{\text{Ax} \frac{}{\vdash \varphi, \varphi^\perp} \quad \text{Ax} \frac{}{\vdash \varphi, \varphi^\perp}}{\wedge \text{I} \frac{}{\vdash \varphi \wedge \varphi, \varphi^\perp, \varphi^\perp}} \quad \frac{\frac{\text{Ax} \frac{}{\vdash \varphi, \varphi^\perp} \quad \text{Ax} \frac{}{\vdash \varphi, \varphi^\perp}}{\wedge \text{I} \frac{}{\vdash \varphi, \varphi, \varphi^\perp \wedge \varphi^\perp}} \quad \text{C} \frac{}{\vdash \varphi, \varphi^\perp \wedge \varphi^\perp}}{\text{CUT} \frac{}{\vdash \varphi \wedge \varphi, \varphi^\perp \wedge \varphi^\perp}} \right] \right] =$$

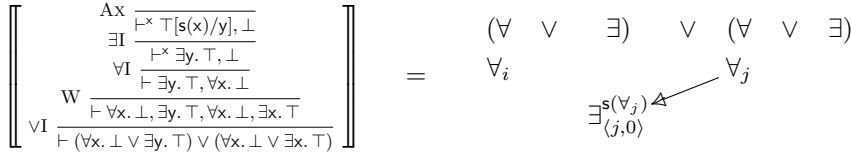
5:20 The True Concurrency of Herbrand's Theorem

The second branch of ϖ is symmetric, so we do not make it explicit. Now, we interpret the CUT rule and the composition yields $\llbracket \varpi \rrbracket$ below.

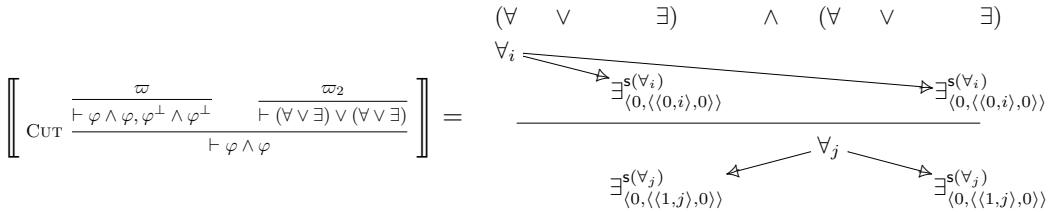


It is interesting to note that although ϖ has arbitrarily large cut-free forms, the corresponding strategy only plays finitely many \exists moves for every \forall move. However, we are on the right path to finding an infinitary Σ -strategy.

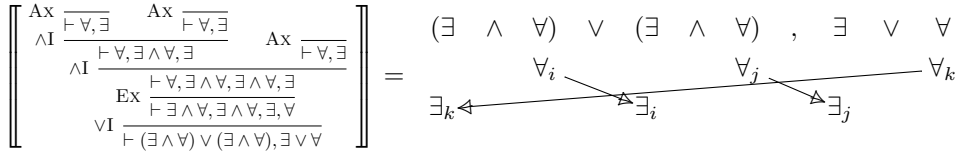
The next step is to set (with s some unary function symbol) the proof ϖ_2 below with interpretation



We now use these to compute the interpretation of ϖ_3 , a cut between ϖ and ϖ_2 :

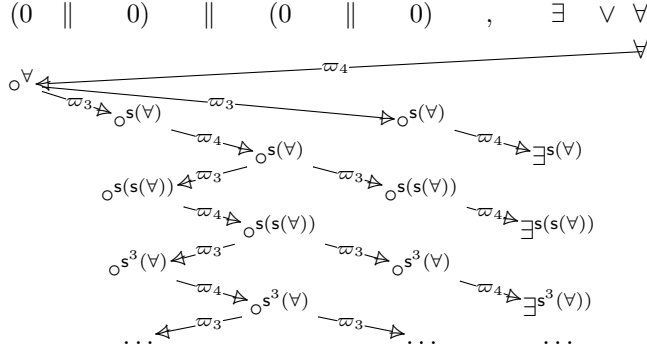


We are almost there. It suffices now to note that ϖ_3 provides a proof of $(\exists x. \top \implies \exists x. \top) \wedge (\exists x. \top \implies \exists x. \top)$. These two implications can be *composed* by cutting ϖ_3 against the following proof ϖ_4 :



Write ϖ_5 for the proof of $\exists x. \top \vee \forall y. \perp$ obtained by cutting ϖ_3 and ϖ_4 . The interpretation of ϖ_5 is the composition of $\llbracket \varpi_3 \rrbracket$ and $\llbracket \varpi_4 \rrbracket$, which triggers the feedback loop causing the infiniteness phenomenon. We display below the corresponding interaction. For the “synchronised” part of formulas, we will use 0 for components resulting from matching dual quantifiers, and \parallel for components resulting for matching dual propositional connectives.

We write \circ for synchronized events (i.e., of neutral polarity), and omit copy indices, which get very unwieldy. For readability, we also annotate the immediate causal links with the sub-proof that they originate from, i.e., ϖ_3 or ϖ_4 .



Therefore, after hiding, Eloïse responds to an initial \forall bélar move ∇ by playing simultaneously all $\exists^{s^n(\nabla)}$, for $n \geq 1$. Finally, cutting ϖ_5 against a proof of $\exists x. \top$ playing a constant symbol 0, we get a proof ϖ_6 of $\vdash \exists x. \top$ whose interpretation plays simultaneously all $\exists^{s^n(0)}$ for $n \geq 1$.

C Compactness

Restricting any winning Σ -strategy $\sigma : \llbracket \varphi \rrbracket$ to $\llbracket \varphi \rrbracket^\exists$ (ignoring \forall bélar's replications) yields $\sigma^\exists : \llbracket \varphi \rrbracket^\exists$, not necessarily finite. Yet, we will show that it has a finite *top-winning* sub-strategy.

A game \mathcal{A} is a **prefix** of \mathcal{B} if $|A| \subseteq |B|$, and all the structure coincides on $|A|$. Notice that $\llbracket \varphi \rrbracket^\exists$ embeds (subject to renaming) as a prefix of $\llbracket \varphi \rrbracket$. Keeping the renaming silent, we have:

► **Lemma 47.** *For any winning $\sigma : \llbracket \varphi \rrbracket$, setting*

$$|\sigma^\exists| = \{a \in |\sigma| \mid [a]_\sigma \subseteq \llbracket \varphi \rrbracket^\exists\}$$

and inheriting the order, polarity and labelling from σ , we obtain $\sigma^\exists : \llbracket \varphi \rrbracket^\exists$ a winning Σ -strategy.

Proof. Most conditions are direct. For $\sigma^\exists : \llbracket \varphi \rrbracket^\exists$ winning we use that for any \exists -maximal $x \in \mathcal{C}^\infty(\sigma^\exists)$, $x \in \mathcal{C}^\infty(\sigma)$ \exists -maximal as well: this follows from $\llbracket \varphi \rrbracket^\exists$ being itself \exists -maximal in $\llbracket \varphi \rrbracket$. ◀

As mentioned above, the extracted σ^\exists may not be finite! Indeed there are classical proofs for which our interpretation yields infinite strategies, even after removing \forall bélar's replications (see Appendix B). This reflects the usual issues one has in getting strong normalization in a proof system for classical logic [8] without enforcing too much sequentiality as with a negative translation.

Despite this, the compactness theorem for propositional logic entails that we can always extract a finite top-winning sub-strategy. For $\sigma : \llbracket \varphi \rrbracket^\exists$ any Σ -strategy, we denote $\mathcal{C}^\forall(\sigma)$ the set of \forall -**maximal** configurations of σ , i.e., they can only be extended in σ by Eloïse moves – inheriting all structure from σ they correspond to its *sub-strategies*, as they are automatically receptive. The proof relies on:

5:22 The True Concurrency of Herbrand's Theorem

► **Lemma 48.** *Let X be a directed set of \forall -maximal configurations. Then, $\mathcal{W}_{\llbracket\varphi\rrbracket^\exists}(\bigcup X)$ is logically equivalent to $\bigvee_{x \in X} \mathcal{W}_{\llbracket\varphi\rrbracket^\exists}(x)$.*

Proof. By induction on φ , using simple logical equivalences and that if $x_1 \subseteq x_2$ are \forall -maximal, then $\mathcal{W}_{\llbracket\varphi\rrbracket^\exists}(x_1)$ implies $\mathcal{W}_{\llbracket\varphi\rrbracket^\exists}(x_2)$. ◀

We complete the proof. For $\sigma : \llbracket\varphi\rrbracket^\exists$ winning, by the lemma above the (potentially infinite) disjunction of finite formulas

$$\bigvee_{x \in \mathcal{C}^\forall(\sigma)} \mathcal{W}_{\llbracket\varphi\rrbracket^\exists}(x)[\lambda_\sigma]$$

is a tautology. By the compactness theorem there is a finite $X = \{x_1, \dots, x_n\} \subseteq \mathcal{C}^\forall(\sigma)$ such that $\bigvee_{x \in X} \mathcal{W}_{\llbracket\varphi\rrbracket^\exists}(x)[\lambda_\sigma]$ is a tautology – *w.l.o.g.* X is directed as $\mathcal{C}^\forall(\sigma)$ is closed under union. By Lemma 48 again, $\mathcal{W}_{\llbracket\varphi\rrbracket^\exists}(\bigcup X)[\lambda_\sigma]$ is a tautology. So, restricting σ to events $\bigcup X$ gives a top-winning finite sub-strategy of σ .


Although this argument is non-constructive, the extraction of a finite sub-strategy can still be performed effectively: Σ -strategies and their operations can be effectively presented, and the finite top-winning sub-strategy can be computed by Markov's principle.

Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities

Carlo Angiuli

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA


cangiuli@cs.cmu.edu

 <https://orcid.org/0000-0002-9590-3303>

Kuen-Bang Hou (Favonia)¹

School of Mathematics, Institute for Advanced Study, Princeton, NJ, USA


favonia@math.ias.edu

 <https://orcid.org/0000-0002-2310-3673>

Robert Harper

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

rwh@cs.cmu.edu

 <https://orcid.org/0000-0002-9400-2941>

Abstract

We present a dependent type theory organized around a Cartesian notion of cubes (with faces, degeneracies, and diagonals), supporting both fibrant and non-fibrant types. The fibrant fragment validates Voevodsky’s univalence axiom and includes a circle type, while the non-fibrant fragment includes exact (strict) equality types satisfying equality reflection. Our type theory is defined by a semantics in cubical partial equivalence relations, and is the first two-level type theory to satisfy the canonicity property: all closed terms of boolean type evaluate to either true or false.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Homotopy Type Theory, Two-Level Type Theory, Computational Type Theory, Cubical Sets

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.6

Related Version <https://arxiv.org/abs/1712.01800>

Funding This research was supported by the Air Force Office of Scientific Research through MURI grant FA9550-15-1-0053 and the National Science Foundation through grant DMS-1638352. Any opinions, findings and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of any sponsoring institution, the U.S. government or any other entity.

Acknowledgements We are greatly indebted to Steve Awodey, Marc Bezem, Evan Cavallo, Daniel Gratzer, Simon Huber, Dan Licata, Ed Morehouse, Anders Mörtberg, Andrew Pitts, Jonathan Sterling, and Todd Wilson for their contributions and advice.

¹ This author thanks the Isaac Newton Institute for Mathematical Sciences for its support and hospitality during the program “Big Proof” when part of work on this paper was undertaken. The program was supported by EPSRC grant number EP/K032208/1.



© Carlo Angiuli, Kuen-Bang Hou, and Robert Harper;
licensed under Creative Commons License CC-BY

27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 6; pp. 6:1–6:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Martin-Löf has proposed two rather different approaches to equality in dependent type theory, in the guise of his extensional [24] and intensional [25] type theories. *Extensional type theory*, particularly its realization as Nuprl’s computational type theory [2], is justified by meaning explanations in which closed terms are programs equipped with an operational semantics, and variables are considered to range over closed terms of their given type.

One consequence is that equations hold whenever they are true for all closed terms; for instance, $n : \mathbf{nat}, m : \mathbf{nat} \gg n + m \doteq m + n \in \mathbf{nat}$ as a judgmental equality because $N + M$ and $M + N$ compute the same natural number for any closed natural numbers N, M . Another consequence is known as *equality reflection*: the equality type $\mathbf{Eq}_A(M, N)$ has at most one element, and is inhabited if and only if $M \doteq N \in A$ judgmentally.

In contrast, in *intensional type theory*, judgmental equality is precisely β - (and at certain types, η -) equivalence, and context variables are treated as additional axioms whose form is indeterminate. The identity type $\mathbf{Id}_A(M, N)$ mediates equality reasoning; in an empty context it is inhabited by a single element if and only if $M \equiv N : A$ judgmentally, but in non-empty contexts includes additional equalities such as $n : \mathbf{nat}, m : \mathbf{nat} \vdash P : \mathbf{Id}_{\mathbf{nat}}(n + m, m + n)$, which does not hold judgmentally for variables n, m .

Traditional type theories, extensional or intensional, are constructive in the sense that they admit an interpretation of proofs as programs, often distilled into the *canonicity property* that closed elements of type \mathbf{bool} evaluate and are judgmentally equal to either \mathbf{true} or \mathbf{false} . In computational type theory, this is the very definition of $M \in \mathbf{bool}$ (see Theorem 15), while in intensional type theory, canonicity can be verified by a metatheoretic argument.

Homotopy type theory [29] extends intensional type theory with a number of axioms, including Voevodsky’s univalence axiom [31] and higher inductive types [23]. These axioms are justified by mathematical models interpreting types as spaces (e.g., simplicial sets [20] or fibrant objects in a model category [10]), elements of types as points, and identity types as path spaces. In such models, homotopy type theory serves as a framework for synthetic homotopy theory [29], in which higher inductive types provide concrete homotopy types (e.g., n -spheres), the rules of the identity type assert that all constructions respect paths, and univalence asserts moreover that all constructions are invariant under homotopy equivalence.

Despite the success of homotopy type theory as a medium for synthetic results in homotopy theory [11, 30, 14], it is believed that certain objects – famously, semi-simplicial types – cannot be constructed without reference to some notion of exact equality stricter than paths [8, 33]. Because exact equality does not respect paths, any theory with both exact equality and paths must therefore stratify types into *fibrant types* that respect paths, and *non-fibrant types* that do not. Candidate such *two-level type theories* include the Homotopy Type System (HTS) of Voevodsky [33] and the two-level type theory of Altenkirch et al. [3].

Critically, homotopy type theory and existing two-level type theories lack the aforementioned canonicity property, because the ordinary judgmental equalities of intensional type theory do not apply to uses of the univalence axiom or paths in higher inductive types. Nor are they known to satisfy the weaker *homotopy canonicity* property that for any closed $M : \mathbf{bool}$ there exists a proof $P : \mathbf{Id}_{\mathbf{bool}}(M, \mathbf{true})$ or $P : \mathbf{Id}_{\mathbf{bool}}(M, \mathbf{false})$ [32].

1.1 Contributions

We define a two-level computational type theory satisfying the canonicity property, whose fibrant types include a cumulative hierarchy of univalent universes of fibrant types, universes of non-fibrant types, dependent function, dependent pair, and path types, and whose non-fibrant types include also exact equality types with equality reflection.

Our type theory is the first two-level type theory with canonicity, and the second univalent type theory with canonicity, after the cubical type theory of Cohen et al. [17]. Like Cohen et al. [17], our type theory is inspired by a model of homotopy type theory in cubical sets [12], and represents n -dimensional cubes as terms parametrized by n variables ranging over a formal interval. However, the fibrant fragment of our type theory differs from Cohen et al. [17] by endowing the interval with less (namely, Cartesian) structure, and defining fibrancy with a substantially different uniform Kan condition. Thus we affirmatively resolve the open question of whether Cartesian interval structure constructively models univalence [18, 22].

In the spirit of Martin-Löf’s meaning explanations [24], we define the judgments of type theory as relations on programs in an untyped programming language. In Section 2, we define a λ -calculus extended by nominal constants representing elements of a formal interval object [26]. In Section 3, we define a cubical generalization of Allen’s partial equivalence relation (PER) semantics of Nuprl [1], sufficient to describe non-fibrant types and their elements at all dimensions. In Section 4, we define fibrant types as non-fibrant types equipped with two Kan operations, called coercion and homogeneous composition. In Sections 5 and 6 we summarize the semantics of each type former, and provide valid rules of inference. We conclude in Section 7 with comparisons to related work.

Full details and proofs for our construction are available in our associated preprint [7]. Our type theory is currently being implemented in the REDPRL proof assistant [28], in which we have already formalized a proof of univalence (<https://git.io/vFjUQ>).

2 Programming language

We begin by defining an untyped *cubical programming language*, a call-by-name λ -calculus extended by nominal constants [26], whose terms serve as the types and elements of our cubical type theory. Names (or dimensions) x, y, \dots represent generic elements of an abstract interval \mathbb{I} with two constant elements (or endpoints) $0, 1$. Given any two finite sets of names Ψ, Ψ' , a *dimension substitution* $\psi : \Psi' \rightarrow \Psi$ sends each name in Ψ' to $0, 1$, or a name in Ψ . We write $\langle r/x \rangle : \Psi \rightarrow (\Psi, x)$ for the dimension substitution sending x to $r \in \Psi \cup \{0, 1\}$ and constant on Ψ . Given $\psi : \Psi' \rightarrow \Psi$ and a term M whose free names are contained in Ψ , we write $M\psi$ for the term obtained by replacing each $x \in \Psi$ in M with $\psi(x)$.

Geometrically, a term M with free dimension names in Ψ (henceforth, a Ψ -dimensional term) represents a $|\Psi|$ -dimensional cube – a point ($|\Psi| = 0$), line ($|\Psi| = 1$), square ($|\Psi| = 2$), and so forth. Dimension substitutions are compositions of permutations, *face maps* $\langle 0/x \rangle, \langle 1/x \rangle : \Psi \rightarrow (\Psi, x)$, *diagonal maps* $\langle y/x \rangle : (\Psi, y) \rightarrow (\Psi, x, y)$, and (silent) *degeneracy maps* $(\Psi, y) \rightarrow \Psi$, and perform the corresponding geometric operation when applied to a term M . Below, we illustrate the faces of a square M in dimensions $\{x, y\}$; note that the bottom endpoint of the left face and the left endpoint of the bottom face are drawn as a single point, because $\langle 0/x \rangle \langle 1/y \rangle = \langle 1/y \rangle \langle 0/x \rangle$.

$$\begin{array}{ccc}
 \begin{array}{c} x \\ \downarrow \\ y \end{array} & \begin{array}{ccc} M\langle 0/x \rangle \langle 0/y \rangle & \xrightarrow{M\langle 0/y \rangle} & M\langle 1/x \rangle \langle 0/y \rangle \\ \downarrow M\langle 0/x \rangle & & \downarrow M\langle 1/x \rangle \\ M\langle 0/x \rangle \langle 1/y \rangle & \xrightarrow{M\langle 1/y \rangle} & M\langle 1/x \rangle \langle 1/y \rangle \end{array} \\
 & & M
 \end{array}$$

This notion of cubes is *Cartesian* because sets of names and dimension substitutions form a free finite-product category generated by the two endpoint maps $\langle 0/x \rangle, \langle 1/x \rangle : \emptyset \rightarrow \{x\}$ [22, 9, 15]. In contrast, Cohen et al. [17] equip the interval with a De Morgan algebra structure also containing *connections* $\langle (x \wedge y)/y \rangle, \langle (x \vee y)/y \rangle : (\Psi, x, y) \rightarrow (\Psi, y)$ and *reversals* $\langle (1 - y)/y \rangle : (\Psi, y) \rightarrow (\Psi, y)$. Cartesian cubes are appealing for their ubiquity and simplicity: dimensions behave like structural variables (with exchange, weakening, and contraction) and have a trivial equational theory (as opposed to De Morgan laws).

Following Martin-Löf’s meaning explanations [24], we only give operational meaning to closed terms, and consider term variables to range over closed terms of their given types. However, we cannot treat dimension names as ranging only over $\{0, 1\}$ – such a semantics would enforce *uniqueness of identity proofs*, by equating all lines whose boundaries coincide.

We therefore define a deterministic small-step operational semantics on terms with no free term variables, but any number of free dimension names. We write $V \text{ val}$ for values, $M \mapsto M'$ when M takes one step of computation to M' , and $M \Downarrow V$ (M evaluates to V), when $M \mapsto^* V$ (in zero or more steps) and $V \text{ val}$. Notably, the operational semantics are not stable under dimension substitution: because face and diagonal maps can expose new simplifications, we have neither (1) if $V \text{ val}$ then $V\psi \text{ val}$, nor (2) if $M \mapsto^* M'$ then $M\psi \mapsto^* M'\psi$. Consider the circle (Section 5.2), inductively generated by a point base and a line loop_x . We arrange that the faces of loop_x are base by including an operational step $(\text{loop}_x)\langle 0/x \rangle = \text{loop}_0 \mapsto \text{base}$. On the other hand, $\text{loop}_x \text{ val}$ because it is a constructor, contradicting (1). Maps out of the circle are determined by a point P (the image of base) and an abstracted line $x.L$ (the image of loop_x). Thus $\mathbb{S}^1\text{-elim}_{c.A}(\text{loop}_x; P, x.L) \mapsto L$ but

$$\begin{aligned} (\mathbb{S}^1\text{-elim}_{c.A}(\text{loop}_x; P, x.L))\langle 0/x \rangle &= \mathbb{S}^1\text{-elim}_{c.A\langle 0/x \rangle}(\text{loop}_0; P\langle 0/x \rangle, x.L) \\ &\mapsto \mathbb{S}^1\text{-elim}_{c.A\langle 0/x \rangle}(\text{base}; P\langle 0/x \rangle, x.L) \\ &\mapsto P\langle 0/x \rangle \end{aligned}$$

where L and $P\langle 0/x \rangle$ are a priori unrelated, contradicting (2). Fortunately, most rules of the operational semantics are in fact *cubically stable*, or preserved by dimension substitutions: for instance, $(\text{loop}_0)\psi \mapsto \text{base}\psi$ for all $\psi : \Psi' \rightarrow \Psi$. We write $M \mapsto_{\square} M'$ when $M\psi \mapsto M'\psi$ for all $\psi : \Psi' \rightarrow \Psi$, and $V \text{ val}_{\square}$ when $V\psi \text{ val}$ for all $\psi : \Psi' \rightarrow \Psi$.

We include some operational semantics rules in Fig. 1, but omit the many rules pertaining to the Kan operations (defined in Section 4), as well as rules that evaluate the principal argument of an elimination form (for example, $\text{app}(M, N) \mapsto \text{app}(M', N)$ when $M \mapsto M'$). We adopt the convention that a, b, c, \dots are term variables, x, y, z, \dots are dimension names, and r, r', r_i are dimension expressions (names x or constants 0, 1).

3 Cubical PER semantics

Type theory is built on the judgments of typehood (and equality of types) and membership in a type (and equality of members in a type). Intensional type theories – including homotopy type theory and the cubical type theory of Cohen et al. [17] – typically define these judgments inductively by a collection of syntactic inference rules. We instead define these judgments semantically as partial equivalence relations (PERs, or symmetric and transitive relations) over terms of the language described in Section 2. Such an approach can be seen as a mathematically precise reading of Martin-Löf’s meaning explanations of type theory [24], or as a relational semantics of type theory in the style of Tait [27], and is the approach adopted by Nuprl [2]. The role of inference rules is therefore not definitional, but rather to summarize desirable properties validated by the semantics.

$(a:A) \rightarrow B \text{ val}_{\mathbb{D}}$ $\lambda a.M \text{ val}_{\mathbb{D}}$ $\text{app}(\lambda a.M, N) \mapsto_{\mathbb{D}} M[N/a]$ $(a:A) \times B \text{ val}_{\mathbb{D}}$ $\langle M, N \rangle \text{ val}_{\mathbb{D}}$ $\text{fst}(\langle M, N \rangle) \mapsto_{\mathbb{D}} M$ $\text{snd}(\langle M, N \rangle) \mapsto_{\mathbb{D}} N$ $\text{Path}_{x.A}(M, N) \text{ val}_{\mathbb{D}}$ $\langle x \rangle M \text{ val}_{\mathbb{D}}$ $\langle \langle x \rangle M \rangle @r \mapsto_{\mathbb{D}} M \langle r/x \rangle$ $\text{Eq}_A(M, N) \text{ val}_{\mathbb{D}}$ $\star \text{ val}_{\mathbb{D}}$ $\text{bool} \text{ val}_{\mathbb{D}}$ $\text{true} \text{ val}_{\mathbb{D}}$ $\text{false} \text{ val}_{\mathbb{D}}$ $\text{if}_{b.A}(\text{true}; T, F) \mapsto_{\mathbb{D}} T$	$\text{if}_{b.A}(\text{false}; T, F) \mapsto_{\mathbb{D}} F$ $\mathbb{S}^1 \text{ val}_{\mathbb{D}}$ $\text{base} \text{ val}_{\mathbb{D}}$ $\text{loop}_x \text{ val}$ $\text{loop}_\varepsilon \mapsto_{\mathbb{D}} \text{base} \quad (\varepsilon \in \{0, 1\})$ $\mathbb{S}^1\text{-elim}_{c.A}(\text{base}; P, x.L) \mapsto_{\mathbb{D}} P$ $\mathbb{S}^1\text{-elim}_{c.A}(\text{loop}_x; P, y.L) \mapsto_{\mathbb{D}} L \langle x/y \rangle$ $\mathbb{V}_x(A, B, E) \text{ val}$ $\mathbb{V}_\varepsilon(A_0, A_1, E) \mapsto_{\mathbb{D}} A_\varepsilon \quad (\varepsilon \in \{0, 1\})$ $\mathbb{V}\text{in}_x(M, N) \text{ val}$ $\mathbb{V}\text{in}_\varepsilon(M_0, M_1) \mapsto_{\mathbb{D}} M_\varepsilon \quad (\varepsilon \in \{0, 1\})$ $\mathbb{V}\text{proj}_x(\mathbb{V}\text{in}_x(M, N), F) \mapsto_{\mathbb{D}} N$ $\mathbb{V}\text{proj}_0(M, F) \mapsto_{\mathbb{D}} \text{app}(F, M)$ $\mathbb{V}\text{proj}_1(M, F) \mapsto_{\mathbb{D}} M$ $\mathcal{U}_j^\kappa \text{ val}_{\mathbb{D}} \quad (\kappa \in \{\text{pre}, \text{Kan}\})$
---	---

■ **Figure 1** Operational semantics, selected rules.

We adopt this semantical approach for multiple reasons. By defining types as relations over programs, we ensure the constructive character of the theory; for instance, it will follow from the definitions that elements of boolean type are programs that evaluate to `true` or `false` (Theorem 15). Moreover, because the meaning of open terms is given by their closed (term) substitution instances, it will naturally follow that judgmental equality is extensional and that the exact equality type satisfies equality reflection.

In Allen’s PER semantics of Nuprl [1], a type A is interpreted as a symmetric and transitive relation $\llbracket A \rrbracket$ on values; the judgment $M \doteq N \in A$ holds whenever $M \Downarrow M_0$, $N \Downarrow N_0$, and $\llbracket A \rrbracket(M_0, N_0)$ (which we henceforth write $\llbracket A \rrbracket^\Downarrow(M, N)$); and $M \in A$ whenever $M \doteq M \in A$. Thus, ignoring equality, A is defined by its set of values $\{V \text{ val} \mid \llbracket A \rrbracket(V, V)\}$, and the elements of A are the programs whose values are elements of that set. (We write \in rather than $:$ to emphasize the semantic character of these judgments.)

We generalize Nuprl’s semantics by instead interpreting types as *cubical sets*: every type has a PER of Ψ -dimensional values for every Ψ , and each $\psi : \Psi' \rightarrow \Psi$ sends its Ψ -dimensional values to its Ψ' -dimensional values. Complications arise when defining the latter functorial action. First, dimension substitutions can engender computation even on values, so the action of ψ must send V to the *value* of the program $V\psi$. Second, substitution-then-evaluation is not necessarily functorial: if $V\psi \Downarrow V'$, there is in general no relationship between the values of $V\psi\psi'$ and $V'\psi'$. Third, types are themselves programs because of dependency, and therefore suffer from the same coherence issues. We solve these issues by interpreting (Ψ -dimensional) types as *value-coherent Ψ -PERs on values*:

► **Definition 1.** A Ψ -relation α (resp., Ψ -relation on values) is a family of binary relations α_ψ for every Ψ' and $\psi : \Psi' \rightarrow \Psi$, over Ψ' -dimensional terms (resp., values). If α_ψ varies only in the choice of Ψ' and not ψ , we say α is *context-indexed* and write $\alpha_{\Psi'}$ for α_ψ .

► **Definition 2.** For any Ψ -relation on values α , define the Ψ -relation $\text{Tm}(\alpha)(M, N)$ to hold when for all $\psi_1 : \Psi_1 \rightarrow \Psi$ and $\psi_2 : \Psi_2 \rightarrow \Psi_1$, $\alpha_{\psi_1 \psi_2}^{\downarrow}$ relates pairwise $M_1 \psi_2$, $M \psi_1 \psi_2$, $N_1 \psi_2$, and $N \psi_1 \psi_2$, where $M \psi_1 \downarrow M_1$ and $N \psi_1 \downarrow N_1$.

A Ψ -relation α can be precomposed with a dimension substitution $\psi : \Psi' \rightarrow \Psi$, yielding a Ψ' -relation $(\alpha \psi)_{\psi'} := \alpha_{\psi \psi'}$.

► **Definition 3.** A Ψ -relation on values α is *value-coherent*, or $\text{Coh}(\alpha)$, when for all $\psi : \Psi' \rightarrow \Psi$, if $\alpha_{\psi}(V, V')$ then $\text{Tm}(\alpha \psi)(V, V')$.

Definition 1 captures the idea that types vary with dimension substitutions (for example, $\mathbb{S}^1\text{-elim}_{c, \mathcal{U}_j^{\text{Kan}}}(\text{loop}_x; A, x.B)$ under $\langle 0/x \rangle$), Definition 2 lifts Ψ -relations on values to arbitrary terms by substitution-then-evaluation, and Definition 3 defines functoriality of that lifting.

► **Remark.** Writing \mathbb{C} for the category of finite sets of names and dimension substitutions, a value-coherent context-indexed PER determines a functor $\mathbb{C}^{\text{op}} \rightarrow \mathbf{Set}$, and a value-coherent Ψ -PER determines a functor $(\mathbb{C}/\Psi)^{\text{op}} \rightarrow \mathbf{Set}$.

3.1 Judgments

We define the judgments of our type theory relative to a value-coherent context-indexed PER of types, each of which gives rise to another PER. In the style of Allen [1] and recently, Anand and Rahli [4], we present this data in a single relation.

► **Definition 4.** A *cubical type system* is a relation $\tau(\Psi, A_0, B_0, \varphi)$ over Ψ -dimensional values A_0, B_0 , and binary relations φ over Ψ -dimensional values, satisfying:

- Functionality: if $\tau(\Psi, A_0, B_0, \varphi)$ and $\tau(\Psi, A_0, B_0, \varphi')$ then $\varphi = \varphi'$.
- PER-valuation: if $\tau(\Psi, A_0, B_0, \varphi)$ then φ is a PER.
- Symmetry: if $\tau(\Psi, A_0, B_0, \varphi)$ then $\tau(\Psi, B_0, A_0, \varphi)$.
- Transitivity: if $\tau(\Psi, A_0, B_0, \varphi)$ and $\tau(\Psi, B_0, C_0, \varphi)$ then $\tau(\Psi, A_0, C_0, \varphi)$.
- Value-coherence: $\text{Coh}(\{(\Psi, A_0, B_0) \mid \tau(\Psi, A_0, B_0, \varphi)\})$.

The first three components of τ define a Ψ -PER for every Ψ , which we write τ^{Ψ} . If $\text{Tm}(\tau^{\Psi})(A, B)$, then the fourth component of τ assigns a Ψ -PER to A, B sending each $\psi : \Psi' \rightarrow \Psi$ to the relation φ^{ψ} where $\tau^{\Psi}(\Psi', A \psi, B \psi, \varphi^{\psi})$. We write this Ψ -PER $\llbracket A \rrbracket$; it is unique by functionality, and independent from the choice of B by symmetry and transitivity.

For the remainder of this section, fix a cubical type system τ . We start by defining the closed judgments relative to τ : when are A and B equal Ψ -dimensional types, and when are M and N equal Ψ -dimensional elements of A ?

► **Definition 5.** $A \doteq B$ $\text{type}_{\text{pre}}[\Psi]$ holds when $\text{Tm}(\tau^{\Psi})(A, B)$ and $\text{Coh}(\llbracket A \rrbracket)$. We write $A \text{type}_{\text{pre}}[\Psi]$ for $A \doteq A \text{type}_{\text{pre}}[\Psi]$.

► **Definition 6.** $M \doteq N \in A[\Psi]$, presupposing ² $A \text{type}_{\text{pre}}[\Psi]$, when $\text{Tm}(\llbracket A \rrbracket)(M, N)$. We write $M \in A[\Psi]$ for $M \doteq M \in A[\Psi]$.

We extend the judgments to open terms by functionality: an open type (resp., elements) is a map sending equal elements of the context to equal closed types (resp., elements). The open judgments must be defined simultaneously, by induction on the length of the context.

² A presupposition is a fact that must be established before a judgment can be sensibly considered. Here, it does not make sense to demand $\text{Tm}(\llbracket A \rrbracket)(M, N)$ unless $\llbracket A \rrbracket$ is known to exist by $A \text{type}_{\text{pre}}[\Psi]$.

► **Definition 7.** $(a_1 : A_1, \dots, a_n : A_n) \text{ ctx } [\Psi]$ when $A_1 \text{ type}_{\text{pre}} [\Psi]$, $a_1 : A_1 \gg A_2 \text{ type}_{\text{pre}} [\Psi]$, \dots , and $a_1 : A_1, \dots, a_{n-1} : A_{n-1} \gg A_n \text{ type}_{\text{pre}} [\Psi]$.

► **Definition 8.** $a_1 : A_1, \dots, a_n : A_n \gg B \doteq B' \text{ type}_{\text{pre}} [\Psi]$, presupposing $(a_1 : A_1, \dots, a_n : A_n) \text{ ctx } [\Psi]$, when for any $\psi : \Psi' \rightarrow \Psi$, $N_1 \doteq N'_1 \in A_1 \psi [\Psi']$, $N_2 \doteq N'_2 \in A_2 \psi [N_1/a_1] [\Psi']$, \dots , and $N_n \doteq N'_n \in A_n \psi [N_1, \dots, N_{n-1}/a_1, \dots, a_n] [\Psi']$, when

$$B \psi [N_1, \dots, N_n/a_1, \dots, a_n] \doteq B' \psi [N'_1, \dots, N'_n/a_1, \dots, a_n] \text{ type}_{\text{pre}} [\Psi'].$$

Under the same hypotheses, $a_1 : A_1, \dots, a_n : A_n \gg M \doteq M' \in B [\Psi]$ when

$$M \psi [N_1, \dots, N_n/a_1, \dots, a_n] \doteq M' \psi [N'_1, \dots, N'_n/a_1, \dots, a_n] \in B \psi [N_1, \dots, N_n/a_1, \dots, a_n] [\Psi'].$$

Given the distinct roles of term variables and dimension names in Definition 8, it is natural for our judgments to separate the contexts $(a_1 : A_1, \dots, a_n : A_n)$ and Ψ . In REDPRL, we utilize a single mixed context of terms and dimensions, as do Cohen et al. [17].

► **Remark.** Allen’s PER semantics are an instance of our semantics, in the case that types are constant presheaves and terms have no free dimension names. If M, N, A , and B have no free dimensions, then $A \doteq B \text{ type}_{\text{pre}} [\Psi]$ if and only if $\tau^\downarrow(\Psi', A, B, \llbracket A \rrbracket_{\Psi'})$ for all Ψ' , and $M \doteq N \in A [\Psi]$ if and only if $(\llbracket A \rrbracket_{\Psi'})^\downarrow(M, N)$ for all Ψ' .

3.2 Properties of Judgments

The main result of this paper is the construction of a cubical type system closed under a variety of type formers. However, many global properties of judgments hold in *any* cubical type system. For instance, equality judgments are all symmetric, transitive, and closed under dimension substitution (if $\mathcal{J} [\Psi]$ and $\psi : \Psi' \rightarrow \Psi$, then $\mathcal{J} \psi [\Psi']$). Open judgments satisfy the hypothesis (if $(\Gamma, a : A, \Gamma') \text{ ctx } [\Psi]$ then $\Gamma, a : A, \Gamma' \gg a \in A [\Psi]$) and weakening rules. Equal types have the same elements (if $A \doteq B \text{ type}_{\text{pre}} [\Psi]$ and $M \doteq N \in A [\Psi]$ then $M \doteq N \in B [\Psi]$).

To prove $M \in A [\Psi]$ in a particular cubical type system, we must compare the definition of $\llbracket A \rrbracket$ with the evaluation behavior of all dimension substitution instances of M . When all instances of M begin to evaluate in lockstep, it suffices to consider only M itself (Lemma 9); otherwise, it suffices to show that the instances of M become coherent up to equality at A , after some number of steps (Lemma 10).

► **Lemma 9** (Head expansion). *If $M' \in A [\Psi]$ and $M \mapsto_{\text{cb}}^* M'$, then $M \doteq M' \in A [\Psi]$.*

► **Lemma 10.** *Suppose that M is a Ψ -dimensional term, and we have a family of terms $\{M_\psi\}$ for each $\psi : \Psi' \rightarrow \Psi$ such that $M \psi \mapsto^* M_\psi$. If $M_\psi \doteq (M_{\text{id}_\Psi}) \psi \in A \psi [\Psi']$ for all ψ , then $M \doteq M_{\text{id}_\Psi} \in A [\Psi]$.*

Once we have established that substitution-then-evaluation of M is functorial, it follows that the instances of M are equal to the instances of its value.

► **Lemma 11.** *If $M \in A [\Psi]$, then $M \downarrow V$ and $M \doteq V \in A [\Psi]$.*

On the other hand, certain properties typical of intensional type theories are generally *not* expected to hold in our semantics. To check $M \in A [\Psi]$, one must, at minimum, show that M terminates; this is clearly undecidable, because M can be an arbitrary untyped term. Moreover, terms do not have unique types, because the meanings of types need not be disjoint. In fact, modern Nuprl has a “Base” type containing every term [4].

4 Kan types

The judgmental apparatus described in Section 3 accounts for *non-fibrant* or *pretypes* – whose paths are not necessarily composable or invertible. A pretype is *Kan fibrant*, or a Kan type, when equipped with two *Kan operations*: coercion (*coe*) and homogeneous composition (*hcom*). Coercion for a (Ψ, x) -dimensional type states that elements of $A\langle r/x \rangle$ can be coerced to $A\langle r'/x \rangle$ for any r, r' , and this operation is the identity when $r = r'$. The coercion of M is written $\text{coe}_{x.A}^{r \rightsquigarrow r'}(M)$. For example, if $M \in A\langle 0/x \rangle [\emptyset]$, then $\text{coe}_{x.A}^{0 \rightsquigarrow 1}(M) \in A\langle 1/x \rangle [\emptyset]$. Moreover, $\text{coe}_{x.A}^{0 \rightsquigarrow x}(M) \in A[x]$ is a line in A whose $\langle 0/x \rangle$ face is M (because $0 = x\langle 0/x \rangle$), and whose $\langle 1/x \rangle$ face is $\text{coe}_{x.A}^{0 \rightsquigarrow 1}(M)$.

$$\begin{array}{ccc}
 \begin{array}{c} x \\ \rightarrow \\ y \downarrow \\ M \xrightarrow{\text{coe}_{x.A}^{0 \rightsquigarrow x}(M)} \text{coe}_{x.A}^{0 \rightsquigarrow 1}(M) \end{array} & & \begin{array}{ccc} & \xrightarrow{M} & \\ \cdot & & \cdot \\ \downarrow N_0 \text{ hcom}_A^{0 \rightsquigarrow y}(M; x=0 \hookrightarrow y.N_0, x=1 \hookrightarrow y.N_1) & & \downarrow N_1 \\ & \xrightarrow{\text{hcom}_A^{0 \rightsquigarrow 1}(M; x=0 \hookrightarrow y.N_0, x=1 \hookrightarrow y.N_1)} & \\ & \text{---} & \end{array}
 \end{array}$$

Homogeneous composition is significantly more complicated, but essentially states that any open box in A (an n -cube without an interior or one of its faces) has a composite (the missing face). For example, given two lines in y , $N_0 \in A\langle 0/x \rangle [y]$ and $N_1 \in A\langle 1/x \rangle [y]$, and a line in x , $M \in A[x]$, that agrees with the y -lines when $y = 0$ ($M\langle 0/x \rangle \doteq N_\varepsilon \in A\langle \varepsilon/x \rangle [\emptyset]$ for $\varepsilon \in \{0, 1\}$), we can obtain an x -line that agrees with the y -lines when $y = 1$, written $\text{hcom}_A^{0 \rightsquigarrow 1}(M; x=0 \hookrightarrow y.N_0, x=1 \hookrightarrow y.N_1)$. Moreover, we can obtain the interior of that square, its *filler*, by composing to y rather than 1. The difficulty of homogeneous composition is that we must define arbitrary open boxes, at any dimension, in a manner that commutes with substitution. We introduce *dimension context restrictions* Ξ , or sets of pairs of dimension expressions (suggestively written as equations), to describe the spatial relationship between the faces of an open box.

► **Definition 12.** A context restriction $\overline{r_i = r'_i}$ is *valid* in Ψ when all r_i, r'_i are dimension expressions in Ψ , and either $r_i = r'_i$ for some i , or $r_i = r_j, r'_i = 0$, and $r'_j = 1$ for some i, j .

► **Definition 13.** A restricted judgment $\mathcal{J} [\Psi \mid \overline{r_i = r'_i}]$ holds when $\mathcal{J}\psi [\Psi']$ holds for every $\psi : \Psi' \rightarrow \Psi$ for which $r_i\psi = r'_i\psi$ for all i .

Restricted judgments behave as one might expect: $\mathcal{J} [\Psi \mid \emptyset]$ if and only if $\mathcal{J} [\Psi]$, $\mathcal{J} [\Psi, x \mid x = 0]$ if and only if $\mathcal{J}\langle 0/x \rangle [\Psi]$, and $\mathcal{J} [\Psi \mid 0 = 1]$ always. Crucially, they are closed under dimension substitution: if $\mathcal{J} [\Psi \mid \Xi]$ and $\psi : \Psi' \rightarrow \Psi$, then $\mathcal{J}\psi [\Psi' \mid \Xi\psi]$.

► **Definition 14.** $B \text{ type}_{\text{Kan}} [\Psi]$, presupposing $B \text{ type}_{\text{pre}} [\Psi]$, when for all $\psi : \Psi' \rightarrow \Psi$, the rules in Fig. 2 hold for $A := B\psi$. ($B \doteq B' \text{ type}_{\text{Kan}} [\Psi]$, presupposing $B \doteq B' \text{ type}_{\text{pre}} [\Psi]$, when B and B' are equipped with equal Kan operations.)

Operationally, both *hcom* and *coe* evaluate their type argument and behave according to the outermost type former. For each type former, we will first show that the formation, introduction, elimination, computation, and eta rules hold; then, using those rules, we show that if its component types are Kan, then it is Kan (for example, if $A \text{ type}_{\text{Kan}} [\Psi]$ and $a : A \gg B \text{ type}_{\text{Kan}} [\Psi]$, then $(a:A) \rightarrow B \text{ type}_{\text{Kan}} [\Psi]$). The only exceptions are exact equality types $\text{Eq}_A(M, N)$ (Section 5.5), which are not generally Kan even when A is Kan.

$$\begin{array}{c}
\frac{}{r_i = r'_i \text{ valid } [\Psi]} \\
A \text{ type}_{\text{Kan}} [\Psi] \\
M \in A [\Psi] \\
(\forall i, j) \quad N_i \doteq N_j \in A [\Psi, y \mid r_i = r'_i, r_j = r'_j] \\
(\forall i) \quad N_i \langle r/y \rangle \doteq M \in A [\Psi \mid r_i = r'_i] \\
\hline
\text{hcom}_A^{r \rightsquigarrow r'} (M; r_i = r'_i \leftrightarrow y.N_i) \in A [\Psi] \\
\equiv \begin{cases} M & \text{when } r = r' \\ N_i \langle r'/y \rangle & \text{when } r_i = r'_i \end{cases}
\end{array}
\qquad
\frac{A \text{ type}_{\text{Kan}} [\Psi, x] \quad M \in A \langle r/x \rangle [\Psi]}{\text{coe}_{x.A}^{r \rightsquigarrow r'} (M) \in A \langle r'/x \rangle [\Psi]} \\
\text{coe}_{x.A}^{r \rightsquigarrow r'} (M) \doteq M \in A \langle r/x \rangle [\Psi]$$

■ **Figure 2** Kan operations.

These Kan operations are variants of the uniform Kan conditions first proposed by Bezem et al. [12]. In unpublished work in 2014, Licata and Brunerie [22] and Coquand [18] considered uniform Kan operations in Cartesian cubical sets, but did not succeed in defining univalent type theories based on those operations. Our Kan operations introduce two important innovations. First, we allow open boxes with sides attached along *diagonals* $x = z$, in addition to faces; this is essential to construct univalent universes (Sections 5.6 and 6). Second, the validity condition requires that every box must contain at least one opposing pair of sides $x = 0$ and $x = 1$; this sharpens our canonicity results for higher inductive types (Section 5.2). We defer further comparison of Kan operations to Section 7.

5 Type formers

We proceed to construct a cubical type system with booleans and the circle (as a representative higher inductive type), and closed under dependent function and pair types, path types, exact equality types, and univalent universes. (Our preprint [7] also includes an empty type and natural numbers.) Each of these type formers is given meaning as a value-coherent Ψ -PER on values, and shown to validate the appropriate rules of inference. (We focus on closed-term rules, from which the open rules follow.) In this section we analyze each type former separately, excepting pretype and Kan universes, which we defer to Section 6.

5.1 Booleans

There are two boolean values at every dimension: $\llbracket \text{bool} \rrbracket_{\Psi} = \{(\text{true}, \text{true}), (\text{false}, \text{false})\}$. This context-indexed PER is clearly value-coherent, as the constructors are unaffected by dimension substitution. The canonicity property follows directly from this definition:

► **Theorem 15** (Canonicity). *If $M \in \text{bool} [\Psi]$ then $M \Downarrow V$ and $M \doteq V \in \text{bool} [\Psi]$, for $V = \text{true}$ or $V = \text{false}$.*

Proof. Then $\text{Tm}(\llbracket \text{bool} \rrbracket)(M, M)$, so $M \Downarrow V$ and $\llbracket \text{bool} \rrbracket(V, V)$. By Lemma 11, $M \doteq V \in \text{bool} [\Psi]$, and by the definition of $\llbracket \text{bool} \rrbracket$, $V = \text{true}$ or $V = \text{false}$. ◀

Consistency is similar: $\text{true} \doteq \text{false} \in \text{bool} [\Psi]$ implies $\llbracket \text{bool} \rrbracket(\text{true}, \text{false})$, which is impossible.

The rules in Fig. 3 all hold: true and false are elements, the elimination rule holds essentially by Theorem 15, and the computation rules hold by Lemma 9. The Kan operations of bool are identity functions, because every line in bool is degenerate.

$$\begin{array}{c}
 \overline{\text{bool type}_{\text{Kan}} [\Psi]} \qquad \overline{\text{true} \in \text{bool} [\Psi]} \qquad \overline{\text{false} \in \text{bool} [\Psi]} \\
 \hline
 \frac{b : \text{bool} \gg A \text{ type}_{\text{pre}} [\Psi] \quad M \in \text{bool} [\Psi] \quad T \in A[\text{true}/b] [\Psi] \quad F \in A[\text{false}/b] [\Psi]}{\text{if}(M; T, F) \in A[M/b] [\Psi]} \\
 \\
 \frac{T \in A [\Psi]}{\text{if}(\text{true}; T, F) \doteq T \in A [\Psi]} \qquad \frac{F \in A [\Psi]}{\text{if}(\text{false}; T, F) \doteq F \in A [\Psi]} \\
 \hline
 \overline{\mathbb{S}^1 \text{ type}_{\text{Kan}} [\Psi]} \qquad \overline{\text{base} \in \mathbb{S}^1 [\Psi]} \qquad \overline{\text{loop}_r \in \mathbb{S}^1 [\Psi]} \qquad \overline{\text{loop}_\varepsilon \doteq \text{base} \in \mathbb{S}^1 [\Psi]} \\
 \\
 \frac{c : \mathbb{S}^1 \gg A \text{ type}_{\text{Kan}} [\Psi] \quad M \in \mathbb{S}^1 [\Psi] \quad P \in A[\text{base}/c] [\Psi] \quad L \in A[\text{loop}_x/c] [\Psi, x] \quad (\forall \varepsilon) L\langle \varepsilon/x \rangle \doteq P \in A[\text{base}/c] [\Psi]}{\mathbb{S}^1\text{-elim}_{c.A}(M; P, x.L) \in A[M/c] [\Psi]} \\
 \\
 \frac{P \in B [\Psi]}{\mathbb{S}^1\text{-elim}_{c.A}(\text{base}; P, x.L) \doteq P \in B [\Psi]} \qquad \frac{L \in B [\Psi, x] \quad (\forall \varepsilon) L\langle \varepsilon/x \rangle \doteq P \in B\langle \varepsilon/x \rangle [\Psi]}{\mathbb{S}^1\text{-elim}_{c.A}(\text{loop}_r; P, x.L) \doteq L\langle r/x \rangle \in B\langle r/x \rangle [\Psi]}
 \end{array}$$

■ **Figure 3** Boolean and circle type.

5.2 Circle

It is tempting to define the circle as the least context-indexed PER generated by a base point and a loop: $\llbracket \mathbb{S}^1 \rrbracket_{\Psi}(\text{base}, \text{base})$ and $\llbracket \mathbb{S}^1 \rrbracket_{(\Psi, x)}(\text{loop}_x, \text{loop}_x)$. Unlike `bool`, \mathbb{S}^1 has non-degenerate lines, so we must explicitly add composites of open boxes to \mathbb{S}^1 if we want it to be Kan. We therefore equip \mathbb{S}^1 with the following *free* Kan structure (writing ξ_i to abbreviate $r_i = r'_i$):

$$\begin{array}{ll}
 \text{coe}_{x.\mathbb{S}^1}^{r \rightsquigarrow r'}(M) \mapsto_{\text{op}} M & \\
 \text{hcom}_{\mathbb{S}^1}^{r \rightsquigarrow r'}(M; \overline{\xi_i \hookrightarrow y.N_i}) \mapsto_{\text{op}} M & \text{if } r = r' \\
 \text{hcom}_{\mathbb{S}^1}^{r \rightsquigarrow r'}(M; \overline{\xi_i \hookrightarrow y.N_i}) \mapsto N_j\langle r'/y \rangle & \text{if } r \neq r', r_j = r'_j, r_i \neq r'_i \text{ for } i < j \\
 \text{hcom}_{\mathbb{S}^1}^{r \rightsquigarrow r'}(M; \overline{\xi_i \hookrightarrow y.N_i}) \text{ val} & \text{if } r \neq r', r_i \neq r'_i
 \end{array}$$

These operational semantics satisfy the equations in Fig. 2: when $r = r'$ in `hcom`, line (2) applies; when $r_i = r'_i$, line (3) applies; and for every `hcom`, one of lines (2–4) applies. Disequalities are needed in lines (3–4) to maintain determinacy. To account for value `hcom`s, we add a clause that $\llbracket \mathbb{S}^1 \rrbracket_{\Psi}(\text{hcom}_{\mathbb{S}^1}^{r \rightsquigarrow r'}(M; \overline{\xi_i \hookrightarrow y.N_i}), \text{hcom}_{\mathbb{S}^1}^{r \rightsquigarrow r'}(M'; \overline{\xi_i \hookrightarrow y.N'_i}))$ whenever these are values and satisfy the premises of the `hcom` rule in Fig. 2. Value-coherence of $\llbracket \mathbb{S}^1 \rrbracket$ follows from the operational semantics of `hcom` $_{\mathbb{S}^1}$ and the premises of the `hcom` typing rule. By limiting the Kan operations to valid context restrictions, we ensure that $\llbracket \mathbb{S}^1 \rrbracket_{\emptyset}$ contains no `hcom`s – there are no valid restrictions at dimension \emptyset in which $r_i \neq r'_i$ for all i .

The rules for the circle can be found in Fig. 3, including the eliminator mapping from \mathbb{S}^1 into any Kan type with a point P and line $x.L$ satisfying $L\langle 0/x \rangle \doteq L\langle 1/x \rangle \doteq P$. The eliminator sends `base` to P , `loopy` to $L\langle y/x \rangle$, and `hcom` $_{\mathbb{S}^1}$ to a Kan composition in the target type. (See our preprint [7] for the latter operational semantics step, which requires a derived notion of *heterogeneous* composition in which the type varies across the open box.) It is therefore essential that the target type is Kan.

5.3 Dependent function and pair types

When $A \text{ type}_{\text{pre}} [\Psi]$ and $a : A \gg B \text{ type}_{\text{pre}} [\Psi]$,

$$\begin{aligned} \llbracket (a:A) \rightarrow B \rrbracket_{\psi} &= \{(\lambda a.N, \lambda a.N') \mid a : A\psi \gg N \doteq N' \in B\psi [\Psi']\} \\ \llbracket (a:A) \times B \rrbracket_{\psi} &= \{(\langle M, N \rangle, \langle M', N' \rangle) \mid M \doteq M' \in A\psi [\Psi'] \wedge N \doteq N' \in B\psi[M/a] [\Psi']\} \end{aligned}$$

Rules for dependent function and dependent pair types are listed in Fig. 4, including judgmental η principles. The Kan operations for dependent function types are:

$$\begin{aligned} \text{hcom}_{(a:A) \rightarrow B}^{r \rightsquigarrow r'}(M; \overline{\xi_i \hookrightarrow y.N_i}) &\mapsto_{\text{Kan}} \lambda a. \text{hcom}_B^{r \rightsquigarrow r'}(\text{app}(M, a); \overline{\xi_i \hookrightarrow y.\text{app}(N_i, a)}) \\ \text{coe}_{x.(a:A) \rightarrow B}^{r \rightsquigarrow r'}(M) &\mapsto_{\text{Kan}} \lambda a. \text{coe}_{x.B[\text{coe}_{x.A}^{r' \rightsquigarrow r}(a)/a]}^{r \rightsquigarrow r'}(\text{app}(M, \text{coe}_{x.A}^{r' \rightsquigarrow r}(a))) \end{aligned}$$

If $A \text{ type}_{\text{Kan}} [\Psi]$ and $a : A \gg B \text{ type}_{\text{Kan}} [\Psi]$, then by the above steps and the introduction, elimination, and eta rules, $(a:A) \rightarrow B \text{ type}_{\text{Kan}} [\Psi]$ (and similarly [7], $(a:A) \times B \text{ type}_{\text{Kan}} [\Psi]$).

5.4 Path types

Whenever $A \text{ type}_{\text{pre}} [\Psi, x]$ and $P_{\varepsilon} \doteq P'_{\varepsilon} \in A\langle \varepsilon/x \rangle [\Psi]$ for $\varepsilon \in \{0, 1\}$, $\llbracket \text{Path}_{x.A}(P_0, P_1) \rrbracket_{\psi} = \{(\langle x \rangle M, \langle x \rangle M') \mid M \doteq M' \in A\psi [\Psi', x] \wedge \forall \varepsilon. (M\langle \varepsilon/x \rangle \doteq P_{\varepsilon}\psi \in A\psi\langle \varepsilon/x \rangle [\Psi'])\}$. That is, paths are abstracted lines with specified endpoints, and dimension abstraction ($\langle x \rangle M$) and application ($M@r$) pack and unpack them. Rules for path types are listed in Fig. 4; once again, Kan operations (see [7]) ensure that $\text{Path}_{x.A}(P_0, P_1) \text{ type}_{\text{Kan}} [\Psi]$ when $A \text{ type}_{\text{Kan}} [\Psi, x]$.

Notably, while homotopy type theory relies on the identity type to generate path structure, in this setting the path type merely internalizes a preexisting judgmental notion of paths. The homotopy-type-theoretic identity elimination principle is definable for $\text{Path}_{x.A}(M, N)$ when A is Kan, but as in Cohen et al. [17], its computation rule holds only up to a path.

5.5 Exact equality types

Whenever $A \text{ type}_{\text{pre}} [\Psi]$, $M \in A [\Psi]$, and $N \in A [\Psi]$, we have $\llbracket \text{Eq}_A(M, N) \rrbracket_{\psi} = \{(\star, \star) \mid M\psi \doteq N\psi \in A\psi [\Psi']\}$. That is, $\text{Eq}_A(M, N)$ is (uniquely) inhabited if and only if $M \doteq N \in A [\Psi]$, and therefore equality reflection holds. Rules for equality types are listed in Fig. 4.

Unlike the previous cases, $\text{Eq}_A(M, N)$ is not necessarily Kan when A is Kan, because coercion in $\text{Eq}_A(M, N)$ implies uniqueness of identity proofs in A . We allow $\text{Eq}_A(M, N) \text{ type}_{\text{Kan}} [\Psi]$ when A is *discrete Kan* [7], roughly, contains only degenerate paths (for example, $A = \text{bool}$).

5.6 Univalence

Voevodsky's univalence axiom [31] concerns a notion of *type equivalence* $\text{Equiv}(A, B)$:

$$\begin{aligned} \text{isContr}(C) &:= C \times ((c:C) \rightarrow (c':C) \rightarrow \text{Path}_{_C}(c, c')) \\ \text{Equiv}(A, B) &:= (f:A \rightarrow B) \times ((b:B) \rightarrow \text{isContr}((a:A) \times \text{Path}_{_B}(\text{app}(f, a), b))) \end{aligned}$$

Essentially, $\text{Equiv}(A, B)$ if there is a map $A \rightarrow B$ such that the (homotopy) preimage in A of any point in B is contractible (has exactly one point up to homotopy). In homotopy type theory, univalence states that $\text{idtoequiv} : \text{Id}_{\mathcal{U}}(A, B) \rightarrow \text{Equiv}(A, B)$ (definable in intensional type theory) is itself an equivalence. By a theorem of Licata [21], univalence in the present setting is equivalent to the existence of a map $\text{ua} : \text{Equiv}(A, B) \rightarrow \text{Path}_{_M_{\text{Kan}}}(A, B)$ and a homotopy $\text{ua}_{\beta}(E)$ between the functions underlying the equivalences E and $\text{idtoequiv}(\text{ua}(E))$.

$$\begin{array}{c}
 \frac{A \text{ type}_{\kappa} [\Psi] \quad a : A \gg B \text{ type}_{\kappa} [\Psi]}{(a:A) \rightarrow B \text{ type}_{\kappa} [\Psi]} \qquad \frac{a : A \gg M \in B [\Psi]}{\lambda a.M \in (a:A) \rightarrow B [\Psi]} \\
 \\
 \frac{M \in (a:A) \rightarrow B [\Psi] \quad N \in A [\Psi]}{\text{app}(M, N) \in B[N/a] [\Psi]} \qquad \frac{a : A \gg M \in B [\Psi] \quad N \in A [\Psi]}{\text{app}(\lambda a.M, N) \doteq M[N/a] \in B[N/a] [\Psi]} \\
 \\
 \frac{M \in (a:A) \rightarrow B [\Psi]}{M \doteq \lambda a.\text{app}(M, a) \in (a:A) \rightarrow B [\Psi]} \\
 \hline
 \\
 \frac{A \text{ type}_{\kappa} [\Psi] \quad a : A \gg B \text{ type}_{\kappa} [\Psi]}{(a:A) \times B \text{ type}_{\kappa} [\Psi]} \qquad \frac{M \in A [\Psi] \quad N \in B[M/a] [\Psi]}{\langle M, N \rangle \in (a:A) \times B [\Psi]} \\
 \\
 \frac{P \in (a:A) \times B [\Psi]}{\text{fst}(P) \in A [\Psi]} \qquad \frac{P \in (a:A) \times B [\Psi]}{\text{snd}(P) \in B[\text{fst}(P)/a] [\Psi]} \qquad \frac{M \in A [\Psi]}{\text{fst}(\langle M, N \rangle) \doteq M \in A [\Psi]} \\
 \\
 \frac{N \in B [\Psi]}{\text{snd}(\langle M, N \rangle) \doteq N \in B [\Psi]} \qquad \frac{P \in (a:A) \times B [\Psi]}{P \doteq \langle \text{fst}(P), \text{snd}(P) \rangle \in (a:A) \times B [\Psi]} \\
 \hline
 \\
 \frac{A \text{ type}_{\kappa} [\Psi, x] \quad (\forall \varepsilon) P_{\varepsilon} \in A\langle \varepsilon/x \rangle [\Psi]}{\text{Path}_{x.A}(P_0, P_1) \text{ type}_{\kappa} [\Psi]} \qquad \frac{M \in A [\Psi, x] \quad (\forall \varepsilon) M\langle \varepsilon/x \rangle \doteq P_{\varepsilon} \in A\langle \varepsilon/x \rangle [\Psi]}{\langle x \rangle M \in \text{Path}_{x.A}(P_0, P_1) [\Psi]} \\
 \\
 \frac{M \in \text{Path}_{x.A}(P_0, P_1) [\Psi]}{M @ r \in A\langle r/x \rangle [\Psi]} \qquad \frac{M \in \text{Path}_{x.A}(P_0, P_1) [\Psi]}{M @ \varepsilon \doteq P_{\varepsilon} \in A\langle \varepsilon/x \rangle [\Psi]} \\
 \\
 \frac{M \in A [\Psi, x]}{\langle x \rangle M @ r \doteq M\langle r/x \rangle \in A\langle r/x \rangle [\Psi]} \qquad \frac{M \in \text{Path}_{x.A}(P_0, P_1) [\Psi]}{M \doteq \langle x \rangle (M @ x) \in \text{Path}_{x.A}(P_0, P_1) [\Psi]} \\
 \hline
 \\
 \frac{A \text{ type}_{\text{pre}} [\Psi] \quad M \in A [\Psi] \quad N \in A [\Psi]}{\text{Eq}_A(M, N) \text{ type}_{\text{pre}} [\Psi]} \qquad \frac{M \doteq N \in A [\Psi]}{\star \in \text{Eq}_A(M, N) [\Psi]} \\
 \\
 \frac{E \in \text{Eq}_A(M, N) [\Psi]}{M \doteq N \in A [\Psi]} \qquad \frac{E \in \text{Eq}_A(M, N) [\Psi]}{E \doteq \star \in \text{Eq}_A(M, N) [\Psi]}
 \end{array}$$

■ **Figure 4** Dependent functions, dependent pairs, paths, and exact equalities.

$$\begin{array}{ccc}
\begin{array}{ccc}
M & & \\
\downarrow F & \dashrightarrow \text{Vin}_x(M, N) & \\
\text{app}(F, M) \doteq N\langle 0/x \rangle & \xrightarrow{N} & N\langle 1/x \rangle
\end{array} & \in & \begin{array}{ccc}
A & & \\
\downarrow F & \dashrightarrow \text{V}_x(A, B, \langle F, _ \rangle) & \\
B\langle 0/x \rangle & \xrightarrow{B} & B\langle 1/x \rangle
\end{array}
\end{array}$$

We achieve both conditions by defining a new type former “ \mathbf{V} ”, such that whenever $A \text{ type}_{\text{pre}} [\Psi, x \mid x = 0]$, $B \text{ type}_{\text{pre}} [\Psi, x]$, and $E \in \text{Equiv}(A, B) [\Psi, x \mid x = 0]$, $\text{V}_x(A, B, E)$ is a type with faces $A\langle 0/x \rangle$ and $B\langle 1/x \rangle$, whose elements are pairs of $N \in B [\Psi, x]$ and $M \in A\langle 0/x \rangle [\Psi]$ such that E sends M to exactly $N\langle 0/x \rangle$. (Bezem et al. [13] employ the same approach in their “G” types.) We then define:

$$\begin{aligned}
\text{idtoequiv} &:= \lambda p. \text{coe}_{x. \text{Equiv}(A, p @ x)}^{0 \rightsquigarrow 1} (\langle \lambda a. a, \text{idisequiv} \rangle) \\
\text{ua} &:= \lambda e. \langle x \rangle \text{V}_x(A, B, e) \\
\text{ua}_\beta &:= \lambda e. \lambda a. \langle x \rangle \text{coe}_{x. B}^{x \rightsquigarrow 1} (\text{app}(\text{fst}(e), a))
\end{aligned}$$

where idisequiv is a proof that the identity function is an equivalence, and ua_β relies on coercion across an equivalence: $\text{coe}_{x. \text{V}_x(A, B, E)}^{0 \rightsquigarrow r'}(M) \mapsto_{\text{app}} \text{Vin}_{r'}(M, \text{coe}_{x. B}^{0 \rightsquigarrow r'}(\text{app}(\text{fst}(E\langle 0/x \rangle), M)))$.

When implementing $\text{coe}_{x. \text{V}_x(A, B, E)}^{y \rightsquigarrow r'}(M)$, we make essential use of an open box with a diagonal $y = r'$ side, to ensure coercion $y \rightsquigarrow y$ is the identity. (See our preprint [7] for this and the other Kan operations.) We have formalized the full proof of univalence for our system in REDPRL (see <https://git.io/vFjUQ>).

6 Universes

Finally, we define two cumulative hierarchies of universes, $\mathcal{U}_j^{\text{pre}}$ and $\mathcal{U}_j^{\text{Kan}}$, classifying pretypes and Kan types respectively, each closed under the appropriate type formers, and satisfying:

$$\frac{}{\mathcal{U}_j^\kappa \text{ type}_{\text{Kan}} [\Psi]} \quad \frac{A \in \mathcal{U}_j^\kappa [\Psi]}{A \text{ type}_\kappa [\Psi]} \quad \frac{A \in \mathcal{U}_j^\kappa [\Psi]}{A \in \mathcal{U}_{j+1}^\kappa [\Psi]} \quad \frac{A \in \mathcal{U}_j^{\text{Kan}} [\Psi]}{A \in \mathcal{U}_j^{\text{pre}} [\Psi]}$$

In order for our type theory to be a suitable setting for synthetic homotopy theory, it is essential that $\mathcal{U}_j^{\text{Kan}}$ is Kan; this is needed, for example, to define maps $\mathbb{S}^1 \rightarrow \mathcal{U}_j^{\text{Kan}}$ used in the calculation of the fundamental group of the circle [29]. As with \mathbb{S}^1 , universes are not automatically Kan, so we equip both with free Kan structure analogous to $\text{hcom}_{\mathbb{S}^1}$.

Because elements of $\mathcal{U}_j^{\text{pre}}$ are pretypes, we must ensure $\text{hcom}_{\mathcal{U}_j^{\text{pre}}}^{r \rightsquigarrow r'}(A; \xi_i \hookrightarrow y.B_i) \text{ type}_{\text{pre}} [\Psi]$ for pretypes A, \overline{B}_i satisfying the appropriate equations. We define these types to be empty. Similarly, we require $\text{hcom}_{\mathcal{U}_j^{\text{Kan}}}^{r \rightsquigarrow r'}(A; \xi_i \hookrightarrow y.B_i) \text{ type}_{\text{Kan}} [\Psi]$ for Kan types A, \overline{B}_i satisfying the appropriate equations. In order to equip $\text{hcom}_{\mathcal{U}_j^{\text{Kan}}}$ with Kan operations, we define its elements to be open boxes consisting of an element $M \in A [\Psi]$, and a family of elements $N_i \in B_i\langle r'/y \rangle [\Psi \mid \xi_i]$ such that $\text{coe}_{y. B_i}^{r' \rightsquigarrow r}(N_i) \doteq M \in A [\Psi \mid \xi_i]$. The diagram below illustrates an element of $H := \text{hcom}_{\mathcal{U}_j^{\text{Kan}}}^{0 \rightsquigarrow 1}(A; x = 0 \hookrightarrow y.B_0, x = 1 \hookrightarrow y.B_1)$.

$$\begin{array}{ccc}
\begin{array}{ccc}
\text{coe}_{y. B_0}^{1 \rightsquigarrow 0}(N_0) & \xrightarrow{M} & \text{coe}_{y. B_1}^{1 \rightsquigarrow 0}(N_1) \\
\downarrow \text{box}^{0 \rightsquigarrow 1}(M; N_0, N_1) & & \downarrow \\
N_0 & \dashrightarrow & N_1
\end{array} & \in & \begin{array}{ccc}
\cdot & \xrightarrow{A} & \cdot \\
B_0 \downarrow & H & \downarrow B_1 \\
B_0\langle 1/y \rangle & \dashrightarrow & B_1\langle 1/y \rangle
\end{array}
\end{array}$$

When $r = r'$, $H \doteq A$ and the $\text{box} \doteq M$. When ξ_i holds, $H \doteq B_i \langle r'/y \rangle$ and the $\text{box} \doteq N_i$. These agree when both $r = r'$ and ξ_i hold: $A \doteq B_i \langle r/y \rangle = B_i \langle r'/y \rangle$ and $M \doteq \text{coe}_{y.B_i}^{r' \rightsquigarrow r}(N_i) \doteq N_i$.

For the complete definition of $\text{hcom}_{\mathcal{U}_j^{\text{Kan}}}$ and its Kan operations, see our preprint [7]. Coercion requires heterogeneous compositions that may not be valid in the sense of Definition 12, but which are nevertheless definable in our setting. (Such compositions are closely related to the $\forall i.\varphi$ operation of Cohen et al. [17].) Finally, to ensure these Kan operations agree with those of A when $r = r'$, we once again make essential use of open boxes with diagonal sides.

Intuitively, each universe $\llbracket \mathcal{U}_j^\kappa \rrbracket$ is defined as the least context-indexed PER closed under all type formers yielding κ -types, that are present in a type theory with j universes. Of course, typehood and membership are mutually defined ($\text{Eq}_A(M, N)$ $\text{type}_{\text{pre}}[\Psi]$ when $M, N \in A[\Psi]$), so the values of each universe depend on both the names *and* semantics of types.

Following Allen [1], we make this construction precise by introducing *candidate cubical type systems*, relations $\tau(\Psi, A_0, B_0, \varphi)$ as in Definition 4 without any conditions of functionality, symmetry, and so forth. Candidate cubical type systems form a complete lattice when ordered by inclusion, so we define each universe as the least fixed point of a monotone operator (guaranteed to exist by the Knaster–Tarski fixed point theorem).

For each κ , we define an operator $F^\kappa(\tau^u, \tau^{\text{pre}}, \tau^{\text{Kan}})$ whose arguments are candidate cubical type systems defining (1) all smaller universes, (2) pretype formers, and (3) Kan type formers, following the meanings given in Section 5. These operators are monotone because $\text{Tm}(-)$ is monotone, and hence the judgments defined in Section 3 are monotone in τ .

Then construct the simultaneous least fixed points $\tau_i^\kappa = F^\kappa(\tau_i^u, \tau_i^{\text{pre}}, \tau_i^{\text{Kan}})$ for each $i \geq 0$, where τ_i^u defines each $\llbracket \mathcal{U}_j^\kappa \rrbracket$ (for $j < i$) as $\tau_i^u(\Psi, \mathcal{U}_j^\kappa, \mathcal{U}_j^\kappa, \{(A_0, B_0) \mid \tau_j^\kappa(\Psi, A_0, B_0, _)\})$, that is, the typehood relation of τ_j^κ . We establish by induction that each τ_i^κ is in fact a cubical type system in the sense of Definition 4, and each is closed under the appropriate type formers. We take the “outermost” cubical type system τ_ω^{pre} (containing universes for all j) as our model, validating every rule presented in this paper. This construction requires no classical reasoning, and in fact Anand and Rahli [4] carry out Allen’s original Nuprl semantics inside the Coq proof assistant using inductive types rather than fixed points.

7 Conclusion and Related Work

We have constructed a two-level type theory with fibrant, univalent universes closed under dependent function, dependent pair, and path types. The non-fibrant (pretype) level includes these type formers as well as exact (strict) equality types with equality reflection. Following the tradition of the Nuprl computational type theory [2] and Martin-Löf’s meaning explanations, our types are relations over untyped programs equipped with an operational semantics, and thereby satisfy canonicity (Theorem 15) by construction. Full details and proofs are available in our associated preprint [7]. An early version of our cubical PER semantics appeared in Angiuli et al. [6], but for a type theory including neither univalence, nor universes, nor exact equality, and equipped with a variant of our Kan operations restricted to open boxes with sides $r_i = 0, r_i = 1$ (and in particular, without $x = z$ sides critical for univalent universes).

We are currently implementing the REDPRL [28] proof assistant based on this type theory. REDPRL implements a proof refinement sequent calculus in the style of Nuprl, rather than the natural deduction rules presented in this paper; we view it as the extension of core Nuprl to a higher-dimensional notion of program.

Cavallo and Harper [16] define a schema of higher inductive types constructible in the semantic framework we describe. Their *fiber family* type validates the rules of the homotopy-type-theoretic identity type (strictly, unlike path types). Our type theory, extended with fiber families, constitutes a fully computational model of univalent intensional type theory.

7.1 Two-level type theories

Voevodsky’s HTS [33] extends homotopy type theory with exact equality types satisfying equality reflection. Our semantics validate the rules of HTS, excepting resizing rules. More recently, Altenkirch et al. [3] have proposed a two-level type theory with two intensional identity types: one to internalize paths, and the other satisfying uniqueness of identity proofs and function extensionality, but not equality reflection. Both theories consider all strict equality types non-fibrant, and neither theory satisfies canonicity, because univalence (and in the latter, uniqueness of identity proofs and function extensionality) are added as axioms that do not compute.

Our contributions to two-level type theory are twofold: (1) we define the first two-level type theory satisfying canonicity, and (2) by introducing the notion of discrete Kan types (see our preprint [7]), we obtain a type theory in which some exact equality types are fibrant.

7.2 Cubical type theories

Our use of cubical structure and uniform Kan conditions traces back to the Bezem et al. [12] cubical set model of type theory, which has only face and degeneracy maps. The cubical type theory of Cohen et al. [17] uses a De Morgan algebra of cubes containing not only face, diagonal, and degeneracy maps, but also connection and reversal maps.

From a proof-theoretic perspective, our semantics can be seen as *cubical logical relations* suitable for proving canonicity (and consistency) for a set of inference rules. In fact, Huber’s canonicity argument [19] for Cohen et al. [17] resembles our PER semantics in various ways, most notably his “expansion lemma,” which is closely related to Lemma 10.

The fibrant fragment of our system constitutes the second univalent type theory with canonicity – after the cubical type theory of Cohen et al. [17] – and the first to employ Cartesian cubical structure. Licata and Brunerie [22] and Coquand [18] previously considered Cartesian cubes, but did not succeed in defining univalent universes. However, neither considered Kan operations with diagonal sides $x = z$, which figure prominently in our constructions of both univalence and fibrant universes. Diagonal sides also permit us to define connections in Kan types, although we remain unable to define an involutive reversal operation, as in Cohen et al. [17].

In ongoing work with Brunerie, Coquand, and Licata [5], we are investigating proof-theoretic and category-theoretic aspects of “diagonal” Kan composition. That project includes an Agda formalization of the Kan operations of various type formers, including a variant of the “Glue” types employed by Cohen et al. [17] to obtain both univalence and fibrancy of the universe. Here we decompose Glue types into \mathbf{V} and $\mathbf{hcom}_{\mathcal{U}_j^{\text{Kan}}}$, simplifying \mathbf{ua}_β .

Unlike prior Kan conditions, we restrict to open boxes containing a pair of sides $x = 0, x = 1$ (Definition 12), in order to trivialize all Kan compositions at dimension zero. Thus we obtain a stronger canonicity result for the circle than Cohen et al. [17]: if $M \in \mathbb{S}^1 [\emptyset]$ then $M \Downarrow \mathbf{base}$. We believe this property to be valuable for programming applications of cubical type theory, by allowing higher inductive types to function as observables at dimension zero. The tradeoff is that we must develop additional machinery to define coercion in $\mathbf{hcom}_{\mathcal{U}_j^{\text{Kan}}}$, essentially because the $\forall i.\varphi$ operation of Cohen et al. [17] does not preserve box validity.

References


- 1 Stuart F. Allen. A Non-type-theoretic Definition of Martin-Löf's Types. In D. Gries, editor, *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science*, pages 215–224, Ithaca, NY, 1987. IEEE Computer Society Press.
- 2 Stuart F Allen, Mark Bickford, Robert L Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran. Innovations in computational type theory using Nuprl. *Journal of Applied Logic*, 4(4):428–469, 2006.
- 3 Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. Extending homotopy type theory with strict equality. In *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, pages 21:1–21:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2016.21.
- 4 Abhishek Anand and Vincent Rahli. Towards a formally verified proof assistant. In *Interactive Theorem Proving*, pages 27–44, Cham, 2014. Springer International Publishing.
- 5 Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. Cartesian cubical type theory. Preprint, December 2017. URL: <https://github.com/dlicata335/cart-cube>.
- 6 Carlo Angiuli, Robert Harper, and Todd Wilson. Computational higher-dimensional type theory. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 680–693, New York, NY, USA, 2017. ACM. doi:10.1145/3009837.3009861.
- 7 Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. Computational higher type theory III: Univalent universes and exact equality, December 2017. arXiv:1712.01800.
- 8 Danil Annenkov, Paolo Capriotti, and Nicolai Kraus. Two-level type theory and applications, 2017. arXiv:1705.03307.
- 9 Steve Awodey. A cubical model of homotopy type theory, June 2016. URL: <https://www.andrew.cmu.edu/user/awodey/preprints/stockholm.pdf>.
- 10 Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, 2009. doi:10.1017/S0305004108001783.
- 11 Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Michael Shulman, Matthieu Sozeau, and Bas Spitters. The HoTT library: A formalization of homotopy type theory in Coq. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 164–172, New York, NY, USA, 2017. ACM. doi:10.1145/3018610.3018615.
- 12 Marc Bezem, Thierry Coquand, and Simon Huber. A model of type theory in cubical sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26, pages 107–128, Toulouse, France, 2014. Dagstuhl Publishing.
- 13 Marc Bezem, Thierry Coquand, and Simon Huber. The univalence axiom in cubical sets, October 2017. arXiv:1710.10941.
- 14 Guillaume Brunerie, Kuen-Bang Hou (Favonia), Evan Cavallo, Eric Finster, Jesper Cockx, Christian Sattler, Chris Jeris, Michael Shulman, et al. Homotopy type theory in Agda, 2018. URL: <https://github.com/HoTT/HoTT-Agda>.
- 15 Ulrik Buchholtz and Edward Morehouse. Varieties of cubical sets. In *Relational and Algebraic Methods in Computer Science: 16th International Conference, RAMiCS 2017, Lyon, France, May 15-18, 2017, Proceedings*, pages 77–92. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-57418-9_5.
- 16 Evan Cavallo and Robert Harper. Computational higher type theory IV: Inductive types, January 2018. arXiv:1801.01568.
- 17 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69, pages 5:1–5:34, Dagstuhl,

- Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TYPES.2015.5.
- 18 Thierry Coquand. Variations on cubical sets, 2014. URL: <http://www.cse.chalmers.se/~coquand/diag.pdf>.
 - 19 Simon Huber. *Cubical Interpretations of Type Theory*. PhD thesis, University of Gothenburg, 2016.
 - 20 Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of univalent foundations (after Voevodsky), June 2016. arXiv:1211.2851.
 - 21 Daniel R. Licata. Weak univalence with “beta” implies full univalence. Email to the Homotopy Type Theory mailing list, 2016. URL: <https://groups.google.com/forum/#!topic/homotopytypetheory/j2KBIVDw53s>.
 - 22 Daniel R. Licata and Guillaume Brunerie. A cubical type theory, November 2014. Talk at Oxford Homotopy Type Theory Workshop. URL: <http://dlicata.web.wesleyan.edu/pubs/lb14cubical/lb14cubes-oxford.pdf>.
 - 23 Peter LeFanu Lumsdaine and Mike Shulman. Semantics of higher inductive types, May 2017. arXiv:1705.07088.
 - 24 P. Martin-Löf. Constructive mathematics and computer programming. *Philosophical Transactions of the Royal Society of London Series A*, 312:501–518, 1984. doi:10.1098/rsta.1984.0073.
 - 25 Per Martin-Löf. *Intuitionistic type theory*. Bibliopolis, Naples, Italy, 1984.
 - 26 A. M. Pitts. Nominal presentation of cubical sets models of type theory. In *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, pages 202–220, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TYPES.2014.202.
 - 27 W. W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967. doi:10.2307/2271658.
 - 28 The RedPRL Development Team. RedPRL – the People’s Refinement Logic, 2018. URL: <http://www.redprl.org/>.
 - 29 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. <http://homotopytypetheory.org/book>, Institute for Advanced Study, 2013.
 - 30 Floris van Doorn, Jakob von Raumer, and Ulrik Buchholtz. Homotopy type theory in Lean. In *Interactive Theorem Proving*, pages 479–495, Cham, 2017. Springer. doi:10.1007/978-3-319-66107-0_30.
 - 31 Vladimir Voevodsky. The equivalence axiom and univalent models of type theory, 2010. Notes from a talk at Carnegie Mellon University. URL: http://www.math.ias.edu/vladimir/files/CMU_talk.pdf.
 - 32 Vladimir Voevodsky. Univalent foundations project. Modified version of an NSF grant application, October 2010. URL: http://www.math.ias.edu/vladimir/files/univalent_foundations_project.pdf.
 - 33 Vladimir Voevodsky. A type system with two kinds of identity types. Slides available at https://uf-ias-2012.wikispaces.com/file/view/HTS_slides.pdf/410105196/HTS_slides.pdf, 2013. URL: <https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/HTS.pdf>.

Definable Inapproximability: New Challenges for Duplicator


Albert Atserias¹

Departament de Ciències de la Computació, Universitat Politècnica de Catalunya,
Barcelona, Catalonia, Spain
atserias@cs.upc.edu

 <https://orcid.org/0000-0002-3732-1989>

Anuj Dawar²

Department of Computer Science and Technology, University of Cambridge, UK
anuj.dawar@cl.cam.ac.uk

 <https://orcid.org/0000-0003-4014-8248>

Abstract

We consider the hardness of approximation of optimization problems from the point of view of definability. For many NP-hard optimization problems it is known that, unless $P = NP$, no polynomial-time algorithm can give an approximate solution guaranteed to be within a fixed constant factor of the optimum. We show, in several such instances and without any complexity theoretic assumption, that no algorithm that is expressible in fixed-point logic with counting (FPC) can compute an approximate solution. Since important algorithmic techniques for approximation algorithms (such as linear or semidefinite programming) are expressible in FPC, this yields lower bounds on what can be achieved by such methods. The results are established by showing lower bounds on the number of variables required in first-order logic with counting to separate instances with a high optimum from those with a low optimum for fixed-size instances.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic, Theory of computation → Finite Model Theory

Keywords and phrases Descriptive Complexity, Hardness of Approximation, MAX SAT, Vertex Cover, Fixed-point logic with counting

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.7

Related Version A full version of this paper is available at [8], <https://arxiv.org/abs/1806.11307>.

Acknowledgements The research reported here was initiated at the Simons Institute for the Theory of Computing during the programme on Logical Structures in Computation in autumn 2016.

1 Introduction

Twenty years ago, the PCP theorem [4] transformed the landscape of complexity theory. It showed that if $P \neq NP$ then not only is it impossible to efficiently solve NP-hard problems exactly but for some of them it is also impossible to approximate the solution to within a

¹ Partially funded by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement ERC-2014-CoG 648276 (AUTAR) and MICCIN grant TIN2016-76573-C2-1P (TASSAT3).

² Supported in part by a Fellowship of the Alan Turing Institute.



constant factor. Consider for instance the problem MAX 3SAT. Here we are given a Boolean formula in 3CNF and we are asked to determine m^* , the maximum number of clauses that can be simultaneously satisfied by an assignment of Boolean values to its variables. It is a consequence of the PCP theorem that there is a constant $c < 1$ such that, assuming $P \neq NP$, no polynomial-time algorithm can be guaranteed to produce an assignment that satisfies at least cm^* clauses, or indeed determine the value of m^* up to a factor of c . The proof of the PCP theorem introduced sophisticated new techniques into complexity theory such as the *probabilistically checkable proofs* that gave the theorem its name. Over the years, stronger results were proved, improving the constant c and, by reductions, proving inapproximability results for a host of other NP-hard problems.

A structural theory of hardness of approximation was introduced by Papadimitriou and Yannakakis [23] who defined the class MAX SNP of approximation problems, with a definition rooted in descriptive complexity theory. They showed that for every problem in this class, there is a constant d such that a polynomial-time algorithm can find approximate solutions within a factor d of the optimum. At the same time, for all problems that are MAX SNP-hard, under approximation-preserving reductions defined by [23], there is a constant c such that no polynomial-time algorithm can approximate solutions within a factor c . This makes it a challenge, for each MAX SNP-complete problem, to determine the exact approximation ratio that is achievable by an efficient algorithm. In some cases, this has been pinned down exactly. For instance, for MAX 3SAT we know that there is a polynomial-time algorithm that will produce an assignment satisfying $7/8$ of the clauses in any formula but, unless $P = NP$, there is no polynomial-time algorithm that is guaranteed to produce a solution within $7/8 + \epsilon$ of the optimal, for any $\epsilon > 0$ [16]. Another interesting case is MAX 3XOR, where we are given a formula which is the conjunction of clauses, each of which is the XOR of three literals. Here, satisfiability is decidable in polynomial time as the problem is essentially that of solving a system of linear equations over the two-element field. However, determining, for an unsatisfiable system, how many of its clauses can be simultaneously satisfied is MAX SNP-hard, and the exact approximation ratio that is achievable efficiently is known: unless $P = NP$, no polynomial-time algorithm can achieve an approximation ratio bounded above $1/2$ [16].

To give a problem of another flavour, consider *minimum vertex cover*, the problem of finding, in a graph G , a minimum set S of vertices such that every edge is incident on a vertex in S . Let $vc(G)$ denote the size of a minimum size vertex cover in G . There are algorithms that are guaranteed to find a vertex cover no larger than $2vc(G)$ (this being a minimization problem, the approximation ratio is expressed as a number $c \geq 1$). It has been proved, by means of rather sophisticated reductions starting at the PCP theorem, that, unless $P = NP$, no polynomial-time algorithm can achieve a ratio better than 1.36 [14]. Very recent results announced in [20] improve this lower bound to $\sqrt{2}$. It is conjectured that indeed no such algorithm could achieve a ratio of $2 - \epsilon$ for arbitrarily small $\epsilon > 0$ but, as of our current knowledge, the right threshold constant could be somewhere between $\sqrt{2}$ and 2 .

We approach these questions on the hardness of approximability from the point of view of definability. Our aim is to show that the tools of descriptive complexity can be brought to bear in showing lower bounds on the definability of approximations and that these definability lower bounds have consequences on understanding commonly used techniques in approximation algorithms.

A reference logic in descriptive complexity is fixed-point logic with counting, FPC. The class of problems definable in this logic form a proper subclass of the complexity class P. However, FPC is very expressive and many natural problems in P are expressible in this logic.

For instance, any polynomial-time decidable problem on a proper-minor closed class of graphs is expressible in FPC [15]. Also, problems that can be formulated as linear programming or semidefinite programming problems are in FPC [2, 9, 13]. At the same time, for many problems we are able to prove categorically, i.e., without complexity theoretic assumptions, that they are not definable in FPC. Among these are NP-complete problems like 3SAT, graph 3-colourability and Hamiltonicity (see [11]). We can also prove that certain problems in P are not in FPC, such as 3XOR.

A particularly interesting class of problems are the optimization problems known as MAX CSP or constraint maximization problems, where we are given a collection of constraints and the problem is to find the maximum number of constraints that can be simultaneously satisfied. When it comes to finding exact solutions, definability in FPC turns out to be an excellent guide to the tractability of such problems. It is known that each such problem is either in P *and* definable in FPC or it is NP-complete and provably *not* definable in FPC [12]. We would like to extend such results also to the *approximability* of such problems. This paper develops the methodology for doing so.

For MAX 3SAT, we prove, without any complexity theoretic assumption, that no algorithm expressible in FPC can achieve an approximation ratio of $7/8 + \epsilon$. The question seems ill-posed at first sight as FPC is a formalism for defining problems rather than expressing algorithms. We return to the precise formulation shortly, but first note that there is a sense in which FPC can express, say the ellipsoid method for solving linear programs [2]. This is the basis for showing that many commonly used algorithmic techniques for approximation problems, such as semidefinite programming relaxations, are also expressible in FPC. Thus, on the one hand, reductions from MAX SNP-hard problems show inapproximability by *any* polynomial-time algorithm, assuming $P \neq NP$. On the other hand, our results show, without the assumption, inapproximability by the most commonly used polynomial-time methods.

Undefinability of a class of structures \mathcal{C} in FPC is typically established by showing that structures in \mathcal{C} cannot be distinguished from structures not in \mathcal{C} in C^k – first-order logic with counting and just k variables – for any fixed k . In the terminology of [13], \mathcal{C} has unbounded *counting width*. On the other hand, hardness of approximation for a maximization problem is typically established by showing that every class that includes all instances with an optimum m^* and excludes all instances with an optimum less than cm^* , is NP-hard. Our method combines these two. We aim to show that any class separating instances with an optimum m^* from instances with an optimum less than cm^* has unbounded counting width. In general, we not only show that counting width is unbounded, but establish stronger bounds on how it grows with the size of instances, as such bounds are directly tied to lower bounds on semidefinite programming hierarchies [13]. This methodology poses new challenges for Spoiler-Duplicator games in finite model theory. Such games are typically played on pairs of structures that are minimally different. In the new setting, we need to show Duplicator winning strategies in games on pairs of structures that differ substantially, on some numeric parameters.

The PCP theorem is the *fons et origo* of results on hardness of approximation. It established the first provably NP-hard constant gap between the fully satisfiable instances of MAX 3SAT, i.e., those in which all clauses can be satisfied, and the less satisfiable ones, those where no more than $1 - \epsilon_0$ can be satisfied, for some explicit $\epsilon_0 > 0$. The gap between 1 and $1 - \epsilon_0$ was then amplified and also transferred to other problems by means of reductions. For us, the starting point is the problem MAX 3XOR. We are able to establish a definability gap between the satisfiable instances of this and instances in which little more than $3/4$ of the clauses can be satisfied. The methods for establishing this *initial gap* are

very different from that for the PCP theorem. We construct a k -locally satisfiable instance of MAX 3XOR which, by a random construction is at the same time highly unsatisfiable. We can then combine this with a construction adapted from [6] to obtain a gap that defeats any fixed counting width. With such an initial gap in hand, we can then amplify the gap and transfer it to other problems by means of reductions, just as in classical inapproximability. Our reductions have to preserve FPC definability and we mostly rely on first-order definable reductions. Indeed, many of the reductions used in the classical theory of approximability turn out to be first-order reductions but this requires close examination and proof.

By expressing the *long-code reductions* from [16] in first-order logic and composing them with our initial gap, we show optimal hardness for MAX 3SAT and MAX 3XOR. For the first, we show that FPC cannot achieve an approximation ratio of $7/8 + \epsilon$, even on satisfiable instances, and for the second it cannot achieve an approximation ratio of $1/2 + \epsilon$. These match known algorithmic lower bounds and are provably tight. For the vertex cover problem, direct reductions from these show that FPC cannot give an approximation better than $7/6$. This can be improved, using the reduction of [14] to 1.36 and the details of this may be found in the full version of this paper [8]. It is possible that this could be improved to $\sqrt{2}$ using the recent breakthrough of [20] but we leave this to future work.

2 Preliminaries

We use \mathbb{F}_2 to denote the 2-element field. For any positive integer n , let $[n] := \{1, \dots, n\}$.

Logics and games. We assume familiarity with first-order logic FO. All our vocabularies are finite and relational, and all structures are finite. For a structure \mathbb{A} , we write A to denote its universe. We refer to fixed-point logic with counting FPC but the definition is not required for the technical development in this paper. Here, it suffices to consider the bounded variable fragments of first-order logic.

For a fixed positive integer k , we write L^k to denote the fragment of first-order logic in which every formula has at most k variables, free or bound. We also write $\exists L^{k,+}$ for the *existential positive* fragment of L^k . This consists of those formulas of L^k formed using only the positive Boolean connectives \wedge and \vee , and existential quantification. FOC is the extension of first-order logic with *counting quantifiers*. For each natural number i , we have a quantifier \exists^i where $\mathbb{A} \models \exists^i x \phi$ if, and only if, there are at least i distinct elements $a \in A$ such that $\mathbb{A} \models \phi[a/x]$. While the extension of first-order logic with counting quantifiers is no more expressive than FO itself, the presence of these quantifiers does affect the number of variables that are necessary to express a query. Let C^k denote the k -variable fragment of FOC in which no more than k variables appear, free or bound.

For two structures \mathbb{A} and \mathbb{B} , we write $\mathbb{A} \equiv_{C^k} \mathbb{B}$ to denote that they are not distinguished by any sentence of C^k . All that we need to know about FPC is that for every formula ϕ of FPC there is a k such that if $\mathbb{A} \equiv_{C^k} \mathbb{B}$ then $\mathbb{A} \models \phi$ if, and only if, $\mathbb{B} \models \phi$. We also write $\mathbb{A} \Rightarrow_k \mathbb{B}$ to denote that every sentence of $\exists L^{k,+}$ that is true in \mathbb{A} is also true in \mathbb{B} . While \equiv_{C^k} is an equivalence relation, \Rightarrow_k is reflexive and transitive but not symmetric. These relations have well established characterizations in terms of two-player pebble games. The relation \Rightarrow_k is characterized by the *existential k -pebble game* [21] and \equiv_{C^k} by the *k -pebble bijective game* [17]. Rather than review the definitions here, we refer the reader to the sources.

For undirected graphs, the relation \equiv_{C^2} has a simple combinatorial characterization in terms of *vertex refinement* (see [19]). For any graph G , there is a coarsest partition C_1, \dots, C_m of the vertices of G such that for each $1 \leq i, j \leq m$ there exists δ_{ij} such that each

$v \in C_i$ has exactly δ_{ij} neighbours in C_j . Let H be another graph and $D_1, \dots, D_{m'}$ be the corresponding partition of H with constants γ_{ij} . Then $G \equiv_{C^2} H$ if, and only if, $m = m'$ and there is a permutation $h \in \text{Sym}_m$ such that $|C_i| = |D_{h(i)}|$ and $\delta_{ij} = \gamma_{h(i)h(j)}$ for all i and j .

Let \mathcal{C} be a class of structures and for any $n \in \mathbb{N}$, let \mathcal{C}_n denote the structures in \mathcal{C} with at most n elements. The *counting width* of \mathcal{C} [13] is the function $k : \mathbb{N} \rightarrow \mathbb{N}$ where $k(n)$ is the smallest value such that for any $\mathbb{A} \in \mathcal{C}_n$ and any $\mathbb{B} \notin \mathcal{C}$, we have $\mathbb{A} \not\equiv_{C^{k(n)}} \mathbb{B}$. Note that $k(n) \leq n$. Because $\mathbb{A} \not\equiv_{C^1} \mathbb{B}$ whenever \mathbb{A} and \mathbb{B} have different numbers of elements, $k(n)$ is also the smallest value such that \mathcal{C}_n is a union of $\equiv_{C^{k(n)}}$ -classes. In particular, it follows that the counting width of \mathcal{C} is the same as that of its complement. For $k : \mathbb{N} \rightarrow \mathbb{N}$, we say that two disjoint classes \mathcal{C} and \mathcal{D} are C^k -separable if whenever $\mathbb{A} \in \mathcal{C}_n$ and $\mathbb{B} \in \mathcal{D}_n$, then we have $\mathbb{A} \not\equiv_{C^{k(n)}} \mathbb{B}$. Equivalently \mathcal{C} and \mathcal{D} are C^k -separable if there is a class \mathcal{E} of counting width at most k such that $\mathcal{C} \subseteq \mathcal{E}$ and $\mathcal{D} \subseteq \bar{\mathcal{E}}$.

Interpretations. Consider two signatures σ and τ . A d -ary FO-interpretation of τ in σ is a sequence of first-order formulas in vocabulary σ consisting of: (i) a formula $\delta(\bar{x})$; (ii) a formula $\varepsilon(\bar{x}, \bar{y})$; (iii) for each relation symbol $R \in \tau$ of arity k , a formula $\phi_R(\bar{x}_1, \dots, \bar{x}_k)$; and (iv) for each constant symbol $c \in \tau$, a formula $\gamma_c(\bar{x})$, where each \bar{x}, \bar{y} or \bar{x}_i is a d -tuple of variables. We call d the *dimension* of the interpretation. If $d = 1$, we say that the interpretation is *linear*. We say that an interpretation Θ associates a τ -structure \mathbb{B} to a σ -structure \mathbb{A} if there is a map h from $\{\bar{a} \in A^d \mid \mathbb{A} \models \delta[\bar{a}]\}$ to the universe B of \mathbb{B} such that: (i) h is surjective onto B ; (ii) $h(\bar{a}_1) = h(\bar{a}_2)$ if, and only if, $\mathbb{A} \models \varepsilon[\bar{a}_1, \bar{a}_2]$; (iii) $R^{\mathbb{B}}(h(\bar{a}_1), \dots, h(\bar{a}_k))$ if, and only if, $\mathbb{A} \models \phi_R[\bar{a}_1, \dots, \bar{a}_k]$; and (iv) $h(\bar{a}) = c^{\mathbb{B}}$ if, and only if, $\mathbb{A} \models \gamma_c[\bar{a}]$. Note that an interpretation Θ associates a τ -structure with \mathbb{A} only if ε defines an equivalence relation on A^d that is a congruence with respect to the relations defined by the formulae ϕ_R and γ_c . In such cases, however, \mathbb{B} is uniquely defined up to isomorphism and we write $\Theta(\mathbb{A}) = \mathbb{B}$. It is also worth noting that the size of \mathbb{B} is at most n^d , if \mathbb{A} is of size n . But, it may in fact be smaller. We call an interpretation p -bounded, for a polynomial p , if $|\mathbb{B}| \leq p(|\mathbb{A}|)$, and say the interpretation is *linearly bounded* if p is linear. Every linear interpretation is linearly bounded, but the converse is not necessarily the case.

For a class of structures \mathcal{C} and an interpretation Θ , we write $\Theta(\mathcal{C})$ to denote the class $\{\Theta(\mathbb{A}) \mid \mathbb{A} \in \mathcal{C}\}$. We mainly use interpretations to define reductions between classes of structures. These allow us to transfer bounds on separability, by the following lemma, which is established by simply composing formulas. The details may be found in Appendix A.

► **Lemma 2.1.** *Let Θ be a p -bounded interpretation of dimension d and let t be the maximum number of variables appearing in any formula of Θ . If \mathcal{C} and \mathcal{D} are two disjoint classes of structures such that $\Theta(\mathcal{C})$ and $\Theta(\mathcal{D})$ are C^k -separable, then \mathcal{C} and \mathcal{D} are $C^{dk(p(n))+t}$ -separable.*

When we wish to define a reduction from a class \mathcal{C} by a first-order interpretation, it suffices to give an interpretation Θ for all structures in \mathcal{C} with at least two elements (or, indeed, at least k elements for any fixed k). This is because we can define an arbitrary map on a finite set of structures by a first-order formula, so we just need to take the disjunction of Θ with the formula that defines the required interpretation on the structures with one element. With this in mind, we define the method of *finite expansions* which gives us interpretations Θ that take a structure \mathbb{A} with universe A to a structure with a universe consisting of l labelled disjoint copies of S for some definable subset S of A . Note that Θ would not, in general, be linear, but it is linearly bounded.

So, fix a value l , and let t be the least integer such that $l \leq 2^t$. In a structure \mathbb{A} with at least two elements, we say that a $t + 1$ -tuple of elements (a_1, \dots, a_{t+1}) codes an integer

$i \in [2^t]$ if $b_1 \cdots b_t$ is the binary representation of $i - 1$ and $b_j = 1$ if, and only if, $a_{j+1} \neq a_1$. For each i , we can clearly define a formula $\gamma_i(\bar{y})$ with $t + 1$ free variables that defines those tuples that code i . Now, for any formula $\sigma(x)$, let $\delta(x, \bar{y})$ be the formula $\sigma(x) \wedge \bigvee_{i \leq l} \gamma_i(\bar{y})$ and let $\epsilon(x_1, \bar{y}_1, x_2, \bar{y}_2)$ be the formula $x_1 = x_2 \wedge \bigvee_i \gamma_i(\bar{y}_1) \wedge \gamma_i(\bar{y}_2)$. In other words, δ picks out those $t + 2$ tuples (s, \bar{a}) where s satisfies σ and \bar{a} codes an integer in $[l]$, and ϵ identifies distinct tuples which have the same s and the same integer l . An interpretation using these can be seen to yield a structure with l disjoint copies of the set of elements of \mathbb{A} satisfying σ .

3 The Basic Gap Construction

The problems 3SAT and 3XOR both ask to decide if a formula consisting of the conjunction of Boolean constraints each on exactly three Boolean variables is satisfiable. In 3SAT the constraints are disjunctions of literals on three distinct variables. In 3XOR the constraints are parities of three distinct variables. Both problems are known to have unbounded counting width [6]: the class of satisfiable instances cannot be separated in C^k , for bounded k , from the class of unsatisfiable ones. Our aim is to show that this result can be strengthened to show that the class of satisfiable instances is not C^k -separable (for constant or, indeed, moderately growing values of k) from the class of instances that are *highly unsatisfiable*, meaning that no assignment to the variables can satisfy more than a fraction s of the constraints for some fixed $s \in (0, 1)$. In this section, we give a basic construction for 3XOR, based on that in [6], that establishes this for any $s > 3/4$, with a lower bound on the value of k that is linear in the number of variables in the system.

3.1 Systems of constraints

Let Γ be a finite set of relations over a finite domain D , also called a *constraint language*. Let $I = \{c_1, \dots, c_m\}$ be a collection (multi-set) of constraints, each of the form $R(x_{i_1}, \dots, x_{i_k})$, where R is a k -ary relation in Γ , and x_{i_1}, \dots, x_{i_k} are k distinct D -valued variables from a set x_1, \dots, x_n of n variables. For $c \in [0, 1]$, we say that the system I is c -satisfiable if there is an assignment $f : \{x_1, \dots, x_n\} \rightarrow D$ that satisfies at least cm constraints; i.e., that satisfies $(f(x_{i_1}), \dots, f(x_{i_k})) \in R$ for at least cm constraints $R(x_{i_1}, \dots, x_{i_k})$ from I . Note that, as we are counting the number of satisfied constraints, multiplicities matter and this is why we have multi-sets rather than sets of constraints.

We think of a system $I = \{c_1, \dots, c_m\}$ over the constraint language Γ as a finite structure in two ways. In the first encoding, the universe is the disjoint union of x_1, \dots, x_n and c_1, \dots, c_m . The vocabulary includes binary relations E_1, E_2, \dots such that $E_i(x, c)$ holds if the constraint c has arity at least i and x is the i th variable in c . The vocabulary also includes a unary relation Z_R for each relation R in Γ such that $Z_R(c)$ holds if c is an R -constraint: a constraint of the form $R(x_{i_1}, \dots, x_{i_k})$ for some variables x_{i_1}, \dots, x_{i_k} , where k is the arity of R . In the second encoding, the universe is just the set of variables x_1, \dots, x_n , and the vocabulary includes a k -ary relation symbol R for each k -ary relation R in Γ , such that $R(x_{i_1}, \dots, x_{i_k})$ holds if this is one of the constraints in the collection c_1, \dots, c_m . Note that in this second encoding the collection of constraints is treated as a set. In particular, the multiplicity of constraints is lost, which could affect its c -satisfiability.

The constraint language Γ is also encoded as a finite structure in two ways. In the first encoding the domain is $D^{\leq r} = D \cup D^2 \cup D^3 \cup \dots \cup D^r$, where r is the maximal arity of a relation in Γ . The relations E_1, E_2, \dots are interpreted by the projections: $E_i(b, (b_1, \dots, b_k))$ holds for $b \in D$ and $(b_1, \dots, b_k) \in D^k$ if, and only if, $i \leq k$ and $b = b_i$. The relations Z_R are interpreted by the relation R itself as a unary relation over the universe: $Z_R((b_1, \dots, b_k))$

holds if k is the arity of R and (b_1, \dots, b_k) belongs to R . In the second encoding, the universe is just D , and the relation symbol R is interpreted by R itself. Where it causes no confusion, we do not distinguish between a constraint language Γ and the structure that encodes it, and similarly between an instance I and its encoding structure.

It is easily seen that, in both encodings as finite structures, a system I over Γ is satisfiable if, and only if, there is a homomorphism from the structure that encodes I to the structure that encodes Γ . We say that the system is k -locally satisfiable if $I \Rightarrow_k \Gamma$.

For 3SAT, the constraint language is denoted $\Gamma_{3\text{SAT}}$. It has domain $D = \{0, 1\}$ and the relations are the eight relations $R_1, \dots, R_8 \subseteq \{0, 1\}^3$ defined by the eight possible clauses on three variables. For 3XOR, the constraint language is denoted $\Gamma_{3\text{XOR}}$. It also has domain $D = \{0, 1\}$ and the relations are the two relations $R_0, R_1 \subseteq \{0, 1\}^3$ defined by the two possible linear equations $x + y + z = b$ with three variables over $\mathbb{F}_2 = \{0, 1\}$. Accordingly, 3XOR instances can be identified with systems of linear equations $Ax = b$ over \mathbb{F}_2 .

3.2 Gap construction

We now focus on 3XOR and hence on systems of linear equations over \mathbb{F}_2 . A starting point for us is the following construction which allows us to convert any k -locally satisfiable system of equations into a pair of systems that are \equiv_{C^k} -indistinguishable. See [1, Prop. 32] for a related construction, which is inspired by the proof in [6] that satisfiability of systems of linear equations over \mathbb{F}_2 is not invariant under \equiv_{C^k} for any k .

For any instance I of 3XOR we define another instance $G(I)$ of 3XOR which has two variables x_j^0 and x_j^1 for each variable x_j of I . For each equation $x_j + x_k + x_l = b$ in I , we have eight equations in $G(I)$ given by the eight possible values of $a_1, a_2, a_3 \in \{0, 1\}$ in $x_j^{a_1} + x_k^{a_2} + x_l^{a_3} = b + a_1 + a_2 + a_3$. We now establish some properties of this construction.

► **Lemma 3.1.** *For any instance I of 3XOR and any $c, s \in [0, 1]$, the following hold:*

1. *if I is c -satisfiable, then $G(I)$ is c -satisfiable,*
2. *if I is not s -satisfiable, then $G(I)$ is not $(1/2 + s/2)$ -satisfiable.*

Proof. In Appendix B. ◀

If I is the system $Ax = b$, then the homogeneous companion of I is the system $Ax = 0$, which we denote I^0 . Since any homogeneous system is satisfiable, the system $G(I^0)$ is satisfiable for any I by Lemma 3.1. We show that, despite this, as long as I is locally satisfiable, then $G(I)$ is hard to distinguish from its homogeneous companion $G(I^0)$.

► **Lemma 3.2.** *For any instance I of 3XOR and any k , if I is k -locally satisfiable, then $G(I) \equiv_{C^k} G(I^0)$.*

Proof. In Appendix B. ◀

To apply this construction to get a gap, we need the following fact. Entirely analogous claims have been known and proved in the context of the proof complexity of propositional resolution; indeed, our proof builds on the methods for resolution width [10], and their relationship to existential pebble games from [5, 7].

In the proof, we need the notion of a graph G that is a *bipartite unique-neighbour expander graph with parameters (m, n, d, s, e)* where m, n, d and s are integer parameters with $s < n$ and e is a positive real number. What this means is that G is a bipartite graph with parts U and V with m and n vertices respectively; each $u \in U$ has exactly d neighbours in V ; and for every $A \subseteq U$ with $|A| \leq s$ we have $|\partial A| \geq e|A|$, where $|\partial A|$ denotes the set of vertices in V that are *unique neighbours* of A ; i.e., they are neighbours of a single vertex in A .

► **Lemma 3.3.** *For every $\epsilon > 0$ there exist an integer $c > 0$ and a $\gamma > 0$ such that for every sufficiently large integer n there is an instance I of 3XOR with n variables and cn equations such that I is not $(1/2 + \epsilon)$ -satisfiable and I is k -locally satisfiable for $k \leq \gamma n$.*

Proof. Fix $\epsilon > 0$ and let $c > 1/\epsilon^2$. Let $n \geq 2$ be sufficiently large that we can construct a graph G that is a bipartite unique-neighbour expander graph with parameters $(cn, n, 3, \alpha n, \epsilon)$ for a fixed $\alpha > 0$. For the existence of such graphs with these parameters see [26, Chapter 4]. For each $b = (b_u : u \in U) \in \{0, 1\}^U$, we produce an instance I of 3XOR by introducing one variable x_v for each $v \in V$, and one equation $e_u : x_{v_1(u)} + x_{v_2(u)} + x_{v_3(u)} = b_u$ for each $u \in U$. We claim that there is at least one choice of $b \in \{0, 1\}^U$ that makes I be not $(1/2 + \epsilon)$ -satisfiable. We also show that every choice of $b \in \{0, 1\}^U$ gives that I is k -locally satisfiable for $k \leq \gamma n$ with $\gamma = \epsilon\alpha/9$.

► **Claim 3.4.** *There exists $b \in \{0, 1\}^U$ such that system I is not $(1/2 + \epsilon)$ -satisfiable.*

Proof. We prove that such a b exists by the probabilistic method: a random $b \in \{0, 1\}^U$ has a good chance of making I be not $(1/2 + \epsilon)$ -satisfiable. For each assignment $f : \{x_v : v \in V\} \rightarrow \{0, 1\}$ and each $u \in U$, let $X_{f,u}$ be the indicator random variable for the event that $f(x_{v_1(u)}) + f(x_{v_2(u)}) + f(x_{v_3(u)}) = b_u$; i.e., for the event that f satisfies the equation $x_{v_1(u)} + x_{v_2(u)} + x_{v_3(u)} = b_u$. The probability of this event is $1/2$, and all such events, as u ranges over U , are mutually independent. Thus, setting $X_f = \sum_{u \in U} X_{f,u}$, we have that X_f is a binomial random variable with expectation $\mathbb{E}[X_f] = m/2$. By Hoeffding's inequality, the probability that $X_f - \mathbb{E}[X_f] \geq t$ is at most $e^{-2t^2/m}$. In particular, the probability that $X_f \geq (1/2 + \epsilon)m$ is at most $e^{-2\epsilon^2 m}$. By the union bound, the probability that some f satisfies $X_f \geq (1/2 + \epsilon)m$ is at most $2^n e^{-2\epsilon^2 m}$. Since $m = cn$ and $c > 1/\epsilon^2$ this probability is at most $2^n e^{-2\epsilon^2 cn}$ and so approaches 0 as n grows. Indeed, it is less than $1/2$ for all values of $n \geq 2$. Thus, for any large enough n there exists a b such that I is not $(1/2 + \epsilon)$ -satisfiable. ◀

► **Claim 3.5.** *For every $b \in \{0, 1\}^U$, every set of at most αn equations from I is satisfiable.*

Proof. For each $A \subseteq U$, let e_A be the set of equations that are indexed by vertices in A , and let v_A be the set of variables that appear in e_A . We prove, by induction on $t \leq \alpha n$, that if $A \subseteq U$ and $|A| = t$, then there exists an assignment that sets all the variables in v_A and that satisfies all the equations in e_A . For $t = 0$ the claim is obvious. Assume now that $1 \leq t \leq \alpha n$ and let A be a subset of U of cardinality t . Then $|\partial A| \geq \epsilon|A| > 0$. Let v_0 be some element in ∂A and let $u_0 \in A$ be the unique neighbour of v_0 in A . The induction hypothesis applied to $B = A \setminus \{u_0\}$ gives an assignment g that sets all the variables in v_B and satisfies all the equations in e_B . The assignment g may assign some of the variables of the equation e_{u_0} , but not all, since v_0 is not a neighbour of any vertex in B . Let f be the unique extension of g that first sets all the variables in $v_A \setminus (v_B \cup \{x_{v_0}\})$ to 0, and then sets x_{v_0} to the unique value that satisfies the equation e_{u_0} . This assignment sets all the variables in v_A and satisfies all the equations in e_A . The proof is complete. ◀

► **Claim 3.6.** *For every $b \in \{0, 1\}^U$ and $k \leq \gamma n$, the instance I is k -locally satisfiable.*

Proof. If I is satisfiable, then Duplicator certainly has a winning strategy and there is nothing to prove. Assume then that I is unsatisfiable and let I' be a minimally unsatisfiable subsystem; a subset of the equations of I that is unsatisfiable and every proper subset of it is satisfiable. For each equation $e_u : x_{v_1(u)} + x_{v_2(u)} + x_{v_3(u)} = b_u$ of I , let F_u be the four clauses $\{x_{v_1(u)}^{(a_1)}, x_{v_2(u)}^{(a_2)}, x_{v_3(u)}^{(a_3)}\}$ with $a_1, a_2, a_3 \in \mathbb{F}_2$ with $a_1 + a_2 + a_3 = b_u$, where $z^{(a)}$ stands

for the negative literal $\neg z$ if $e = 0$ and the positive literal z if $e = 1$. Let F be the 3CNF formula that is the union of all the F_u as u ranges over U . Observe that F is an unsatisfiable 3CNF. We intend to apply Theorem 5.9 from [10] to it.

Let \mathcal{A} be the collection of all Boolean functions $f_u : \{0, 1\}^V \rightarrow \{0, 1\}$ defined by

$$f_u(x_v : v \in V) = x_{v_1(u)} + x_{v_2(u)} + x_{v_3(u)} + b_u \pmod 2,$$

for $u \in U$. Each function in \mathcal{A} is sensitive in the sense of Definition 5.5 from [10], and compatible with F in the sense of Definition 5.3 from [10]. Moreover, if $\mathcal{A}_0 \subseteq \mathcal{A}$ is the set of functions that corresponds to the minimally unsatisfiable subsystem I' of I , then its cardinality m_0 satisfies $m_0 > \alpha n$ by Claim 3.5. It follows that the expansion $e(\mathcal{A})$ in the sense of Definition 5.8 from [10] is at least $e\alpha n/3$. By Theorem 5.9 in [10], every resolution refutation of F requires width at least $e\alpha n/3$, and hence at least $3k$ since $k \leq \gamma n = e\alpha n/9$. By Theorem 2 in [7], Duplicator has a winning strategy for the existential $3k$ -pebble game played on the structures F and the constraint language $\Gamma_{3\text{SAT}}$ of 3SAT, in the second encoding discussed in Section 3.1. We use this winning strategy to design a winning strategy for Duplicator in the existential k -pebble game played on I and $\Gamma_{3\text{XOR}}$.

While playing the game on I , Duplicator plays the game on F on the side and keeps the invariant that each pebbled variable in the game on I is also pebbled in the side game, and each pebbled equation in the game on I has its three variables pebbled in the side game. Whenever a new variable is pebbled in the game on I , Duplicator pebbles the same variable in the side game, and copies the answer from its strategy on it. Whenever a new equation is pebbled in the game on I , Duplicator pebbles its three variables in the side game, and answers the pebbled equation accordingly from its strategy. Since at each position of the game on I there are no more than k pebbles on the board, at each time during the simulation the side game has no more than $3k$ pebbles on the board. This shows that the simulation can be carried on forever and the proof is complete. \blacktriangleleft

This completes the proof of Lemma 3.3. \blacktriangleleft

We can now prove our first two gap theorems.

► **Theorem 3.7.** *For any $\epsilon > 0$, if \mathcal{C} is the collection of 3XOR instances that are satisfiable and \mathcal{D} is the collection of 3XOR instances that are not $(3/4 + \epsilon)$ -satisfiable, then \mathcal{C} and \mathcal{D} are not C^k -separable for any $k = o(n)$.*

Proof. By Lemma 3.3, there is a family of systems $(S_k)_{k \geq 1}$ with $O(k)$ variables and equations such that S_k is k -locally satisfiable but not $(1/2 + 2\epsilon)$ -satisfiable. Let $I_k^1 = G(S_k)$ and $I_k^0 = G(S_k^0)$. Note that, by Lemma 3.1, the system I_k^0 is satisfiable and I_k^1 is not $(3/4 + \epsilon)$ -satisfiable. However, $I_k^0 \equiv_{C^k} I_k^1$ by Lemma 3.2. Since each of I_k^0 and I_k^1 has two variables for each variable in S_k and eight equations for each equation in S_k , they also have $O(k)$ variables and equations and the result follows. \blacktriangleleft

► **Theorem 3.8.** *For any $\epsilon > 0$, if \mathcal{C} is the collection of 3SAT instances that are satisfiable and \mathcal{D} is the collection of 3SAT instances that are not $(15/16 + \epsilon)$ -satisfiable, then \mathcal{C} and \mathcal{D} are not C^k -separable for any $k = o(n)$.*

Proof. Consider again the reduction Θ from 3XOR to 3SAT given by translating each equation into a conjunction of four clauses. Thus $x + y + z = d$ translates into the four clauses $\{x^{(a)}, y^{(b)}, z^{(c)}\}$ with $a, b, c \in \mathbb{F}_2$ with $a + b + c = d$, where $z^{(e)}$ stands for the negative literal $\neg z$ if $e = 0$ and the positive literal z if $e = 1$. This is easily defined in first-order logic. As the set of variables in I is the same as in $\Theta(I)$, it is linearly bounded. We claim that

applying Θ to Theorem 3.7 with ϵ reset to $\epsilon/4$ gives the theorem through Lemma 2.1. First, it is clear that if I is a 3XOR instance that is satisfiable, then $\Theta(I)$ is also satisfiable. Now, suppose that I is a system of m equations that is not $(3/4 + \epsilon/4)$ -satisfiable, and let g be an assignment of truth values to the variables X of $\Theta(I)$. Applied to I , the assignment g falsifies at least $(1/4 - \epsilon/4)m$ of the equations. For each equation, g must falsify at least one of the four corresponding clauses in $\Theta(I)$. Thus, g falsifies at least $(1/4 - \epsilon/4)m$ clauses in $\Theta(I)$ and so satisfies at most $4m - (1/4 - \epsilon/4)m = (15/16 + \epsilon) \cdot 4m$ of the $4m$ clauses. \blacktriangleleft

4 Amplifying the Gap

In this section we show that certain reductions from the theory of inapproximability can be expressed as FO-interpretations, allowing us to derive optimal and unconditional undefinability results that match the optimal NP-hardness results from [16].

4.1 Parallel repetition

An instance I of the LABEL COVER problem is given by two disjoint sets of variables U and V with domains of values A and B , respectively, a predicate $P : U \times V \times A \times B \rightarrow \{0, 1\}$, and an assignment of weights $W : U \times V \rightarrow \mathbb{N}$. If all the non-zero weights $W(u, v)$ are equal, then the instance is said to have *uniform weights*. If for all $u \in U$ the sums $W(u) := \sum_{v \in V} W(u, v)$ of incident weights are equal, then the instance is called *left-regular*. A *right-regular* instance is defined analogously in terms of $W(v) := \sum_{u \in U} W(u, v)$. The instance is a *projection game* if for every $(u, v) \in U \times V$ with $W(u, v) \neq 0$ it holds that for every $a \in A$ there is exactly one $b \in B$ satisfying $P(u, v, a, b) = 1$. It is called a *unique game* if $|A| = |B|$ and it is a projection game both ways: from A to B , and from B to A . The instance is said to have parameters (m, n, p, q) if $|U| = m$, $|V| = n$, $|A| = p$ and $|B| = q$. Its *domain size* is $p + q$.

A value-assignment for an instance I is a pair of functions $f : U \rightarrow A$ and $g : V \rightarrow B$. The weight $v(f, g)$ of the value-assignment (f, g) is the total weight of the pairs $(u, v) \in U \times V$ satisfying the constraint $P(u, v, f(u), g(v)) = 1$; i.e.,

$$v(f, g) = \sum_{(u, v) \in U \times V} W(u, v) P(u, v, f(u), g(v)). \quad (1)$$

For $c \in [0, 1]$, we say that the instance is c -satisfiable if there is a value-assignment whose weight is at least $c \cdot W_0$, where $W_0 = \sum_{(u, v) \in U \times V} W(u, v)$ is the maximum possible weight. We call it satisfiable if it is 1-satisfiable.

The *bipartite reduction* takes an instance I of 3XOR and produces a projection game instance $L(I)$ of LABEL COVER defined as follows. The sets U and V are the set of equations in I and the set of variables in I , respectively. The weight $W(u, v)$ is 1 if v is one of the variables in the equation u , and 0 otherwise. The domains of values associated to U and V are $A = \{(a_1, a_2, a_3) \in \mathbb{F}_2^3 : a_1 + a_2 + a_3 = 0\}$ and $B = \mathbb{F}_2$, respectively. The predicate P associates to the pair (u, v) , where u is the equation $v_1 + v_2 + v_3 = b$ and $v = v_i$ for $i \in \{1, 2, 3\}$, the set of pairs $((a_1, a_2, a_3), a) \in A \times B$ satisfying $a = a_i + b$. In other words, $P(u, v, (a_1, a_2, a_3), a) = 1$ if, and only if, v appears in the equation u , and if u is $v_1 + v_2 + v_3 = b$ and $v = v_i$, then the (partial) assignment $\{v_1 \mapsto a_1 + b, v_2 \mapsto a_2 + b, v_3 \mapsto a_3 + b\}$, which satisfies the equation $v_1 + v_2 + v_3 = b$ by construction, agrees with the (partial) assignment $\{v_i \mapsto a\}$. Clearly, this defines a projection game.

► **Lemma 4.1.** *For every instance I of 3XOR and every $c, s \in [0, 1]$, the following hold:*

1. *if I is c -satisfiable, then $L(I)$ is c -satisfiable,*
2. *if I is not s -satisfiable, then $L(I)$ is not $(s + 2)/3$ -satisfiable.*

Moreover, $L(I)$ is a left-regular projection game that has uniform weights.

Proof. Let m be the number of equations in I , so $L(I)$ has exactly $3m$ pairs (u, v) of unit weight. Such pairs are called constraints. For proving 1, let h be an assignment for I that satisfies at least cm of the m equations in I . For each equation u in I , say $v_1 + v_2 + v_3 = b$, define $f(u) = (h(v_1) + b, h(v_2) + b, h(v_3) + b)$ if h satisfies $v_1 + v_2 + v_3 = b$, and define $f(u) = (0, 0, 0)$ otherwise. For each variable v in I , define $g(v) = h(v)$. Each equation in I gives rise to exactly three constraints in $L(I)$, and if the equation is satisfied by h , then all three constraints associated to it in $L(I)$ are satisfied by (f, g) . Thus (f, g) satisfies at least $3cm$ of the $3m$ constraints in $L(I)$, so $L(I)$ is c -satisfiable. For proving 2, let (f, g) be an assignment for $L(I)$ that satisfies at least $(s + 2)m$ of the $3m$ constraints in $L(I)$. For each variable v in I , define $h(v) = g(v)$. Let t be the number of equations of I that are satisfied by h . In terms of t , the assignment (f, g) satisfies at most $3t + 2(m - t)$ of the $3m$ constraints of $L(I)$. Thus $t \geq sm$, so I is s -satisfiable. ◀

The *parallel repetition reduction* takes an instance I of LABEL COVER, and a positive integer $t \geq 1$, and produces another instance $R(I, t)$ of LABEL COVER defined as follows. Let U and V be the sets of variables in I and let $W : U \times V \rightarrow \mathbb{N}$ be the weight assignment. The sets of variables of $R(I, t)$ are U^t and V^t . For $\bar{u} = (u_1, \dots, u_t) \in U^t$ and $\bar{v} = (v_1, \dots, v_t) \in V^t$, the weight $W(\bar{u}, \bar{v})$ is defined as $\prod_{i=1}^t W(u_i, v_i)$. If A and B are the domains of values associated to U and V , then the domains of values associated to U^t and V^t are A^t and B^t respectively. For $\bar{u} = (u_1, \dots, u_t) \in U^t$, $\bar{v} = (v_1, \dots, v_t) \in V^t$, $\bar{a} = (a_1, \dots, a_t) \in A^t$ and $\bar{b} = (b_1, \dots, b_t) \in B^t$, the predicate $P(\bar{u}, \bar{v}, \bar{a}, \bar{b})$ is defined as $\prod_{i=1}^t P(u_i, v_i, a_i, b_i)$. Observe that this definition guarantees that if I is a projection game, then so is $R(I, t)$.

► **Theorem 4.2** (Parallel Repetition Theorem [24, 18]). *There exists a constant $\alpha > 0$ such that for every instance I of LABEL COVER with domain size at most $d \geq 1$, every $s \in [0, 1]$ and every $t \geq 1$ the following hold:*

1. *if I is satisfiable, then $R(I, t)$ is satisfiable,*
2. *if I is not s -satisfiable, then $R(I, t)$ is not $(1 - (1 - s)^3)^{\alpha t/d}$ -satisfiable.*

Moreover, if I is a projection game, left-regular, right-regular, or has uniform weights, then so is $R(I, t)$.

Although it is the case that the bipartite and the parallel repetition reductions are both FO-interpretations, we do not need to formulate this. Instead, we show the FO-definability of the composition of these reductions with the long-code reductions that we discuss next.

4.2 First long-code reduction

The *first long-code reduction* that we consider takes a projection game instance I of LABEL COVER and a rational $\epsilon \in [0, 1]$ and produces an instance $C(I, \epsilon)$ of 3XOR defined as follows. Let U and V be the sets of variables of sizes m and n , respectively, with associated domains of values $A = [p]$ and $B = [q]$, let $W : U \times V \rightarrow \mathbb{N}$ be the weight assignment, let $P : U \times V \times A \times B \rightarrow \{0, 1\}$ be the predicate of I , and for each $(u, v) \in U \times V$ with $W(u, v) \neq 0$ and each $a \in A$ let $\pi_{u,v}(a)$ be the unique value $b \in B$ that satisfies $P(u, v, a, b) = 1$. The existence of such a function $\pi_{u,v} : A \rightarrow B$ is guaranteed from the assumption that I is a projection game. The set of variables of $C(I, \epsilon)$ includes one variable $u(a)$ for each $u \in U$ and $a \in \mathbb{F}_2^{p-1}$, and one variable $v(b)$ for each $v \in V$ and $b \in \mathbb{F}_2^{q-1}$,

for a total of $m2^{p-1} + n2^{q-1}$ variables. Before we are able to define the set of equations of $C(I, \epsilon)$ we need a piece of notation. For a vector $z = (z_1, \dots, z_d) \in \mathbb{F}_2^d$ of dimension $d \geq 2$, we write $S(z) = z_d$ and $F(z) = (z_1 + S(z), \dots, z_{d-1} + S(z))$. Note that $S(z)$ is a single field element, and $F(z)$ is a vector of dimension $d - 1$. With this notation, the set of equations of $C(I, \epsilon)$ includes $W(u, v) \cdot M^q \cdot \epsilon^D \cdot (1 - \epsilon)^{q-D}$ copies of the equation $v(F(x)) + u(F(y)) + u(F(z)) = S(x) + S(y) + S(z)$ for each $(u, v) \in U \times V$, each $x \in \mathbb{F}_2^q$ and each $y, z \in \mathbb{F}_2^p$, where M is the denominator of $\epsilon = N/M$ reduced to lowest terms, D is the number of positions $i \in [p]$ such that $z_i \neq x_{\pi(i)} + y_i$, and $\pi = \pi_{u,v}$ if $W(u, v) \neq 0$.

► **Theorem 4.3** (Håstad 3-Query Linear Test [16]). *For every $s, \epsilon \in [0, 1]$ with $\epsilon > 0$ and $s > 0$ and every projection game instance I of LABEL COVER, the following hold:*

1. *if I is satisfiable, then $C(I, \epsilon)$ is $(1 - \epsilon)$ -satisfiable,*
2. *if I is not s -satisfiable, then $C(I, \epsilon)$ is not $(1/2 + (s/\epsilon)^{1/2}/4)$ -satisfiable.*

The proof of Theorem 4.3 follows from Lemmas 5.1 and 5.2 in [16]. There are notational differences that may obscure this and a detailed explanation is provided in Appendix C.

Next, by composing Lemma 4.1, Theorem 4.2, and Theorem 4.3 with the appropriate parameters we get the following:

► **Theorem 4.4.** *For every $s, \epsilon \in [0, 1]$ with $0 < s < 1$ and $\epsilon > 0$, there is an FO-interpretation Θ that maps instances of 3XOR to instances of 3XOR in such a way that, for every 3XOR instance I the following hold:*

1. *if I is satisfiable, then $\Theta(I)$ is $(1 - \epsilon)$ -satisfiable,*
2. *if I is not s -satisfiable, then $\Theta(I)$ is not $(1/2 + \epsilon)$ -satisfiable.*

Proof. First we define $\Theta(I)$ and then check that this definition is an FO-interpretation. In anticipation for the proof, let t be a large enough integer so that the following inequality holds:

$$(1 - (1 - (s + 2)/3)^3)^{\alpha t/6} \leq 16\epsilon^3, \quad (2)$$

where α is the constant in Theorem 4.2. Such a t exists because $s < 1$ and $\epsilon > 0$. Apply the bipartite reduction to I to obtain the instance $I' = L(I)$ from Lemma 4.1. Observe that the domain size d of I' is $|A| + |B| = 6$. Next apply the parallel repetition reduction to I' with parameter t to obtain a new instance I'' . Finally apply the long-code reduction to I'' with parameter ϵ to obtain the system I''' . The parameters were chosen in a way that the system I''' satisfies properties 1 and 2, through Theorem 4.3.

It remains to argue that I''' can be produced from I by an FO-interpretation. To define I' from I there is no difficulty at all: the FO-interpretation is even linear. To define I'' from I' we note that t is a constant, and that the weights $W(u, v)$ of I' are 0 or 1, so again there is no difficulty. In this case the FO-interpretation has dimension t , and it is n^t -bounded. To define I''' from I'' we note that the domain sizes p and q of the instance I'' are constants, indeed $p = 4^t$ and $q = 2^t$. This means that there are $|U| \cdot 2^{p-1}$ variables of type $u(a)$, and $|V| \cdot 2^{q-1}$ variables of type $v(b)$, and these are constant multiples of $|U|$ and $|V|$, respectively. Such domains are FO-definable by the method of finite expansions (see Section 2). Finally, since the weights $W(u, v)$ of I'' are still zeros or ones and both ϵ and q are constants, the multiplicities of the equations of I''' are also constants, and hence FO-definable. ◀

4.3 Second long-code reduction

The *second long-code reduction* takes a projection game instance I of LABEL COVER and a rational $\delta \in [0, 1]$ and produces an instance $D(I, \delta)$ of 3SAT defined as follows. Before we define $D(I, \delta)$, let us define an intermediate instance $D'(I, \epsilon)$ of 3SAT that takes a different

parameter $\epsilon \in [0, 1]$. Let $U, V, m, n, A, B, p, q, W, P$, and $\pi_{u,v}(a)$ be as in the first long-code reduction. The set of variables of $D(I, \epsilon)$ is defined as in the first long-code reduction: a variable $u(a)$ for each $u \in U$ and each $a \in \mathbb{F}_2^{p-1}$, and a variable $v(b)$ for each $v \in V$ and each $b \in \mathbb{F}_2^{q-1}$. We also use the *folding* notation $F(z)$ and $S(z)$ from the first long-code reduction. Now the instance $D'(I, \epsilon)$ includes $W(u, v) \cdot M^q \cdot \epsilon^D \cdot (1 - \epsilon)^{E-D} \cdot H$ copies of the clause $\{v(F(x))^{(S(x))}, u(F(y))^{(S(y))}, u(F(z))^{(S(z))}\}$ for each $(u, v) \in U \times V$, each $x \in \mathbb{F}_2^q$ and each $y, z \in \mathbb{F}_2^p$, where M is the denominator of $\epsilon = N/M$ reduced to lowest terms, E is the number of positions $i \in [p]$ with $x_{\pi(i)} = 1$ and D is the number of positions $i \in [p]$ with $x_{\pi(i)} = 1$ and $z_i \neq y_i$ for $\pi = \pi_{u,v}$ if $W(u, v) \neq 0$, while $H \in \{0, 1\}$ is the indicator for the event that in each position $i \in [p]$ with $x_{\pi(i)} = 0$ we have $z_i \neq y_i$. Finally, to define the instance $D(I, \delta)$, set $t = \lceil \delta^{-1} \rceil$ and $\epsilon_1 = \delta$, and $\epsilon_{i+1} = \delta^{71} 2^{-35} \epsilon_i$ for $i = 1, \dots, t-1$, and let the instance be $\bigcup_{i=1}^t D'(I, \epsilon_i)$.

► **Theorem 4.5** (Håstad 3-Query Disjunction Test [16]). *There exists $s_0 > 0$ such that for every $s \in [0, 1]$ with $0 < s < s_0$ and every projection game instance I of LABEL COVER the following hold:*

1. *if I is satisfiable, then $C(I, \epsilon)$ is satisfiable,*
2. *if I is not s -satisfiable, then $C(I, \epsilon)$ is not $(7/8 + \log_2(1/s)^{-1/2})$ -satisfiable.*

For the proof of Theorem 4.5, see Lemmas 6.12 and 6.13 in [16]. As in the first long-code reduction, some explanation is needed for seeing this.

Besides the notational differences that were already pointed out in the first long-code reduction, the second long-code reduction adds the following. First, the constants 71 and 35 in the definition of ϵ_{i+1} come from setting $c = 1/35$ in the definition of Test $F3S^\delta(u)$ in [16]. According to Lemma 6.9 in [16], this is an acceptable setting of c . Second, the constant $s_0 > 0$ in Theorem 4.5 is meant to be chosen small enough so as to ensure that, for each s satisfying $s < s_0$, we have $2^{-64\delta^{-2}/25} < 2^{-d\delta^{-1} \log_2(\delta^{-1})}$ for $\delta = 8 \log_2(1/s)^{-1/2}/5$, where d is the constant hidden in the asymptotic O -notation of Lemma 6.13 in [16]. Such an s_0 exists because $N \log_2(N) = o(N^2)$ as $N \rightarrow +\infty$. With this notation, Lemma 6.12 in [16] gives point 1, and Lemma 6.13 in [16] with $\delta = 8 \log_2(1/s)^{-1/2}/5$ gives point 2 in Theorem 4.5.

By composing Lemma 4.1, Theorem 4.2, and Theorem 4.5 with the appropriate parameters we get the following:

► **Theorem 4.6.** *For every $s, \epsilon \in [0, 1]$ with $0 < s < 1$ and $\epsilon > 0$, there is an FO-interpretation Θ that maps instances of 3XOR to instances of 3SAT in such a way that, for every 3XOR instance I the following hold:*

1. *if I is satisfiable, then $\Theta(I)$ is satisfiable,*
2. *if I is not s -satisfiable, then $\Theta(I)$ is not $(7/8 + \epsilon)$ -satisfiable.*

Proof. First we define $\Theta(I)$ and then check that this definition is an FO-interpretation. Let t be a large enough integer so that the following inequality holds:

$$(1 - (1 - (s + 2)/3)^3)^{\alpha t/6} \leq \min\{2^{-1/\epsilon^2}, s_0\} \quad (3)$$

where α is the constant in Theorem 4.2 and $s_0 > 0$ is small enough as in Theorem 4.5. Such a t exists because $s < 1$ and $\epsilon > 0$ as well as $s_0 > 0$. Apply the bipartite reduction to I to obtain the instance $I' = L(I)$ from Lemma 4.1. Observe that the domain size d of I' is $|A| + |B| = 6$. Next apply the parallel repetition reduction to I' with parameter t to obtain a new instance I'' . Finally apply the second long-code reduction to I'' to obtain the system I''' . The parameters were chosen so that the system I''' satisfies properties 1 and 2, through Theorem 4.5. As in the proof of Theorem 4.4 this reduction is FO-definable. ◀

4.4 Optimal gap inexpressibility

We are ready to state the main results of this section. Composing Theorem 3.7, Theorem 4.4, and Lemma 2.1 we get the following.

► **Theorem 4.7.** *For each $\epsilon > 0$, there is a $\delta > 0$ such that if \mathcal{C} is the collection of 3XOR instances that are $(1 - \epsilon)$ -satisfiable and \mathcal{D} is the collection of 3XOR instances that are not $(1/2 + \epsilon)$ -satisfiable then \mathcal{C} and \mathcal{D} are not C^k -separable for any $k = o(n^\delta)$.*

Composing Theorem 3.7, Theorem 4.6, and Lemma 2.1 we get the following.

► **Theorem 4.8.** *For each $\epsilon > 0$, there is a $\delta > 0$ such that if \mathcal{C} is the collection of 3SAT instances that are satisfiable and \mathcal{D} is the collection of 3SAT instances that are not $(7/8 + \epsilon)$ -satisfiable then \mathcal{C} and \mathcal{D} are not C^k -separable for any $k = o(n^\delta)$.*

A statement similar to Theorem 4.8 can be obtained from applying the standard reduction from 3XOR to 3SAT to Theorem 4.7 as in Theorem 3.8. However, this would *only* show that the class of 3SAT instances that are $(1 - \epsilon)$ -satisfiable is C^k -inseparable from the class of instances that are not $(7/8 + \epsilon)$ -satisfiable; note that Theorem 4.8 states the stronger claim that this is the case for the class of *fully satisfiable* instances, instead of the $(1 - \epsilon)$ -satisfiable ones. A natural question to ask is whether the $(1 - \epsilon)$ in Theorem 4.7 could be improved to 1. This would, however, require different techniques since there is a polynomial-time algorithm that separates the satisfiable instances of 3XOR from the unsatisfiable ones.

On the other hand, $7/8 + \epsilon$ bound in Theorem 4.8 and the $1/2 + \epsilon$ bound in Theorem 4.7 are optimal. Every instance of 3SAT is $7/8$ -satisfiable, and every instance of 3XOR is $1/2$ -satisfiable. Thus, the algorithms that achieve these approximation ratios are trivial and expressible in FPC.

It is also worth comparing the statement of Theorem 3.8 to that of Theorem 4.8. While the latter has the stronger bound on the approximability ratio ($7/8$ rather than $15/16$) the former gives the stronger lower bound on the counting width. One significance of the lower bounds on counting width is that they provide bounds on the number of levels of semidefinite programming hierarchies such as Lasserre hierarchy needed to solve a problem. Thus, it is known [13, 9] that if a constraint maximization problem can be solved using t levels of the Lasserre hierarchy, its counting width is at most $O(t)$. Thus, it is an immediate consequence of our results that approximation algorithms obtained through $o(n^\delta)$ levels of the Lasserre hierarchy cannot achieve an approximation ratio for 3SAT and 3XOR better than the trivial $7/8$ and $1/2$ respectively. These lower bounds on Lasserre relaxations are already known (see [25]) but our results provide a systematic explanation in terms of definability.

5 Vertex Cover

We investigate gap inexpressibility results for the vertex cover problem VC on graphs. Recall that a set $X \subseteq V$ of vertices in a graph $G = (V, E)$ is a *vertex cover* if every edge in E has at least one of its endpoints in X . If the graph comes with a weight function $w : V \rightarrow \mathbb{R}^+$, then the weight of X is the sum of the weights of the vertices in X . If the weights of the vertices are omitted in the specification of the graph, then all the vertices are assumed to have unit weight. The problem of finding the minimum weight vertex cover in a graph is a classic NP-complete problem.

In the following we write $\text{vc}(G)$ for the weight of a minimum weight vertex cover, and $\text{vc}(G) := \text{vc}(G)/W_0$, where $W_0 := \sum_{v \in V} w(v)$. Analogously, we write $\text{IS}(G)$ for the weight of a maximum weight independent set, and $\text{isd}(G) := \text{IS}(G)/W_0$. Clearly $\text{vc}(G) = 1 - \text{isd}(G)$ holds for all weighted graphs.

The standard reduction that proves the NP-completeness of the vertex cover problem (see, e.g. [22, Thm. 9.4]) takes an instance I of 3SAT with n variables and m clauses and gives a graph G with $3m$ vertices in which the minimum vertex cover has size exactly $2cm$, if cm is the maximum number of clauses in I that can be simultaneously satisfied. It is also easy to see that this reduction can be given as an FO-interpretation. This interpretation is linearly bounded and therefore it follows from Theorem 4.8 and Lemma 2.1 that for any $\epsilon > 0$, there is a $\delta > 0$ such that the collection of graphs G with $\text{vc}(G) \leq 7/12 + \epsilon$ and the collection of graphs G with $\text{vc}(G) \geq 2/3 - \epsilon$ cannot be separated in C^k for any $k = o(n^\delta)$. This has the consequence that no approximation algorithm for the vertex cover problem expressible in FPC can achieve an approximation ratio better than $8/7$.

We can improve on this by considering instead the so-called FGLSS reduction from 3XOR to vertex-cover, which we describe next.

► **Theorem 5.1.** *There is a linearly-bounded first-order reduction G that takes an instance I of 3XOR with m equations to a graph $G(I)$ with $4m$ vertices so that if m^* is the maximum number of equations of I that can be simultaneously satisfied, then $\text{vc}(G) = 4m - m^*$.*

Proof. For each equation $x + y + z = b$ in I , $G(I)$ has a 4-clique of vertices, each labelled with a distinct assignment of values to the three variables that make the equation true. In addition, we have an edge between any pair of vertices that are labelled by inconsistent assignments. It is easily seen that the largest independent set in $G(I)$ is obtained by taking an assignment g of values to the variables of I that satisfies m^* equations and, for each satisfied equation, selecting the vertex in its 4-clique that is the projection of g . This yields an independent set of size exactly m^* and the result follows. ◀

From this, and Theorem 3.7, we immediately get the following result.

► **Corollary 5.2.** *For any $\epsilon > 0$, if \mathcal{C} is the collection of graphs G with $\text{vc}(G) \leq 3/4$ and \mathcal{D} is the collection of graphs G with $\text{vc}(G) \geq 13/16 - \epsilon$ then \mathcal{C} and \mathcal{D} are not C^k -separable for any $k = o(n)$.*

Similarly, combining Theorem 5.1 and Theorem 4.7 yields the following corollary.

► **Corollary 5.3.** *For any $\epsilon > 0$, there is a $\delta > 0$ such that, if \mathcal{C} is the collection of graphs G with $\text{vc}(G) \leq 3/4 + \epsilon$ and \mathcal{D} is the collection of graphs G with $\text{vc}(G) \geq 7/8 - \epsilon$ then \mathcal{C} and \mathcal{D} are not C^k -separable for any $k = o(n^\delta)$.*

These two corollaries are incomparable. While the latter yields the stronger approximation ratio ($7/6$ rather than $13/12$), the former gives the stronger lower bound on k .

Better lower bounds on the approximation ratio are known under the assumption that $P \neq NP$. One such lower bound was achieved by Dinur and Safra [14] who showed that, under this assumption, no polynomial-time algorithm for approximating vertex cover can achieve an approximation ratio better than 1.36. In the full version of this paper [8] we argue that this reduction is also an FO-interpretation, so we get the same inapproximability ratio for algorithms that are expressible in FPC, giving a strengthening of Corollary 5.3.

There are straightforward polynomial-time algorithms that yield a vertex cover in a graph with guaranteed approximation ratio 2. It is conjectured that no polynomial-time algorithm can achieve an approximation ratio of $2 - \epsilon$ for any $\epsilon > 0$. It would be interesting to prove a version of this conjecture for algorithms expressible in FPC, and without the assumption that $P \neq NP$. This could be established by a strengthened version of Corollary 5.3 with better ratios. We next show that we can at least do this for the special case of $k = 2$.

► **Theorem 5.4.** *For any $\epsilon > 0$, if \mathcal{C} is the collection of graphs G with $\text{vc}(G) \leq 1/2$ and \mathcal{D} is the collection of graphs G with $\text{vc}(G) \geq 1 - \epsilon$ then \mathcal{C} and \mathcal{D} are not C^2 -separable.*

Proof. Let $(G_n)_{n \in \mathbb{N}}$ be a family of 3-regular expander graphs on n vertices, so that the largest independent set in G_n has size $o(n)$. For the existence of such graphs see [26, Chapter 4]. It follows that the smallest vertex cover in G_n has size $n - o(n)$. Hence, we can choose a value m such that G_{2m} has no vertex cover smaller than $2m(1 - \epsilon)$.

Let H_m be a 3-regular bipartite graph on two sets of m vertices. Now, each part of a bipartite graph is a vertex cover, so H_m has a vertex cover of size m . However, it is known that $G \equiv_{C^2} H$ holds for any pair G and H of d -regular graphs with the same number of vertices, for any d . Thus, $G_{2m} \equiv_{C^2} H_m$ and the result follows. ◀

Essentially, Theorem 5.4 tells us that no algorithm that is invariant under \equiv_{C^2} can determine $\text{vc}(G)$ to an approximation better than 2, and Corollary 5.3 tells us that no algorithm that is invariant under \equiv_{C^k} for constant or even slowly growing k can determine $\text{vc}(G)$ to an approximation better than $7/6$. A legitimate question at this point is whether there is any algorithm that is invariant under \equiv_{C^k} , such as one expressible in FPC would be, that *does* achieve an approximation ratio of 2. The natural polynomial-time algorithms that give a vertex cover with size at most $2\text{vc}(G)$ are not expressible in FPC. Indeed, we cannot expect a formula of FPC to define an actual vertex cover in a graph G as this is not invariant under automorphisms of G . We can only ask for an estimate of the size, i.e. of $\text{vc}(G)$, and this we can get up to a factor of 2. For this, it turns out that $k = 2$ is enough, showing that the lower bound of Theorem 5.4 is tight:

► **Theorem 5.5.** *For any δ , if \mathcal{C} is the collection of graphs G with $\text{vc}(G) \leq \delta$ and \mathcal{D} is the collection of graphs G with $\text{vc}(G) > 2\delta$ then \mathcal{C} and \mathcal{D} are \equiv_{C^2} -separable.*

The proof of Theorem 5.5 can be found in Appendix D.

6 Conclusions

This paper introduces a new method for studying the hardness of approximability of optimization problems by showing that the approximation cannot be *defined* in a suitable logic such as FPC. This is done by showing that no class of bounded counting width can separate instances of the problem with a high optimum from those with a low one. This raises a number of new challenges in the application of this method. A clear demonstration of the power of this method would be to derive a lower bound stronger than one for which NP-hardness is known. For instance, can we improve, in the context of inexpressibility, on the 1.36-inapproximability for vertex cover from the NP-hardness result of Dinur and Safra [14]? In other words, can show that the class of graphs that have a vertex cover of density δ is not C^k -separable from the class of graphs that do not have a vertex cover of density $c\delta$, for some $\delta \in (0, 1)$ and some constant c greater than 1.36? If this were achieved for unbounded k , it would have major consequences in the study of semidefinite programming hierarchies of relaxations of vertex cover. And this applies, indeed, to any optimization problem for which the exact inapproximability factor is not known, including MAX CUT, sparsest cut, etc.

References

- 1 S. Abramsky, A. Dawar, and P. Wang. The pebbling comonad in finite model theory. In *Proc. of the 32nd IEEE Symp. on Logic in Computer Science (LICS)*, 2017.
- 2 M. Anderson, A. Dawar, and B. Holm. Solving linear programs without breaking abstractions. *J. ACM*, 62, 2015.

- 3 S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
- 4 S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
- 5 A. Atserias. On sufficient conditions for unsatisfiability of random formulas. *J. ACM*, 51:281–311, 2004.
- 6 A. Atserias, A. Bulatov, and A. Dawar. Affine systems of equations and counting infinitary logic. *Theoretical Computer Science*, 410(18):1666–1683, 2009.
- 7 A. Atserias and V. Dalmau. A combinatorial characterization of resolution width. *J. Comput. Syst. Sci.*, 74:323–334, 2008.
- 8 A. Atserias and A. Dawar. Definable inapproximability: New challenges for duplicator. *arXiv*, 2018. [arXiv:1806.11307](https://arxiv.org/abs/1806.11307).
- 9 A. Atserias and J. Ochremiak. Definable ellipsoid method, sums-of-squares proofs, and the isomorphism problem. [arxiv 1802.02388](https://arxiv.org/abs/1802.02388).
- 10 E. Ben-Sasson and A. Wigderson. Short proofs are narrow - resolution made simple. *J. ACM*, 48:149–169, 2001.
- 11 A. Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, pages 8–21, 2015.
- 12 A. Dawar and P. Wang. A definability dichotomy for finite valued CSPs. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015*, pages 60–77, 2015.
- 13 A. Dawar and P. Wang. Definability of semidefinite programming and lasserre lower bounds for CSPs. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, 2017. [doi:10.1109/LICS.2017.8005108](https://doi.org/10.1109/LICS.2017.8005108).
- 14 I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162:439–485, 2005.
- 15 Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*, volume 47 of *Lecture Notes in Logic*. Cambridge University Press, 2017.
- 16 J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.
- 17 Lauri Hella. Logical hierarchies in PTIME. *Information and Computation*, 129(1):1–19, 1996.
- 18 T. Holenstein. Parallel repetition: Simplifications and the no-signaling case. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing, STOC '07*, pages 411–419, New York, NY, USA, 2007. ACM. [doi:10.1145/1250790.1250852](https://doi.org/10.1145/1250790.1250852).
- 19 N. Immerman and E. S. Lander. Describing graphs: A first-order approach to graph canonization. In A. Selman, editor, *Complexity Theory Retrospective*. Springer-Verlag, 1990.
- 20 S. Khot, D. Minzer, and M. Safra. Pseudorandom sets in grassmann graph have near-perfect expansion. Technical Report TR18-006, Electronic Colloquium on Computational Complexity (ECCC), 2018.
- 21 Phokion G Kolaitis and Moshe Y Vardi. On the expressive power of Datalog: Tools and a case study. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 61–71. ACM, 1990.
- 22 Ch. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- 23 Ch. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991.
- 24 R. Raz. A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803, 1998.
- 25 G. Schoenebeck. Linear level lasserre lower bounds for certain k-CSPs. In *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 593–602, 2008.
- 26 S. Vadhan. *Pseudorandomness*, volume 7:1–3 of *Foundations and Trends in Theoretical Computer Science*. Now Foundations and Trends, December 2012.
- 27 V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

A Proof of Lemma 2.1

Proof of Lemma 2.1. Let $\mathbb{A} \in \mathcal{C}_n$ and $\mathbb{B} \in \mathcal{D}_n$ be two structures. Then, since $\Theta(\mathbb{A})$ and $\Theta(\mathbb{B})$ have size at most $p(n)$, there is a formula $\phi \in C^{k(p(n))}$ such that $\Theta(\mathbb{A}) \models \phi$ and $\Theta(\mathbb{B}) \not\models \phi$. We compose ϕ with the interpretation Θ to obtain ϕ' . That is to say, we replace every relation symbol by its defining formula, including replacing all occurrences of equality by ε , and we relativize all quantifiers to δ . Note that this involves replacing quantification over elements with quantification over tuples. It is well known that a counting quantifier over tuples $\exists^i \bar{x}$ can be replaced by a series of counting quantifiers over single elements without increasing the total number of variables. Then $\mathbb{A} \models \phi'$ and $\mathbb{B} \not\models \phi'$. It is also easy to check that ϕ' has at most $dk(p(n)) + t$ variables. The multiplicative factor d comes from the fact that every variable in ϕ is replaced by a d -tuple and the additive t accounts for any other variables that may appear in the formulas of Θ . ◀

B Proofs Omitted from Section 3.2

Proof of Lemma 3.1. For proving 1, let $h : \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ be an assignment of values to the variables of I that satisfies at least cm of the m equations in I . Define the assignment g on the variables of $G(I)$ by $g(x^a) = g(x) + a$. For each equation e satisfied by h , all eight equations arising from e are satisfied by g and so g satisfies at least $8cm$ of the $8m$ equations in $G(I)$.

For proving 2, suppose g is an assignment of values in $\{0, 1\}$ to the variables x_i^a in $G(I)$. Let $h : \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ be the assignment defined by $h(x_j) = g(x_j^0)$. We claim that if e_i is an equation $x_j + x_k + x_l = b$ in I that is not satisfied by h then at least four of the eight equations in $G(I)$ arising from e_i are falsified by g . To see this, consider two cases. First, suppose that $g(x_t^0) = g(x_t^1)$ for some $t \in \{j, k, l\}$. Without loss of generality, we assume $t = j$. Then consider the four *pairs* of equations

$$\begin{aligned} x_j^0 + x_j^{a_1} + x_k^{a_2} &= b_i + a_1 + a_2, \\ x_j^1 + x_j^{a_1} + x_k^{a_2} &= b_i + a_1 + a_2 + 1 \end{aligned}$$

obtained by taking the four possible values of a_1 and a_2 . Since $g(x_j^0) = g(x_j^1)$, if one equation in a pair is satisfied by g the other is necessarily falsified. Thus, at least four equations are falsified. For the second case, suppose that for each $t \in \{j, k, l\}$ occurring in e_i we have $g(x_t^0) \neq g(x_t^1)$. But then, since we assume that h falsifies e_i , it follows that g falsifies $x_j^0 + x_k^0 + x_l^0 = b$ and hence it falsifies all eight equations arising from e_i . In either case, g falsifies at least four of the equations arising from e_i .

Now, suppose that g satisfies at least $(1/2 + s/2) \cdot 8m$ of the $8m$ equations in $G(I)$. We claim that h satisfies at least sm equations in I . Suppose for contradiction that h falsifies a proportion $\epsilon > 1 - s$ of the equations. By the above argument, then g falsifies at least $4\epsilon m$ of the equations in $G(I)$. But $4\epsilon m > (1/2 - s/2) \cdot 8m$ contradicting the assumption that g satisfies at least $(1/2 + s/2) \cdot 8m$ equations. ◀

Proof of Lemma 3.2. We describe a strategy for Duplicator in the bijective k -pebble game played on $G(I)$ and $G(I^0)$, given a strategy in the existential k -pebble game on I and $\Gamma = \Gamma_{3\text{XOR}}$.

Suppose we have a position in the existential k -pebble game on I and Γ with pebbles on $x_1, \dots, x_{k'}$, for some $k' \leq k$ in I , and corresponding pebbles on $v_1, \dots, v_{k'} \in \{0, 1\}$ in Γ . Suppose further that this is a winning position for Duplicator, i.e. she has a strategy to play

forever from this position. Then, we claim that the position in the bijective game where the pebbles in $G(I)$ are on $x_1^{a_1}, \dots, x_{k'}^{a_{k'}}$, for some $a_1, \dots, a_{k'} \in \{0, 1\}$ and the matching pebbles in $G(I^0)$ are on $x_1^{a_1+v_1}, \dots, x_{k'}^{a_{k'}+v_{k'}}$ is a winning position in the bijective game on these two structures. To see this, note first that, if $x_r + x_s + x_t = b_i$ is an equation in I , for $1 \leq r, s, t \leq k'$, then by assumption that the position is winning in the existential game, $v_r + v_s + v_t = b_i$. Hence, $x_r^{a_r} + x_s^{a_s} + x_t^{a_t} = b_i$ is an equation in $G(I)$ if, and only if, $x_r^{a_r} + x_s^{a_s} + x_t^{a_t} = 0$ is an equation in $G(I^0)$ if, and only if, $x_r^{a_r+v_r} + x_s^{a_s+v_s} + x_t^{a_t+v_t} = v_r + v_s + v_t$ is an equation in $G(I^0)$, but this last equation is $x_r^{a_r+v_r} + x_s^{a_s+v_s} + x_t^{a_t+v_t} = b_i$. Thus, the map from $x_1^{a_1}, \dots, x_{k'}^{a_{k'}}$ to $x_1^{a_1+v_1}, \dots, x_{k'}^{a_{k'}+v_{k'}}$ is a partial isomorphism. To see that Duplicator can maintain the condition, suppose Spoiler moves the pebble on x_j^a . By assumption, Duplicator has a response in the existential game whenever Spoiler moves the pebble from x_j to x_l . This response defines a function f from the variables in x to $\{0, 1\}$. We use this to define the bijection taking x_l^a to $x_l^{a+f(x_l)}$. This is a winning move in the bijective game. ◀

C Deriving Theorem 4.3 from [16]

The proof of Theorem 4.3 follows from Lemmas 5.1 and 5.2 in [16]. In order to see this, we need to explain how our notation matches the one in [16]. Besides the obvious and minor correspondance between multiplicative and additive notation for \mathbb{F}_2 , with $-1 \leftrightarrow 1$ and $+1 \leftrightarrow 0$, there are three other noticeable differences between the statement of Theorem 4.3 and the statements of Lemmas 5.1 and 5.2 in [16].

The first difference is that Theorem 4.3 applies to arbitrary projection game instances of LABEL COVER, while the statements in [16] are phrased only for the special cases of the problem that result from applying parallel repetition to a suitable bipartite reduction applied to a 3SAT instance. We chose to formulate Theorem 4.3 in this more general and modular form because this is what the proofs of Lemmas 5.1 and 5.2 in [16] show, and also because this is how more recent expositions of these results are presented (see, e.g., [3]).

The second difference is that the conclusion of our statement is phrased in terms of the c -satisfiability of a 3XOR instance, while the statements of Lemmas 5.1 and 5.2 in [16] are phrased in terms of the acceptance rate of a probabilistic test that has the following form: given access to certain tables A_u and A_v , with \mathbb{F}_2 entries $\{A_u(x)\}_{x \in I}$ and $\{A_v(y)\}_{y \in J}$ for certain index sets I and J , respectively, choose a random 3-variables parity test on the $A_u(x)$ and $A_v(y)$ entries under a specially designed distribution, and check if it is satisfied. This difference is only notational and minor: our instance of XOR is built by viewing the $A_u(x)$ and $A_v(y)$ entries as variables $u(x)$ and $v(y)$, and assigning weight to each 3-variable parity equation on these variables proportionally to the probability that it is checked by the probabilistic test on the A_u and A_v tables. With this change, c -satisfiability of the instance translates into the probability of acceptance of the test being at least c , and vice-versa.

The third difference in the notation is that our variables $u(x)$ and $v(y)$, and the corresponding entries $A_u(x)$ and $A_v(y)$ of the tables A_u and A_v , are indexed by \mathbb{F}_2^{p-1} and \mathbb{F}_2^{q-1} instead of the more natural \mathbb{F}_2^p and \mathbb{F}_2^q , respectively. This is due to the fact that we implement the operations of *folding over true* and *conditioning upon h* from [16] directly in our construction. In other words, our tables A_u and A_v are what [16] calls $A_{W,h,true}$ and $A_{U,true}$, respectively. Folding over true as in $A_{U,true}$ is achieved for A_v through the notation $S(z)$ and $F(z)$ defined above: we chose to partition \mathbb{F}_2^p into 2^{p-1} pairs of the form $(z, 0), (F((z, 1)), 1)$, as z ranges over \mathbb{F}_2^{p-1} , and view an arbitrary $A_v : \mathbb{F}_2^{p-1} \rightarrow \mathbb{F}_2$ as representing the function $A'_v : \mathbb{F}_2^p \rightarrow \mathbb{F}_2$ defined by $A'_v(z) = A_v(F(z)) + S(z)$ for every $z \in \mathbb{F}_2^p$. It is straightforward to see that A'_v is folded over true, in the definition of [16], by construction.

Conditioning upon h as in $A_{W,h,true}$ for A_u is achieved through the same mechanism as folding over true with the additional observation that the operation of conditioning upon h is necessary only if the instance of LABEL COVER fails to satisfy the property that for every $(u, v) \in U \times V$ and every $a \in A$ there is at least one $b \in B$ that satisfies the predicate $P(u, v, a, b)$. When this is the case, one defines $h = h_{u,v} : A \rightarrow \{0, 1\}$ as the predicate indicating if a given a has at least one b that satisfies $P(u, v, a, b)$, and *conditions the table A_u upon h* . In our case we do not require this since the given instance of LABEL COVER is a projection game instance, and, in particular, for every a there is exactly one b , and hence at least one b , such that $P(u, v, a, b) = 1$; i.e., $h = h_{u,v}$ is the constant 1 predicate. It should be added that the reason why we can assume that I is a projection game instance is that our bipartite reduction is designed in such a way that the values a in A are partial assignments that always satisfy the corresponding constraints u in U . In contrast, in [16] the values are taken as arbitrary truth assignments to the variables of a collection of clauses, and not all such assignments satisfy all the clauses. Our exposition is again more modular and also matches more recent expositions of the results in [16] (again, see, e.g., [3]).

With this notational correspondance, it is now easy to see that Lemma 5.1 in [16] gives the first claim in Theorem 4.3, and Lemma 5.2 in [16] applied with $\delta = (s/\epsilon)^{1/2}/4$ gives the second claim in Theorem 4.3.

D Proof of Theorem 5.5

The proof of Theorem 5.5 proceeds through a series of lemmas.

► **Lemma D.1.** *If G is a d -regular graph on n vertices, for any $d \geq 1$, then $\text{vc}(G) \geq n/2$.*

Proof. Let S be any set of vertices in G . Then the number of edges incident on vertices in S is at most $d|S|$. Since the number of edges in G is $dn/2$, if S is a vertex cover $d|S| \geq dn/2$ and so $|S| \geq n/2$. ◀

Let G be a graph and C_1, \dots, C_m be the partition of the vertices of G given by *vertex refinement*. So, there are constants δ_{ij} such that each $v \in C_i$ has exactly δ_{ij} neighbours in C_j . Since the graph is undirected, the number of edges from C_i to C_j is the same as in the other direction and so $\delta_{ij}|C_i| = \delta_{ji}|C_j|$, for all i and j . Also, $\delta_{ij} = 0$ if, and only if, $\delta_{ji} = 0$.

Let $X = \{i \mid \delta_{ii} = 0\}$ and $Y = \{i \mid \delta_{ii} > 0\}$. Consider the undirected graph X_G with vertices X and edges $\{(i, j) \mid \delta_{ij} > 0\}$. Consider the instance (X_G, w) of *weighted vertex cover* obtained by taking the graph X_G and giving each vertex i the weight $w(i) = |C_i|$. Let p_G denote the value of the minimum weighted vertex cover of this instance. Also, let $q_G = \sum_{i \in Y} |C_i|$. Finally, define $v_G = p_G + q_G$.

► **Lemma D.2.** *If $G \equiv_{C^2} H$ then $v_G = v_H$.*

Proof. The value v_G is determined entirely by the sizes of C_i in the vertex refinement of G and the corresponding values of δ_{ij} . Since $G \equiv_{C^2} H$, these values are the same for H . ◀

► **Lemma D.3.** $\text{vc}(G) \leq v_G$.

Proof. Let $Z \subseteq X$ be a minimum-weight vertex cover in (X_G, w) . Take the set $S \subseteq V(G)$ defined by $S = \bigcup_{i \in Y \cup Z} C_i$. Note that the sets Y and Z are disjoint, $\sum_{i \in Y} |C_i| = q_G$ by definition, and $\sum_{i \in Z} |C_i| = p_G$ by construction. So S has exactly v_G vertices. We claim that S is a vertex cover in G . Let e be any edge of G with endpoints in C_i and C_j . If either i or

j is in Y , then the corresponding endpoint of e is in S since $C_i \subseteq S$ for all $i \in Y$. If both i and j are not in Y then both are in X and $\delta_{ij} > 0$. Thus, since Z is a vertex cover for the graph X_G then one of i or j must be in Z and again at least one endpoint of e is in S . ◀

For the proof of the next lemma, we need the notion of a *fractional vertex cover* of a graph $G = (V, E)$. This is a function $f : V \rightarrow [0, 1]$ satisfying the condition that for every $(u, v) \in E$, $f(u) + f(v) \geq 1$. It is known that if f is a fractional vertex cover of G , then $\sum_{v \in V} f(v) \geq \text{vc}(G)/2$ (see [27, Thm. 14.2]). More generally, suppose we have an instance of *weighted vertex cover*, i.e. G along with a weight function $w : V \rightarrow \mathbb{N}$ where $\text{vc}(G, w)$ is defined as the value of the minimum weighted vertex cover. Then $\sum_{v \in V} f(v)w(v) \geq \text{vc}(G, w)/2$.

► **Lemma D.4.** $v_G \leq 2\text{vc}(G)$.

Proof. Let S be any vertex cover of G . Let $U_X = \bigcup_{i \in X} C_i$ and $U_Y = \bigcup_{i \in Y} C_i$ and note that these sets are disjoint. We claim that $|S \cap U_X| \geq p_G/2$ and $|S \cap U_Y| \geq q_G/2$, and therefore $|S| = |S \cap U_X| + |S \cap U_Y| \geq v_G/2$, establishing the result.

First, consider $S \cap U_Y$. Note that for any $i \in Y$, the subgraph of G induced by C_i is δ_{ii} -regular. Since $\delta_{ii} > 0$ by definition of Y , by Lemma D.1 we have $|S \cap C_i| \geq |C_i|/2$ and therefore $|S \cap U_Y| \geq q_G/2$.

Secondly, consider the function $f : X \rightarrow [0, 1]$ defined by $f(i) = |S \cap C_i|/|C_i|$. We claim that this is a fractional vertex cover of the graph X_G . To verify this, we need to check that $f(i) + f(j) \geq 1$ whenever $\delta_{ij} > 0$. There are $\delta_{ij}|C_i|$ edges between C_i and C_j . Each element of $S \cap C_i$ can cover at most δ_{ij} of these edges and similarly each element of $S \cap C_j$ covers at most δ_{ji} of them. Thus, since S is a vertex cover $|S \cap C_i|\delta_{ij} + |S \cap C_j|\delta_{ji} \geq \delta_{ij}|C_i|$. Substituting for δ_{ji} using the identity $\delta_{ij}|C_i| = \delta_{ji}|C_j|$ gives $|S \cap C_i|\delta_{ij} + |S \cap C_j|\delta_{ij}|C_i|/|C_j| \geq \delta_{ij}|C_i|$. Now dividing through by $\delta_{ij}|C_i|$ gives $f(i) + f(j) \geq 1$.

Thus, we have that the weighted vertex cover instance (X_g, w) admits the fractional solution f whose total weight is

$$\sum_{i \in X} f(i)|C_i| = \sum_{i \in X} |S \cap C_i| = |S \cap U_X|.$$

Since p_G is the value of the minimum weight vertex cover of (X_g, w) , we have $|S \cap U_X| \geq p_G/2$, as was to be shown. ◀

Proof of Theorem 5.5. Suppose for contradiction that there is a $G \in \mathcal{C}$ and $H \in \mathcal{D}$ such that $G \equiv_{C^2} H$. Since G and H must have the same number of vertices, we have $2\text{vc}(G) < \text{vc}(H)$. But, by Lemma D.4 we have $v_G \leq 2\text{vc}(G)$, by Lemma D.3 we have $\text{vc}(H) \leq v_H$ and by Lemma D.2 we have $v_G = v_H$, giving a contradiction. ◀

Safety, Absoluteness, and Computability

Arnon Avron

School of Computer Science, Tel Aviv University
Tel Aviv, Israel
aa@post.tau.ac.il

Shahar Lev

School of Computer Science, Tel Aviv University
Tel Aviv, Israel
shaharle@post.tau.ac.il

Nissan Levi

School of Computer Science, Tel Aviv University
Tel Aviv, Israel
nisis.levi@gmail.com

Abstract

The semantic notion of dependent safety is a common generalization of the notion of absoluteness used in set theory and the notion of domain independence used in database theory for characterizing safe queries. This notion has been used in previous works to provide a unified theory of constructions and operations as they are used in different branches of mathematics and computer science, including set theory, computability theory, and database theory. In this paper we provide a complete syntactic characterization of general first-order dependent safety. We also show that this syntactic safety relation can be used for characterizing the set of strictly decidable relations on the natural numbers, as well as for characterizing rudimentary set theory and absoluteness of formulas within it.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Dependent Safety, Computability, Absoluteness, Decidability, Domain Independence

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.8

1 Introduction

The semantic notion of dependent safety is a common generalization of the notion of absoluteness used in set theory ([14, 9]) and the notion of domain independence used in database theory for characterizing safe queries ([1, 20]). It has been introduced in [3] and used there to provide a unified theory of constructions and operations as they are used in different branches of mathematics and computer science, including set theory, computability theory, and database theory. The notion is based on the following two basic ideas (taken from logic programming and database theory):

- From an abstract logical point of view, the focus of a general theory of computations should be on functions of the form:

$$\lambda y_1, \dots, y_k. \{ \langle x_1, \dots, x_n \rangle \in S^n \mid S \models \varphi(x_1, \dots, x_n, y_1, \dots, y_k) \}$$

where S is a structure for some first-order signature σ , φ is some formula of σ , and $\{ \{x_1, \dots, x_n\}, \{y_1, \dots, y_k\} \}$ is a partition of the set $Fv(\varphi)$ of the free variables of φ . Here the tuple $\langle y_1, \dots, y_k \rangle$ provides the input, while the output is the set of answers to the resulting query.



© Arnon Avron, Shahar Lev, and Nissan Levi;
licensed under Creative Commons License CC-BY
27th EACSL Annual Conference on Computer Science Logic (CSL 2018).
Editors: Dan Ghica and Achim Jung; Article No. 8; pp. 8:1–8:17



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

- An allowable query should be *safe* in the sense that the answer to it does not depend on the exact domain of S , but only on the values of the parameters $\{y_1, \dots, y_k\}$ and the part of S which is relevant to them and to the query, under certain conditions concerning the language and the structures that are taken as relevant to the query.

Examples.

1. In every computerized system, what is taken as the type of natural numbers is actually only some finite initial segment of the full set of natural numbers. Therefore a reasonable query should be one that has the same answer in all implementations in which this initial segment includes the inputs to the query and the natural numbers mentioned in it.
2. Query languages for database theory allow only *domain independent* queries, that is: queries for which the corresponding answer would be the same in all databases which have the same scheme and exactly the same tables for it.

The above two principles were translated in [3, 4, 5, 6] into precise definitions. Those works (especially [3]) also naturally lead to the following two theses concerning the development of a general theory of decidability and computability in arbitrary structures:

- The study of decidability of relations should be a part of a more general study of *absoluteness* of formulas and queries;
- The study of computability/constructibility should be a part of a more general study of *dependent safety* of formulas and queries. (We call this type of safety ‘dependent’ because it is a property of queries which might contain parameters.)

A significant step in this program of developing general theory of dependent safety, absoluteness, and computability was made in [3, 5], where a syntactic framework for these notions was developed. The main virtues of that framework are its generality and universality: it is based on few basic simple syntactic principles, that can be used in what seem to be very different and unrelated areas. The main result of this paper is that it is actually *complete* for general first-order dependent safety and general first-order absoluteness. This explains its generality, and why its principles were independently discovered in different areas.¹

With the exception of the relatively simple case of databases, the above mentioned general syntactic principles may of course be insufficient in more complex *particular* cases. Still, we show that they suffice also in the case of the arithmetics of the natural numbers, while in the especially important case of set theory our syntactic characterization of absoluteness is equivalent to the usual syntactic approximation that is currently in use.

The structure of the rest of this paper is as follows. Section 2 we review (an improvement of) the framework developed in [3], including all the necessary definitions. In Section 3 we prove the completeness of our syntactic approximation of general first-order dependent safety, while in Section 4 we provide a direct syntactic approximation of general first-order absoluteness. (The latter is a very important special case of the former.) In Section 5 we give a syntactic characterization of absoluteness in the structure \mathcal{N} of the natural numbers. Finally, in Section 6 we study absoluteness in rudimentary set theory, using a language that includes abstract set terms. We show that while the use of such terms involves a proper extension of our syntactic dependent safety, this is not true for syntactic absoluteness.

¹ The principles were originally identified as generalizations of principles used in database theory. As far as we know, this is a rare case in which ideas and principles originally taken from computer science are applied for understanding purely mathematical theories like set theories and number theory.

2 Preliminaries

Throughout this paper, σ is a first-order signature with equality, and *no function symbols* (except for constants). $Fv(\varphi)$ and $Bv(\varphi)$ respectively denote the set of free variables and the set of bound variables of φ . The notation $\varphi(z_1, \dots, z_k)$ means that $Fv(\varphi) = \{z_1, \dots, z_k\}$.

2.1 Basic Definitions

► **Definition 1.** Let S_1 and S_2 be two structures for σ . S_1 is a *weak substructure* of S_2 (notation: $S_1 \subseteq_\sigma S_2$) if the domain of S_1 is a subset of the domain of S_2 , and the interpretations in S_1 and S_2 of the constants of σ are identical.

► **Definition 2.** Let $S_1 \subseteq_\sigma S_2$, where S_1 and S_2 are structures for σ . A formula of σ $\varphi(x_1, \dots, x_n, y_1, \dots, y_m)$ is *safe* for S_1 and S_2 with respect to $\{x_1, \dots, x_n\}$ (notation: $\varphi \succ^{S_1; S_2} \{x_1, \dots, x_n\}$), if for all $b_1, \dots, b_m \in S_1$:

$$\{\vec{a} \in S_2^n \mid S_2 \models \varphi(\vec{a}, \vec{b})\} = \{\vec{a} \in S_1^n \mid S_1 \models \varphi(\vec{a}, \vec{b})\}$$

In other words, φ is safe for S_1 and S_2 with respect to $\{x_1, \dots, x_n\}$ if by viewing y_1, \dots, y_m as parameters, and assigning elements from S_1 to these parameters, we get a query in x_1, \dots, x_n having the same answers in S_1 and S_2 .

► **Definition 3.** A *safety-signature* is a pair (σ, F) , where σ is an ordinary first-order signature with equality and no function symbols, and F is a function which assigns to every n -ary predicate symbol of σ a subset of the powerset of $\{1, \dots, n\}$, so that $F(=)$ is $\{\{1\}, \{2\}\}$.

► **Definition 4.** Let (σ, F) be a safety-signature, and let S_1, S_2 be structures for σ . S_2 is called a (σ, F) -*extension* of S_1 (and S_1 is a (σ, F) -*substructure* of S_2) if $S_1 \subseteq_\sigma S_2$ and $p(x_1, \dots, x_n) \succ^{S_1; S_2} \{x_{i_1}, \dots, x_{i_k}\}$ whenever p is an n -ary predicate of σ , x_1, \dots, x_n are n distinct variables, and $\{i_1, \dots, i_k\} \in F(p)$.

► **Definition 5.** Let (σ, F) be a safety-signature, S a structure for σ , and φ a formula of σ .

1. φ is (S, F) -*safe w.r.t.* X (notation: $\varphi \succ_{(S, F)} X$) if $\varphi \succ^{S'; S} X$ whenever S' is a (σ, F) -extension of S . φ is (S, F) -*absolute* if $\varphi \succ_{(S, F)} \emptyset$.
2. φ is (σ, F) -*safe w.r.t.* X ($\varphi \succ_{(\sigma, F)} X$) if it is (S, F) -safe w.r.t. X for every structure S for σ . φ is (σ, F) -*absolute* if $\varphi \succ_{(\sigma, F)} \emptyset$.

► **Note 6.** The reason that we have demanded $F(=)$ to be $\{\{1\}, \{2\}\}$ (or $\{\{1\}, \{2\}, \emptyset\}$, which is equivalent) is that $x_1 = x_2$ is always safe w.r.t. both $\{x_1\}$ and $\{x_2\}$, but usually not w.r.t. $\{x_1, x_2\}$.

► **Note 7.** If $\varphi \succ_{(\sigma, F)} X$ and $Z \subseteq X$, then $\varphi \succ_{(\sigma, F)} Z$. In particular: if $\varphi \succ_{(\sigma, F)} X$ for some X then φ is (σ, F) -absolute. The same applies to (S, F) -safety and to (S, F) -absoluteness.

► **Note 8.** If $F(p)$ is nonempty for every p in σ , then by Note 7 S_1 is a substructure of S_2 (in the usual sense of model theory) whenever S_2 is a (σ, F) -extension of S_1 .

2.2 Examples

2.2.1 Computability Theory

Several applications of dependent safety to the theory of computability and decidability have been made in [3]. Here is one of them.

Define the safety-signature $(\sigma_{\mathcal{N}}, F_{\mathcal{N}})$ as follows:

- $\sigma_{\mathcal{N}}$ is the first-order signature which includes the constant 0, the binary predicate \leq , and the ternary relations P_+ , P_\times .
- $F_{\mathcal{N}}(\leq) = \{\{1\}\}$, $F_{\mathcal{N}}(P_+) = F_{\mathcal{N}}(P_\times) = \{\emptyset\}$.

The standard structure \mathcal{N} for $\sigma_{\mathcal{N}}$ has the set N of natural numbers as its domain, with the usual interpretations of 0 and \leq , and the (graphs of the) operations $+$ and \times on N (viewed as ternary relations on N) as the interpretations of P_+ and P_\times , respectively. It is easy to see that \mathcal{N} is a $(\sigma_{\mathcal{N}}, F_{\mathcal{N}})$ -extension of a structure S for $\sigma_{\mathcal{N}}$ iff the domain of S is an initial segment of \mathcal{N} (where the interpretations of the relation symbols are the corresponding reductions of the interpretations of those symbols in \mathcal{N}). Thus $\varphi \succ_{(\mathcal{N}, F_{\mathcal{N}})} X$ iff the query $\{\langle x_1, \dots, x_n \rangle \in S^n \mid S \models \varphi(x_1, \dots, x_n, y_1, \dots, y_k)\}$ is “reasonable” in the sense explained in example 1 above (where $X = \{x_1, \dots, x_n\}$). Using this observation, it was proved in [3] that a relation R on N is recursively enumerable iff R is definable by a formula of the form $\exists y_1, \dots, y_n \psi$, where the formula ψ is $(\sigma_{\mathcal{N}}, F_{\mathcal{N}})$ -absolute.

2.2.2 Set Theory

Let $\sigma_{ZF} = \{\in\}$, $F_{ZF}(\in) = \{\{1\}\}$. A structure S_2 for σ_{ZF} is a (σ_{ZF}, F_{ZF}) -extension of S_1 iff S_2 is an extension of S_1 , and $x_1 \in x_2 \succ_{S_1; S_2} \{x_1\}$. The latter condition means that S_1 is a transitive substructure of S_2 . Therefore $\varphi \succ_{(\sigma_{ZF}, F_{ZF})} \emptyset$ iff the following holds whenever S_1 is a transitive substructure of S_2 : $S_1 \models \varphi \Leftrightarrow S_2 \models \varphi$. Hence a formula is (σ_{ZF}, F_{ZF}) -absolute iff it is absolute in the usual sense of set theory. (See e.g. [14].)

Other applications to set theories of dependent safety in general, and of $\succ_{(\sigma_{ZF}, F_{ZF})}$ in particular, have been made in [5] and [4]. In [5] it is suggested that an abstract set term $\{x \mid \varphi\}$ denotes a predicatively acceptable set if $\varphi \succ_{(\sigma_{ZF}, F_{ZF})} \{x\}$. In [4] the relation $\succ_{(\sigma_{ZF}, F_{ZF})}$ is used as the basis for purely logical characterizations of the comprehension schemas allowed in various set theories (including ZF).

2.2.3 Databases

From a logical point of view, a database of scheme $D = \{P_1, \dots, P_n\}$ is just a given set of *finite* interpretations of P_1, \dots, P_n . A corresponding query language is usually an ordinary first-order language which is based on a signature σ with equality such that σ contains D , but no function symbols. A query is called *domain independent* ([1, 20]) if its answer is the same in all interpretations in which P_1, \dots, P_n are given by the database, while the interpretations of all other predicate symbols (like $<$ or \leq) and of the constants are absolute (and externally given). It can easily be seen that a formula φ is domain independent iff $\varphi \succ_{(\sigma, F)} Fv(\varphi)$ for the function F defined by: $F(Q) = \{\{1, \dots, n_Q\}\}$ in case $Q \in \{P_1, \dots, P_n\}$ (where n_Q is the arity of Q), while $F(Q) = \{\emptyset\}$ otherwise.

2.2.4 Querying the Web

In [15] the web is modeled as an ordinary database augmented with three more special relations (together with some other, which for simplicity we ignore): $N(id, title, \dots)$, $C(node, value)$, $L(source, destination, \dots)$. The intuitive interpretations of these relations are the following:

- The relation N contains the Web objects which are identified by a Uniform Resource Locator (URL). id represents the URL and is a key.
- The meaning of C is that the string which is represented by its second argument occurs within the body of the document in the URL which is represented by its first argument.
- The relation L holds between nodes $source$ and $destination$ if there is a hypertext link from the first to the second.

The question investigated in [15] is: what queries should be taken as safe, if we assume that what is practically possible in the case of N and L is to list all their tuples which *correspond to a given first argument*, while C is only assumed to be decidable. It is not difficult to see that the notion of safety given there for this framework is equivalent to (σ_{web}, F_{web}) -safe in our sense, where $\{L, N, C\} \subseteq \sigma_{web}$, and F is defined like in ordinary databases, except that $F(L) = \{2, \dots, m\}$ (where m is the arity of L), $F(N) = \{2, \dots, k\}$ (where k is the arity of N), and $F(C) = \{\emptyset\}$.

2.3 The Corresponding Syntactic Relation

In [10] it was proved that the property of domain independence in databases is undecidable. In [3] it was shown that the property of (σ, F) -absoluteness is also in general undecidable. This means that in order to use the relation $\succ_{(\sigma, F)}$ in practice we need a decidable syntactic approximation. The one that was used in [3, 4, 5] is presented in the next definition. It was inspired by the recursive definition of syntactic safety given in [20], and generalizes it in a sense explained below.²

► **Definition 9.** Given a safety-signature (σ, F) , we recursively define the relation $\succ_{(\sigma, F)}$ between formulas of σ and sets of variables as follows:

1. $p(t_1, \dots, t_n) \succ_{(\sigma, F)} X$ in case p is an n -ary predicate symbol of σ , and there is $I \in F(p)$ such that:
 - a. For every $x \in X$ there is $i \in I$ such that $x = t_i$.
 - b. $X \cap Fv(t_j) = \emptyset$ for every $j \in \{1, \dots, n\} \setminus I$.
2. $\neg\varphi \succ_{(\sigma, F)} \emptyset$ if $\varphi \succ_{(\sigma, F)} \emptyset$.
3. $\varphi \vee \psi \succ_{(\sigma, F)} X$ if $\varphi \succ_{(\sigma, F)} X$ and $\psi \succ_{(\sigma, F)} X$
4. $\varphi \wedge \psi \succ_{(\sigma, F)} X \cup Y$ if $\varphi \succ_{(\sigma, F)} X$, $\psi \succ_{(\sigma, F)} Y$, and either $Fv(\varphi) \cap Y = \emptyset$ or $Fv(\psi) \cap X = \emptyset$.
5. $\exists y.\varphi \succ_{(\sigma, F)} X \setminus \{y\}$ if $y \in X$ and $\varphi \succ_{(\sigma, F)} X$.

► **Theorem 10 ([3]).** $\succ_{(\sigma, F)}$ is sound: if $\varphi \succ_{(\sigma, F)} \{x_1, \dots, x_n\}$ then $\varphi \succ_{(\sigma, F)} \{x_1, \dots, x_n\}$

► **Note 11.** In what follows $\forall x_1 \dots \forall x_n.\varphi \rightarrow \psi$ as an abbreviation for $\neg\exists x_1 \dots \exists x_n.\varphi \wedge \neg\psi$. Using items 2,4, and 5 from Definition 9, this implies that $\forall x_1 \dots \forall x_n.\varphi \rightarrow \psi \succ_{(\sigma, F)} \emptyset$ if $\varphi \succ_{(\sigma, F)} \{x_1, \dots, x_n\}$ and $\psi \succ_{(\sigma, F)} \emptyset$. We shall use this fact freely.

► **Note 12.** It follows from Definition 9 and the fact that $F(=)$ is $\{\{1\}, \{2\}\}$ that $x = t \succ_{(\sigma, F)} \{x\}$ and $t = x \succ_{(\sigma, F)} \{x\}$ in case $x \notin Fv(t)$, and $t = s \succ_{(\sigma, F)} \emptyset$ for every t, s .

Examples.

1. The set of formulas φ such that $\varphi \succ_{(\sigma_N, F_N)} \emptyset$ includes all formulas in the well-known set of arithmetical Δ_0 -formulas (also called “bounded formulas” or “ σ_0 -formulas” in [17]). In the context of σ_N these are the formulas in which all quantifications are of the form $\exists x \leq y$ (or $\forall x \leq y$, by Note 11), where x and y are distinct variables.
2. Similarly, the set of formulas φ such that $\varphi \succ_{(\sigma_{ZF}, F_{ZF})} \emptyset$ is an extension of the set of set-theoretical Δ_0 formulas ([14]).³ However, in this case not only this special case of syntactic dependent safety is important. In fact, if $\varphi(x_1, \dots, x_n, y_1, \dots, y_k) \succ_{(\sigma_{ZF}, F_{ZF})} \{x_1, \dots, x_n\}$ then the function $\lambda y_1, \dots, y_k. \langle x_1, \dots, x_n \mid \varphi \rangle$ is rudimentary. (Rudimentary functions

² Other closely related works in database theory are e.g. [16], [19], and [18].

³ In the context of σ_{ZF} Δ_0 -formulas (again also called “bounded formulas”) are the formulas in which all quantifications are of the form $\exists x \in y$ (or $\forall x \in y$, by Note 11), where x and y are variables.

were independently introduced by Gandy in [12] and by Jensen in [13]. See also [9].) In particular: if $\varphi(x_1, \dots, x_n, y_1, \dots, y_k) \succ_{(\sigma_{ZF}, F_{ZF})}^s \{x_1, \dots, x_n\}$ then the function $\lambda y_1 \in \mathcal{HF}, \dots, y_k \in \mathcal{HF}. \{x_1, \dots, x_n\} \in \mathcal{HF}^n \mid \mathcal{HF} \models \varphi(x_1, \dots, x_n, y_1, \dots, y_k)\}$ (where \mathcal{HF} is the set of hereditarily finite sets) is a computable function from \mathcal{HF}^k to \mathcal{HF} . (We shall return to this example in Section 6.)

3. Let D , σ , and F be like in Section 2.2.3. Then $\varphi \succ_{(\sigma, F)}^s Fv(\varphi)$ for any formula φ which is syntactically safe according to the definition in [20].
4. $\varphi \succ_{(\sigma_{web}, F_{web})}^s Fv(\varphi)$ for any formula φ which is safe according to the ‘‘Safe Web Calculus’’ given in [15] as a syntactic approximation for the class of (σ_{web}, F_{web}) -safe formulas.

► **Note 13.** It is easy to see that if $\varphi \succ_{(\sigma, F)}^s X$ and $Y \subseteq X$ then $\varphi \succ_{(\sigma, F)}^s Y$. In particular, if $\varphi \succ_{(\sigma, F)}^s X$ then $\varphi \succ_{(\sigma, F)}^s \emptyset$.

3 The General Completeness Theorem

Our main goal in this section is to prove an appropriate converse to Theorem 10.

► **Notation 14.** $\varphi \equiv \psi$ if φ and ψ are logically equivalent, and $Fv(\varphi) = Fv(\psi)$.

► **Lemma 15.** *Let (σ, F) be a safety-signature. Let φ and ψ be two formulas of σ such that $Y \subseteq Fv(\varphi) \cap Fv(\psi)$. If φ and ψ are logically equivalent, then $\varphi \succ_{(\sigma, F)} Y$ iff $\psi \succ_{(\sigma, F)} Y$. In particular: if $\varphi \equiv \psi$ then for every Y it holds that $\varphi \succ_{(\sigma, F)} Y$ iff $\psi \succ_{(\sigma, F)} Y$.*

Proof. Immediate from the definitions. ◀

► **Theorem 16.** *Let (σ, F) be a safety-signature such that σ includes a constant. Then for every φ and Y , $\varphi \succ_{(\sigma, F)} Y$ iff there exists ψ such that $\psi \succ_{(\sigma, F)}^s Y$ and $\varphi \equiv \psi$.*

Proof. We begin with some notations. If S is a structure for σ , and v is an assignment in S , then $S, v \models \varphi$ denotes that φ is satisfied in S by the assignment v . $T \vdash^t \varphi$ denotes that $S, v \models \varphi$ whenever $S, v \models \psi$ for every $\psi \in T$. If $\bar{x} = \langle x_1, \dots, x_m \rangle$ is a finite list of distinct variables, and $\bar{a} \in S^m$, then we denote by $\bar{x} := \bar{a}$ some assignment v in S such that $v(x_i) = a_i$ for every $1 \leq i \leq m$. If $Fv(\varphi) = \{x_1, \dots, x_m\}$ and $\bar{a} \in S^m$, then $S \models \varphi(\bar{a})$ means that $S, \bar{x} := \bar{a} \models \varphi$.

Let (σ, F) be a safety-signature.

► **Lemma 17.** *Let $(\hat{\sigma}, \hat{F})$ be the safety-signature such that $\hat{\sigma}$ is σ without the predicates p for which $F(p) = \emptyset$ and \hat{F} is the restriction of F to predicates of $\hat{\sigma}$. If $\varphi \succ_{(\sigma, F)} Y$ then there exists a formula $\hat{\varphi}$ of $\hat{\sigma}$ such that $\hat{\varphi} \succ_{(\hat{\sigma}, \hat{F})} Y$ and $\varphi \equiv \hat{\varphi}$.*

Proof. Let S_1 and S_2 be two structures for σ that have the same domain and the same interpretations for the constants of σ and the predicates of $\hat{\sigma}$. Then S_1 and S_2 are (σ, F) -substructures of one another, and so $S_1, v \models \varphi$ iff $S_2, v \models \varphi$ for every assignment v in their common domain. Therefore Beth definability theorem implies that there exists a formula $\hat{\varphi}$ of $\hat{\sigma}$ such that $\varphi \equiv \hat{\varphi}$. By Lemma 15, $\hat{\varphi} \succ_{(\sigma, F)} Y$ and so $\hat{\varphi} \succ_{(\hat{\sigma}, \hat{F})} Y$. ◀

► **Lemma 18.** *Let S_1, S_2, S_3 be structures for σ such that S_1 is a substructure of S_2 , S_2 is a substructure of S_3 , and S_1 is a (σ, F) -substructure of S_3 . Then S_1 is a (σ, F) -substructure of S_2 .*

Proof. Let p be a n -ary predicate of σ , and let $I \in F(p)$. Suppose that $a_1, \dots, a_n \in S_2$ and $a_i \in S_1$ for every $i \in \{1, \dots, n\} \setminus I$.

- Assume $S_2 \models p(\bar{a})$. Since S_2 is a substructure of S_3 then $S_3 \models p(\bar{a})$. Since S_1 is a (σ, F) -substructure of S_3 , $\bar{a} \in S_1^n$ and $S_1 \models p(\bar{a})$.
- Assume $\bar{a} \in S_1^n$ and $S_1 \models p(\bar{a})$. Since S_1 is a substructure of S_2 then $S_2 \models p(\bar{a})$. ◀

► **Lemma 19.** For a structure S for σ and $\bar{a} \in S^m$, let $\alpha_{(\sigma, F)}[S, \bar{a}]$ be the substructure of S whose domain is the set of all $b \in S$ for which there exists a formula $\theta(x_1, \dots, x_m, z)$ of σ (where x_1, \dots, x_m, z are $m+1$ distinct variables) such that $\theta(\bar{x}, z) \succ_{(\sigma, F)}^s \{z\}$ and $S \models \theta(\bar{a}, b)$. Then $\alpha_{(\sigma, F)}[S, \bar{a}]$ is a (σ, F) -substructure of S .

Proof. First note that $\alpha_{(\sigma, F)}[S, \bar{a}]$ is indeed a well-defined substructure of S . This follows from the facts that σ contains no function symbols, and that for every constant c in σ , the formula $c = z$ of σ satisfies $c = z \succ_{(\sigma, F)}^s \{z\}$, assuring that $\alpha_{(\sigma, F)}[S, \bar{a}]$ contains all the interpretations in S of the constants of σ . (In particular: $\alpha_{(\sigma, F)}[S, \bar{a}] \neq \emptyset$.)⁴

Now suppose that p is a n -ary predicate of σ , $I \in F(p)$, $\bar{b} \in S^n$, and $b_j \in \alpha_{(\sigma, F)}[S, \bar{a}]$ for every $j \in \{1, \dots, n\} \setminus I$.

- Assume $b_i \in \alpha_{(\sigma, F)}[S, \bar{a}]$ for every $i \in I$ and $\alpha_{(\sigma, F)}[S, \bar{a}] \models p(\bar{b})$. Since $\alpha_{(\sigma, F)}[S, \bar{a}]$ is a substructure of S , we get that $S \models p(\bar{b})$.
- Assume $S \models p(\bar{b})$. Let $x_1, \dots, x_m, y_1, \dots, y_n, z$ be $m+n+1$ distinct variables. Let $j \in \{1, \dots, n\} \setminus I$. Since we assume that $b_j \in \alpha_{(\sigma, F)}[S, \bar{a}]$, the definition of $\alpha_{(\sigma, F)}[S, \bar{a}]$ implies that there exists a formula $\theta_j(\bar{x}, y_j)$ of σ such that $\theta_j(\bar{x}, y_j) \succ_{(\sigma, F)}^s \{y_j\}$ and $S \models \theta_j(\bar{a}, b_j)$. Define the following formulas of σ :

$$\xi(\bar{x}, \bar{y}) : \left(\bigwedge_{j \in \{1, \dots, n\} \setminus I} \theta_j(\bar{x}, y_j) \right) \wedge p(\bar{y})$$

$$\mu_i(\bar{x}, z) : \exists \bar{y} [\xi(\bar{x}, \bar{y}) \wedge z = y_i] \quad (i \in I)$$

Now we know that:

$$\bigwedge_{j \in \{1, \dots, n\} \setminus I} \theta_j(\bar{x}, y_j) \succ_{(\sigma, F)}^s \{y_j \mid j \in \{1, \dots, n\} \setminus I\}$$

$$p(\bar{y}) \succ_{(\sigma, F)}^s \{y_i \mid i \in I\}$$

It follows that $\xi(\bar{x}, \bar{y}) \succ_{(\sigma, F)}^s \{y_1, \dots, y_n\}$. Moreover, $S \models \xi(\bar{a}, \bar{b})$. Thus $\mu_i(\bar{x}, z) \succ_{(\sigma, F)}^s \{z\}$ and $S \models \mu_i(\bar{a}, b_i)$ for every $i \in I$. By definition of $\alpha_{(\sigma, F)}[S, \bar{a}]$, $b_i \in \alpha_{(\sigma, F)}$ for every $i \in I$. Since $\alpha_{(\sigma, F)}[S, \bar{a}]$ is a substructure of S then $\alpha_{(\sigma, F)}[S, \bar{a}] \models p(\bar{b})$. ◀

► **Definition 20.** Let φ and $\psi(x_1, \dots, x_m, z)$ be two formulas of σ such that φ contains no bound instances of x_1, \dots, x_m . $Re_{\psi(\bar{x}, z)}[\varphi]$ is the formula obtained by recursively replacing in φ all subformulas of the forms $\exists w \theta$ with $\exists w. \psi(\bar{x}, w) \wedge \theta$.

► **Lemma 21.** Assume that $F(p) \neq \emptyset$ for every predicate p of σ . Let φ and $\psi(x_1, \dots, x_m, z)$ be two formulas of σ such that φ contains no bound instances of x_1, \dots, x_m , and $\psi(\bar{x}, z) \succ_{(\sigma, F)}^s \{z\}$. Then $Re_{\psi(\bar{x}, z)}[\varphi] \succ_{(\sigma, F)}^s \emptyset$.

Proof. The proof is by induction on the structure of φ :

1. Since $F(p) \neq \emptyset$ for every predicate p of σ , $\theta \succ_{(\sigma, F)}^s \emptyset$ for every atomic formula θ of σ (see Note 7).

⁴ This is the place in the proof of Theorem 16 where we use the assumption that σ includes a constant.

2. By clauses 3,4,5 of Definition 9, $\theta \succ_{(\sigma,F)}^s \emptyset$ for every boolean combination θ of formulas $\theta_1, \dots, \theta_k$ of σ such that $\theta_i \succ_{(\sigma,F)}^s \emptyset$ for every $1 \leq i \leq k$.
3. By clause 6 of Definition 9 $\exists w. \psi(\bar{x}, w) \wedge \theta \succ_{(\sigma,F)}^s \emptyset$ whenever $\theta \succ_{(\sigma,F)}^s \emptyset$. \blacktriangleleft

End of the proof of Theorem 16

By Theorem 10 and Lemma 15, it suffices to prove that if $\varphi \succ_{(\sigma,F)} Y$ then there exists a formula ψ of σ such that $\psi \succ_{(\sigma,F)}^s Y$ and $\varphi \equiv \psi$. Moreover, by Lemma 17 we may assume that σ contains no predicate p for which $F(p) = \emptyset$.

Let $x_1, \dots, x_m, y_1, \dots, y_n, z$ be $m + n + 1$ distinct variables, and let $\varphi(\bar{x}, \bar{y})$ be a formula of σ such that $\varphi(\bar{x}, \bar{y}) \succ_{(\sigma,F)} \{y_1, \dots, y_n\}$. Without a loss in generality, we may assume that φ contains no bound instances of x_1, \dots, x_m . Let σ_q be obtained from σ by the addition of a new $(m + 1)$ -ary predicate symbol q . Define in σ_q :

$$\psi'(\bar{x}, \bar{y}) := Re_{q(\bar{x}, z)}[\varphi(\bar{x}, \bar{y})] \wedge \bigwedge_{i=1}^n q(\bar{x}, y_i)$$

Let T be the set of all formulas of σ_q of the form $\forall z[\theta(\bar{x}, z) \rightarrow q(\bar{x}, z)]$ where $\theta(\bar{x}, z) \succ_{(\sigma,F)}^s \{z\}$ (and so θ is in σ). We will prove that $T \vdash^t \forall \bar{y}(\varphi(\bar{x}, \bar{y}) \leftrightarrow \psi'(\bar{x}, \bar{y}))$.

Let S be a structure for σ_q and let \bar{a} be a tuple in S^m such that $S, \bar{x} := \bar{a} \models T$. Let S_3 be the structure for σ obtained from S by restricting it to σ . Let S_2 be the substructure of S_3 whose domain is the set of all $b \in S$ such that $S \models q(\bar{a}, b)$. Let S_1 be the structure $\alpha_{(\sigma,F)}[S_3, \bar{a}]$. By definition of T and the fact that $S, \bar{x} := \bar{a} \models T$, S_1 is a substructure of S_2 . By Lemmas 19 and 18, S_1 is a (σ, F) -substructure of both S_2 and S_3 . In addition, $\varphi(\bar{x}, \bar{y}) \succ_{(\sigma,F)} \{y_1, \dots, y_n\}$ and $\bar{a} \in S_1^m$. (The latter can be justified by the fact that for every $1 \leq i \leq m$, the formula $x_i = z$ of σ satisfies $x_i = z \succ_{(\sigma,F)}^s \{z\}$.) Therefore, for every $\bar{b} \in S_3^n$:

$$S_3 \models \varphi(\bar{a}, \bar{b}) \iff \bar{b} \in S_1^n \wedge S_1 \models \varphi(\bar{a}, \bar{b})$$

$$S_3 \models \varphi(\bar{a}, \bar{b}) \iff \bar{b} \in S_2^n \wedge S_2 \models \varphi(\bar{a}, \bar{b})$$

Since $\varphi(\bar{x}, \bar{y})$ is a formula of σ (since it does not contain the predicate q), we get that for every $\bar{b} \in S_3^n$, $S \models \varphi(\bar{a}, \bar{b})$ iff $S_3 \models \varphi(\bar{a}, \bar{b})$. By relativization and definition of S_2 , we get that for every $\bar{b} \in S_3^n$, $\bar{b} \in S_2^n \wedge S_2 \models \varphi(\bar{a}, \bar{b})$ iff $S \models \psi'(\bar{a}, \bar{b})$. By transitivity, we get that for every $\bar{b} \in S_3^n$, $S \models \varphi(\bar{a}, \bar{b})$ iff $S \models \psi'(\bar{a}, \bar{b})$. Because S_3 and S have the same domain, we get that $S, \bar{x} := \bar{a} \models \forall \bar{y}(\varphi(\bar{x}, \bar{y}) \leftrightarrow \psi'(\bar{x}, \bar{y}))$.

We proved that $T \vdash^t \forall \bar{y}(\varphi(\bar{x}, \bar{y}) \leftrightarrow \psi'(\bar{x}, \bar{y}))$. By compactness, there exists a finite subset T_1 of T such that:

$$(*) \quad T_1 \vdash^t \forall \bar{y}(\varphi(\bar{x}, \bar{y}) \leftrightarrow \psi'(\bar{x}, \bar{y}))$$

Suppose $T_1 = \{\forall z[\theta_i(\bar{x}, z) \rightarrow q(\bar{x}, z)] \mid 1 \leq i \leq n\}$, and let $\mu(\bar{x}, z)$ be the disjunction of $\theta_1(\bar{x}, z), \dots, \theta_n(\bar{x}, z)$. Then μ is a formula of σ such that $\mu(\bar{x}, z) \succ_{(\sigma,F)}^s \{z\}$. Obtain the set of formulas T_2 of σ and the formula $\psi(\bar{x}, \bar{y})$ of σ from T_1 and ψ' respectively by replacing every atom of the form $q(\bar{x}, z)$ in them by $\mu(\bar{x}, z)$. Since classical first-order logic is structural (that is: its consequence relation is closed under allowed substitutions of formulas for predicate symbols), $(*)$ implies that $T_2 \vdash^t \forall \bar{y}(\varphi(\bar{x}, \bar{y}) \leftrightarrow \psi(\bar{x}, \bar{y}))$. Since the definition of μ entails that all formulas in T_2 are logically valid, this implies that $\varphi \equiv \psi$. Moreover, $\psi(\bar{x}, \bar{y})$ is $Re_{\mu(\bar{x}, z)}[\varphi(\bar{x}, \bar{y})] \wedge \bigwedge_{i=1}^n \mu(\bar{x}, y_i)$. Hence Lemma 21 entails that $\psi(\bar{x}, \bar{y}) \succ_{(\sigma,F)}^s \{y_1, \dots, y_n\}$ (relying on earlier assumption that $F(p) \neq \emptyset$ for every predicate p in σ). \blacktriangleleft

► **Note 22.** Theorem 16 is not always correct as is in case σ contains no constant. Take for example the case where σ is empty. (So the language has ‘=’ as its sole predicate symbol, and no constants or function symbols.) It is easy to prove that there is no formula ψ of this language such that $\psi \succ_{(\sigma, F)}^s Fv(\psi)$. Hence there is no ψ in this language such that $\psi \equiv x \neq x$ and $\psi \succ_{(\sigma, F)}^s \{x\}$, even though obviously $x \neq x \succ_{(\sigma, F)} \{x\}$ (where $F(=)$ is $\{\{1\}, \{2\}\}$). Still, $x \neq x$ is logically equivalent to some formula ψ such that $\psi \succ_{(\sigma, F)}^s \{x\}$ and $x \in Fv(\psi)$ (e.g. $\psi := x = y \wedge x \neq x$). It is indeed easy to infer from Theorem 16 that in general, if σ contains no constant and $\varphi \succ_{(\sigma, F)} X$ then there is a formula ψ such that ψ is logically equivalent to φ , $\psi \succ_{(\sigma, F)}^s \{x\}$, and $Fv(\varphi) \subseteq Fv(\psi)$.

4 Characterization of General Absoluteness

As we saw in the first section, while in database theory the main interest is in formulas which are domain-independent (i.e. formulas which are safe with respect to their full set of free variables), in formal number theory (and in computability theory) and in set theory the main interest has been in absolute formulas.⁵ Now in the previous section we have given a general syntactic characterization of absoluteness: Given a safety signature (σ, F) , a formula φ is (σ, F) -absolute iff there exists a formulas ψ such that $\varphi \equiv \psi$, and $\varphi \succ_{(\sigma, F)}^s \emptyset$. However, this characterization of the *property* of absoluteness is based in an essential way on the *relation* $\succ_{(\sigma, F)}^s$ between formulas and sets of variables. Therefore in order to check whether a certain formula φ is absolute using this characterization, one should check on the way with respect to what sets of variables are the subformulas of φ safe. In contrast, in formal number theory and in set theory a direct syntactic approximation of absoluteness has been used in the form of what is called in both Δ_0 -formulas. In this section we generalize the notion of Δ_0 -formulas to arbitrary safety signatures, and use the generalized notion for providing a *direct* syntactic characterization of (σ, F) -absoluteness. Note that in order to use this characterization one needs not know anything about the more general binary relation $\succ_{(\sigma, F)}^s$.

► **Notation 23.** For a formula φ and a set of variables $Z = \{z_1, \dots, z_k\}$, $\exists^Z \varphi$ denotes the formula $\exists z_1, \dots, \exists z_k. \varphi$, and $\forall^Z \varphi$ denotes the formula $\forall z_1, \dots, \forall z_k. \varphi$.

► **Definition 24.** Let (σ, F) be a safety signature. The class $\Delta_{(\sigma, F)}$ of formulas⁶ is recursively defined as follows:

1. $p(t_1, \dots, t_n) \in \Delta_{(\sigma, F)}$ in case p is an n -ary predicate symbol of σ , and $F(p) \neq \emptyset$.
2. If $\varphi, \psi \in \Delta_{(\sigma, F)}$ then so is any boolean combination of them.
3. $\exists^Z \varphi_1 \wedge \varphi_2 \in \Delta_{(\sigma, F)}$ and $\forall^Z \varphi_1 \rightarrow \varphi_2 \in \Delta_{(\sigma, F)}$ in case $\varphi_2 \in \Delta_{(\sigma, F)}$, $\varphi_1 = p(t_1, \dots, t_n)$, where p is an n -ary predicate of σ other than $=$, and $\varphi_1 \succ_{(\sigma, F)}^s Z$, that is: there is $I \in F(p)$ such that:
 - a. For every $z \in Z$ there is $i \in I$ such that $z = t_i$.
 - b. $Z \cap Fv(t_j) = \emptyset$ for every $j \in \{1, \dots, n\} \setminus I$.

Examples

$\Delta_{(\sigma_{ZF}, F_{ZF})}$ is exactly the class Δ_0 used in set theory. Similarly, $\Delta_{(\sigma_{\mathcal{N}}, F_{\mathcal{N}})}$ is equivalent to the class Δ_0 used in formal number theory. (See the first two examples in Section 2.3.)

⁵ Actually, absolute formulas may be of interest for databases too, since they can be used for effectively decidable yes-or-no queries. See [3].

⁶ This is a proper extension of the class GF of guarded formulas ([2]), in case F is the particular function which assigns the powerset of $\{1, \dots, n\}$ to every n -ary primitive predicate R of σ .

► **Theorem 25.** $\varphi \succ_{(\sigma,F)}^s \emptyset$ iff there exists a formula $\varphi' \in \Delta_{(\sigma,F)}$ such that $\varphi \equiv \varphi'$.

Proof. Obviously, if $\varphi' \in \Delta_{(\sigma,F)}$ then $\varphi' \succ_{(\sigma,F)}^s \emptyset$. Hence the condition is sufficient. In order to prove that it is also necessary, we need the following lemma:

► **Lemma 26.** Let $\succ_{(\sigma,F)}^*$ be defined like $\succ_{(\sigma,F)}^s$, except that the clause for conjunction is replaced by:

If $\varphi_1 \succ_{(\sigma,F)}^* Y_1$, $\varphi_2 \succ_{(\sigma,F)}^* Y_2$, $Fv(\varphi_1) \cap Y_2 = \emptyset$ and φ_1 is an atomic formula or $Y_1 = \emptyset$, then

$$\varphi_1 \wedge \varphi_2 \succ_{(\sigma,F)}^* Y_1 \cup Y_2.$$

Then for every formula φ , $\varphi \succ_{(\sigma,F)}^s Y$ iff there is a formula φ' such that $\varphi' \succ_{(\sigma,F)}^* Y$ and $\varphi \equiv \varphi'$.

Proof. Obviously, if $\varphi' \succ_{(\sigma,F)}^* Y$ then $\varphi' \succ_{(\sigma,F)}^s Y$. Hence the condition is sufficient. In order to prove that it is also necessary, it suffices to show that up to logical equivalence, $\succ_{(\sigma,F)}^*$ abides the condition concerning \wedge used in the definition of $\succ_{(\sigma,F)}^s$. So assume e.g. that $\varphi_1 \succ_{(\sigma,F)}^* Y_1$, $\varphi_2 \succ_{(\sigma,F)}^* Y_2$ and $Fv(\varphi_1) \cap Y_2 = \emptyset$. We prove the existence of a formula φ' such that $\varphi_1 \wedge \varphi_2 \equiv \varphi'$ and $\varphi' \succ_{(\sigma,F)}^* Y_1 \cup Y_2$. The proof is by induction on the structure of φ_1 :

- Assume φ_1 is an atomic formula. Then $\varphi_1 \wedge \varphi_2 \succ_{(\sigma,F)}^* Y_1 \cup Y_2$ by the new conjunction safety clause.
- Assume φ_1 is the formula $\psi_1 \vee \psi_2$ where $\psi_1 \succ_{(\sigma,F)}^* Y_1$ and $\psi_2 \succ_{(\sigma,F)}^* Y_1$. Since $Fv(\varphi_1) = Fv(\psi_1) \cup Fv(\psi_2)$, we know that $Fv(\psi_1) \cap Y_2 = Fv(\psi_2) \cap Y_2 = \emptyset$. Then, by induction assumption, there exist formulas θ_1 and θ_2 such that $\psi_1 \wedge \varphi_2 \equiv \theta_1$, $\psi_2 \wedge \varphi_2 \equiv \theta_2$, $\theta_1 \succ_{(\sigma,F)}^* Y_1 \cup Y_2$ and $\theta_2 \succ_{(\sigma,F)}^* Y_1 \cup Y_2$. Therefore $\varphi_1 \wedge \varphi_2 \equiv \theta_1 \vee \theta_2$ and $\theta_1 \vee \theta_2 \succ_{(\sigma,F)}^* Y_1 \cup Y_2$.
- Assume φ_1 is the formula $\psi_1 \wedge \psi_2$ where $\psi_1 \succ_{(\sigma,F)}^* Z_1$, $\psi_2 \succ_{(\sigma,F)}^* Z_2$, $Fv(\psi_1) \cap Z_2 = \emptyset$, $Y_1 = Z_1 \cup Z_2$ and ψ_1 is an atomic formula or $Z_1 = \emptyset$. Since $Fv(\psi_2) \cap Y_2 = \emptyset$, we get by induction assumption the existence of a formula θ such that $\psi_2 \wedge \varphi_2 \equiv \theta$ and $\theta \succ_{(\sigma,F)}^* Z_2 \cup Y_2$. Since $Fv(\psi_1) \cap (Z_2 \cup Y_2) = \emptyset$, we get that $\varphi_1 \wedge \varphi_2 \equiv \psi_1 \wedge \theta$ and $\psi_1 \wedge \theta \succ_{(\sigma,F)}^* Y_1 \cup Y_2$.
- Assume φ_1 is the formula $\neg\psi$ where $\psi \succ_{(\sigma,F)}^* \emptyset$. Then $Y_1 = \emptyset$ and then $\varphi_1 \wedge \varphi_2 \succ_{(\sigma,F)}^* Y_1 \cup Y_2$ by the new conjunction safety clause.
- Assume $\varphi_1 = \exists z\psi$ where $\psi \succ_{(\sigma,F)}^* Y_1 \cup \{z\}$ and $z \notin Y_1$. In addition, assume w.l.o.g. that $z \notin Fv(\varphi_2)$. Since $Fv(\psi) \cap Y_2 = \emptyset$, we get by induction assumption the existence of a formula θ such that $\psi \wedge \varphi_2 \equiv \theta$ and $\theta \succ_{(\sigma,F)}^* Y_1 \cup Y_2 \cup \{z\}$. Then $\varphi_1 \wedge \varphi_2 \equiv \exists z\theta$ and $\exists z\theta \succ_{(\sigma,F)}^* Y_1 \cup Y_2$.

This completes the induction. ◀

End of the proof of Theorem 25

We show the necessity of the condition by proving a stronger claim: For every formula φ such that $\varphi \succ_{(\sigma,F)}^s Y$ there exists a formula $\varphi' \in \Delta_{(\sigma,F)}$ such that $\exists^Y \varphi \equiv \varphi'$. By Lemma 26, we only need to prove the latter under the assumption that $\varphi \succ_{(\sigma,F)}^* Y$. The proof in this case is by induction on the structure of φ :

- Assume φ is atomic. If $Y = \emptyset$ then $\varphi \in \Delta_{(\sigma,F)}$. Otherwise, choosing $y \in Y$, we get $y = y \succ_{(\sigma,F)} \emptyset$, $\exists^Y(\varphi \wedge y = y) \in \Delta_{(\sigma,F)}$ and $\exists^Y \varphi \equiv \exists^Y(\varphi \wedge y = y)$.
- Assume φ is the formula $\psi_1 \vee \psi_2$ where $\psi_1 \succ_{(\sigma,F)}^* Y$ and $\psi_2 \succ_{(\sigma,F)}^* Y$. By induction assumption, there exists formulas $\theta_1 \in \Delta_{(\sigma,F)}$ and $\theta_2 \in \Delta_{(\sigma,F)}$ such that $\exists^Y \psi_1 \equiv \theta_1$ and $\exists^Y \psi_2 \equiv \theta_2$. Then $\theta_1 \vee \theta_2 \in \Delta_{(\sigma,F)}$ and $\exists^Y \varphi \equiv \theta_1 \vee \theta_2$.

- Assume φ is $\psi_1 \wedge \psi_2$ where $\psi_1 \succ_{(\sigma,F)}^* Y_1$, $\psi_2 \succ_{(\sigma,F)}^* Y_2$, $Fv(\psi_1) \cap Y_2 = \emptyset$, $Y = Y_1 \cup Y_2$ and ψ_1 is an atomic formula or $Y_1 = \emptyset$. By induction assumption, there exists a formula $\theta_2 \in \Delta_{(\sigma,F)}$ such that $\exists^{Y_2} \psi_2 \equiv \theta_2$. If $Y_1 = \emptyset$ then, by induction assumption, there exists a formula $\theta_1 \in \Delta_{(\sigma,F)}$ such that $\psi_1 \equiv \theta_1$ and so $\theta_1 \wedge \theta_2 \in \Delta_{(\sigma,F)}$ and $\exists^Y \varphi \equiv \theta_1 \wedge \theta_2$. Otherwise, if $Y_1 \neq \emptyset$ then ψ_1 is an atomic formula, $\exists^{Y_1} (\psi_1 \wedge \theta_2) \in \Delta_{(\sigma,F)}$ and $\exists^Y \varphi \equiv \exists^{Y_1} (\psi_1 \wedge \theta_2)$.
- Assume φ is the formula $\neg\psi$ where $\psi \succ_{(\sigma,F)}^* \emptyset$. Then $Y = \emptyset$ and, by induction assumption, there exists a formula $\theta \in \Delta_{(\sigma,F)}$ such that $\psi \equiv \theta$ and so $\neg\theta \in \Delta_{(\sigma,F)}$ and $\varphi \equiv \neg\theta$.
- Assume $\varphi = \exists z \psi$ where $\psi \succ_{(\sigma,F)}^* Y \cup \{z\}$ and $z \notin Y$. By induction assumption, there exists a formula $\theta \in \Delta_{(\sigma,F)}$ such that $\exists^Y \varphi \equiv \exists^{Y \cup \{z\}} \psi \equiv \theta$.

By Note 11, this completes the proof. \blacktriangleleft

5 Characterization of Absoluteness in \mathcal{N}

Theorem 25 is about *general* (σ, F) -absoluteness, and so it is not applicable to the notion of (S, F) -absoluteness, where S is a structure for σ . In this section we prove a similar theorem for one particular, but very important, case of (S, F) -absoluteness: $(\mathcal{N}, \sigma_{\mathcal{N}})$ -absoluteness.

► **Note 27.** Recall that in Section 2.2.1 it was noted that by a result of [3], relation R on N is recursively enumerable iff R is definable by a formula of the form $\exists y_1, \dots, y_n \psi$, where the formula ψ is $(\sigma_{\mathcal{N}}, F_{\mathcal{N}})$ -absolute. It was further observed there that every relation on \mathcal{N} that is defined by a $(\mathcal{N}, F_{\mathcal{N}})$ -absolute formula is decidable, and that there are decidable relations on \mathcal{N} that are not definable by any formula φ such that $\varphi \succ_{(\sigma_{\mathcal{N}}, F_{\mathcal{N}})}^s \emptyset$. It was left open whether every decidable relation on \mathcal{N} is definable by a $(\sigma_{\mathcal{N}}, F_{\mathcal{N}})$ -absolute formula, and whether every relation which is definable by such a formula is already definable by a formula φ such that $\varphi \succ_{(\sigma_{\mathcal{N}}, F_{\mathcal{N}})}^s \emptyset$. In view of the above-mentioned observations, the next theorem implies that the answer to the first question is negative, while the answer to the second is positive.

► **Theorem 28.** *A formula φ such that $Fv(\varphi) \neq \emptyset$ is $(\mathcal{N}, \sigma_{\mathcal{N}})$ -absolute iff there is an arithmetical bounded formula⁷ φ' such that φ is equivalent in \mathcal{N} to φ' .*

Proof. We assume without loss of generality that for every formula ψ it holds that $Fv(\psi) \cap Bv(\psi) = \emptyset$, and that any two variables that appear in ψ to the right of two different occurrences of quantifiers are different. For $k \in N$ we denote by \mathcal{N}_k the structure with domain $\{0, 1, \dots, k\}$, and the interpretations of the relation symbols are the corresponding reductions of the interpretations of those symbols in \mathcal{N} . For an assignment v , a variable u , and a natural value n , we denote $v[u := n]$ the assignment that agree with v on all variables except u , and assigns the value n to u .

Given a formula φ and a set $\{x_1, \dots, x_k\}$ of variables such that $\{x_1, \dots, x_k\} \cap Bv(\varphi) = \emptyset$, we denote by $\varphi^{\leq x_1, \dots, x_k}$ the formula $Re_{\psi(\bar{x}, z)}[\varphi]$ (Definition 20), where $\psi(\bar{x}, z)$ is $z \leq x_1 \vee \dots \vee z \leq x_k$. The proof of the theorem is based on the following three lemmas:

► **Lemma 29.** *$\varphi^{\leq x_1, \dots, x_k}$ is logically equivalent to a bounded formula for every formula φ .*

Proof. This follows immediately from the definitions, and the fact that $\exists z. (z \leq x_1 \vee \dots \vee z \leq x_k) \wedge \psi$ is logically equivalent to the formula $\exists z \leq x_1. \psi \wedge \dots \wedge \exists z \leq x_k. \psi$. \blacktriangleleft

⁷ See first example in Section 2.3.

8:12 Safety, Absoluteness, and Computability

► **Lemma 30.** *Let φ be a formula, let $\{y, x_1, \dots, x_k\}$ be a set of variables s.t. $Bv(\varphi) \cap \{y, x_1, \dots, x_k\} = \emptyset$, and let v be an assignment s.t. $v(y) \leq \max(v(x_1), \dots, v(x_k))$. Then the following holds*

$$\mathcal{N}, v \models \varphi^{\leq x_1, \dots, x_k} \quad \text{iff} \quad \mathcal{N}, v \models \varphi^{\leq y, x_1, \dots, x_k}$$

Proof. We prove it by a structural induction on φ . The only non-trivial case is when φ is of the form $\exists z.\psi$. In this case $\varphi^{\leq x_1, \dots, x_k}$ is $\exists z.(z \leq x_1 \vee \dots \vee z \leq x_k) \wedge \psi^{\leq x_1, \dots, x_k}$. Hence $\mathcal{N}, v \models \varphi^{\leq x_1, \dots, x_k}$ iff (*) there exists $n \in N$ such that:

$$\mathcal{N}, v[z := n] \models (z \leq x_1 \vee \dots \vee z \leq x_k) \wedge \psi^{\leq x_1, \dots, x_k}$$

Obviously, $v'(y) \leq \max(v'(x_1), \dots, v'(x_k))$ for every assignment v' that agrees with v on $\{y, x_1, \dots, x_k\}$. Hence the induction hypothesis for ψ implies that for any such v' :

$$\mathcal{N}, v' \models \psi^{\leq x_1, \dots, x_k} \quad \text{iff} \quad \mathcal{N}, v' \models \psi^{\leq y, x_1, \dots, x_k}. \quad (1)$$

Also for any such v' , $\mathcal{N}, v' \models z \leq y \vee z \leq x_1 \vee \dots \vee z \leq x_k$ iff $\mathcal{N}, v' \models z \leq x_1 \vee \dots \vee z \leq x_k$ (because $z \notin Bv(\varphi)$, and so $z \notin \{y, x_1, \dots, x_n\}$). This observation and 1 imply that (*) holds iff there exists $n \in N$ such that:

$$\mathcal{N}, v[z := n] \models (z \leq y \vee z \leq x_1 \vee \dots \vee z \leq x_k) \wedge \psi^{\leq y, x_1, \dots, x_k}$$

And this is equivalent to: $\mathcal{N}, v \models \varphi^{\leq y, x_1, \dots, x_k}$. ◀

► **Lemma 31.** *Let $\{x_1, \dots, x_k\}$ be a non-empty set of variables, let φ be a formula such that $Fv(\varphi) \subseteq \{x_1, \dots, x_k\}$, and let v be an assignment. Denote by $\tilde{m} := \max(v(x_1), \dots, v(x_k))$. Then:*

$$\mathcal{N}_{\tilde{m}}, v \models \varphi \quad \text{iff} \quad \mathcal{N}, v \models \varphi^{\leq x_1, \dots, x_k} \quad (2)$$

Proof. By a structural induction on φ . Again the only non-trivial case is when φ is of the form $\exists z.\psi$. So let v be an assignment, and assume that $\mathcal{N}_{\tilde{m}}, v \models \exists y.\psi$. It follows that there exists $n \in N$, $0 \leq n \leq \tilde{m}$, s.t. $\mathcal{N}_{\tilde{m}}, v[y := n] \models \psi$. By the induction hypothesis for ψ and $\{y, x_1, \dots, x_k\}$, it holds that $\mathcal{N}, v[y := n] \models \psi^{\leq y, x_1, \dots, x_k}$. Denote the assignment $v[y := n]$ by v' . Since $v'(y) = n \leq \tilde{m} = \max(v'(x_1), \dots, v'(x_k))$, $\mathcal{N}, v[y := n] \models \psi^{\leq x_1, \dots, x_k}$ by Lemma 30. Hence $\mathcal{N}, v \models \exists y.(y \leq x_1 \vee \dots \vee y \leq x_k) \wedge \psi^{\leq x_1, \dots, x_k}$, that is: $\mathcal{N}, v \models \varphi^{\leq x_1, \dots, x_k}$. To prove the converse we just repeat the argument in reverse order: Assume that $\mathcal{N}, v \models \varphi^{\leq x_1, \dots, x_k}$. This means that $\mathcal{N}, v \models \exists y.(y \leq x_1 \vee \dots \vee y \leq x_k) \wedge \psi^{\leq x_1, \dots, x_k}$. It follows that there is $0 \leq n \leq \tilde{m}$ s.t. $\mathcal{N}, v[y := n] \models \psi^{\leq x_1, \dots, x_k}$. Using Lemma 30, it follows that $\mathcal{N}, v[y := n] \models \psi^{\leq y, x_1, \dots, x_k}$. Therefore the induction hypothesis and the fact that $\tilde{m} := \max(v(x_1), \dots, v(x_k))$ together imply that $\mathcal{N}_{\tilde{m}}, v[y := n] \models \psi$. Hence $\mathcal{N}_{\tilde{m}}, v \models \varphi$. ◀

End of the proof of Theorem 28

Suppose that $\varphi \succ_{(\mathcal{N}, F_{\mathcal{N}})} \emptyset$, and let $Fv(\varphi) = \{x_1, \dots, x_k\}$ where $k \geq 1$. Consider the formula $\varphi' = \varphi^{\leq x_1, \dots, x_k}$. ($\varphi' \succ_{\mathcal{N}}^s \emptyset$ by Lemma 29.) We show that

$$\{\bar{n} \in N^k \mid \mathcal{N}, \bar{x} := \bar{n} \models \varphi\} = \{\bar{n} \in N^k \mid \mathcal{N}, \bar{x} := \bar{n} \models \varphi^{\leq x_1, \dots, x_k}\}$$

Let $\langle n_1, \dots, n_k \rangle \in N^k$, and let v be an assignment that assigns n_i to x_i for every $1 \leq i \leq k$. Since $\varphi \succ_{(\mathcal{N}, F_{\mathcal{N}})} \emptyset$, $\mathcal{N}, v \models \varphi$ iff $\mathcal{N}_{\max(n_1, \dots, n_k)}, v \models \varphi$. By Lemma 31 $\mathcal{N}_{\max(n_1, \dots, n_k)}, v \models \varphi$ iff $\mathcal{N}, v \models \varphi^{\leq x_1, \dots, x_k}$, and the claim follows. ◀

6 Absoluteness in Rudimentary Set Theory

To complete the picture concerning absoluteness, we return in this section to the area in which this notion has first been introduced: set theory. In Sections 2.2.2 and 2.3 (second example) we have noted that the notion of (σ_{ZF}, F_{ZF}) -absoluteness is identical to Gödel's original notion of absoluteness, and that $\{\varphi \mid \varphi \succ_{(\sigma_{ZF}, F_{ZF})}^s \emptyset\}$ is a natural extension of the set of Δ_0 -formulas in the language of σ_{ZF} . However, in order to fully exploit the power of the idea of dependent safety in the framework of set theory, we need to use a language which is stronger (and more natural) than the official language of ZF . The main feature of the stronger language, \mathcal{L}_{RST} , is that it employs a rich class of set terms of the form $\{x \mid \varphi\}$. Of course, not every formula φ can be used in such a term. The basic idea in [5] was that from a predicative point of view, one should allow only formulas which are safe with respect to $\{x\}$. Since safety is a semantic notion, again what is used instead in [5] is a formal approximation \succ_{RST} . \succ_{RST} is basically the natural extension of $\succ_{(\sigma_{ZF}, F_{ZF})}^s$ to the richer language. However, the definition of that very language depends in turn on that of \succ_{RST} . Accordingly, the sets of terms and formulas of \mathcal{L}_{RST} , and the relation \succ_{RST} , are defined together by a simultaneous induction:

► **Definition 32.** The language \mathcal{L}_{RST} is defined as follows:

Terms:

1. Every variable is a term.
2. If x is a variable, and φ is a formula such that $\varphi \succ_{RST} \{x\}$, then $\{x \mid \varphi\}$ is a term (and $Fv(\{x \mid \varphi\}) = Fv(\varphi) - \{x\}$).

Formulas:

1. If t, s are terms then $t = s$ and $t \in s$ are atomic formulas.
2. If φ and ψ are formulas, then $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, and $\exists x\varphi$ are formulas.

The safety relation \succ_{RST} :

1. $\varphi \succ_{RST} \emptyset$ if φ is atomic.
2. $\varphi \succ_{RST} \{x\}$ if $\varphi \in \{x \in x, x = t, t = x, x \in t\}$, and $x \notin Fv(t)$.
3. $\neg\varphi \succ_{RST} \emptyset$ if $\varphi \succ_{RST} \emptyset$.
4. $\varphi \vee \psi \succ_{RST} X$ if $\varphi \succ_{RST} X$ and $\psi \succ_{RST} X$.
5. $\varphi \wedge \psi \succ_{RST} X \cup Y$ if $\varphi \succ_{RST} X$, $\psi \succ_{RST} Y$, and $Y \cap Fv(\varphi) = \emptyset$ or $X \cap Fv(\psi) = \emptyset$.
6. $\exists y\varphi \succ_{RST} X - \{y\}$ if $y \in X$ and $\varphi \succ_{RST} X$.

► **Theorem 33 ([5]).** *Every term of \mathcal{L}_{RST} with n free variables explicitly defines an n -ary rudimentary function, and every rudimentary function is defined by some term of \mathcal{L}_{RST} .*

The two most basic formal set theories in the language \mathcal{L}_{RST} are described next.

► **Definition 34.**

1. RST^m is the first-order theory with equality in the language \mathcal{L}_{RST} ⁸ which has the following axioms:
 - Extensionality: $\forall z(z \in x \leftrightarrow z \in y) \rightarrow x = y$
 - Comprehension: $\forall x(x \in \{x \mid \varphi\} \leftrightarrow \varphi)$ if $\varphi \succ_{RST} \{x\}$.
2. RST is the system obtained from RST^m by the addition of the following schema:
 - \in -induction: $(\forall x(\forall y(y \in x \rightarrow \varphi\{y/x\}) \rightarrow \varphi)) \rightarrow \forall x\varphi$

⁸ \mathcal{L}_{RST} has richer classes of terms than those allowed in orthodox first-order systems. In particular: a variable can be bound in them within a term. The notion of a term being free for substitution should be extended accordingly. Otherwise the rules/axioms concerning the quantifiers, terms, and equality remain unchanged.

► **Note 35.** The use of \in -induction seems to be predicatively justified. Therefore RST is the basic system used in [5]. However, for the results below we use just one very weak corollary of it: $\forall x.x \notin x$. (It is needed for the new clause $x \in x \succ_{RST} \{x\}$ in Definition 32.)

► **Note 36.** RST (or even just RST^m) serves in [5], [6] and [7] as the basis of *computational set theories*. By this we mean a theory whose set of closed terms suffices for defining its minimal model, and can be used to make explicit the potential computational content of set theories (first suggested and partially demonstrated in [8]). On the other hand, such theories also suffice (as is shown in [6] and [7]) for developing large portions of what was called by Feferman in [11] ‘scientifically applicable mathematics’.

► **Note 37.** Despite the fact that the definition of \succ_{RST} uses almost exactly the same principles that underlie that of $\succ_{(\sigma_{ZF}, F_{ZF})}^s$ (with the slight addition that $x \in x \succ_{RST} \{x\}$, while we only have $x \in x \succ_{(\sigma_{ZF}, F_{ZF})}^s \emptyset$), the use of abstract set terms induces a significantly stronger safety relation on the basic language of σ_{ZF} . The reason is that the fact that $x = t \succ_{RST} \{x\}$ is equivalent in RST^m to the following principle:

■ If $\varphi \succ_{RST} \{y\}$ then $\forall y(y \in x \leftrightarrow \varphi) \succ_{RST} \{x\}$ if $x \notin Fv(\varphi)$.

(It is not difficult to show that the addition of this clause indeed suffices for getting a system in the language of ZF which is equivalent to RST .) Nevertheless, the next theorem and its corollary imply that when it comes to *absoluteness*, the addition of the abstract set terms does not provide extra expressive power.

► **Theorem 38.** *Let ψ be a Δ_0 formula of σ_{ZF} (that is, without abstract set terms).*

1. *If x is a variable, and t is a term which is free for x in ψ , then $\psi\{t/x\}$ is equivalent in RST to a Δ_0 -formula of σ_{ZF} .*
2. *If $\varphi \succ_{RST} \{x_1, \dots, x_n\}$ then the formula $\exists x_1 \dots x_n(\varphi \wedge \psi)$ is equivalent in RST to a Δ_0 -formula of σ_{ZF} .*

Proof. By a simultaneous induction on the complexity of t and φ .

- If t is a variable then the claim is obvious.
- Suppose t is $\{y \mid \varphi\}$, where $\varphi \succ_{RST} \{y\}$. We prove the claim for t by an internal induction on the complexity of ψ .
 - If x is not free in ψ then the claim is obvious.
 - If ψ is $x \in x$ then $\psi\{t/x\}$ is equivalent in RST to the formula $\exists x \in x.x \in x$.
 - If ψ is $x = x$ then $\psi\{t/x\}$ is equivalent in RST to the formula $\neg \exists x \in x.x \in x$.
 - Suppose ψ is $z \in x$, where z is different from x . We may assume that z is not bound in φ . Then $\psi\{t/x\}$ is equivalent in RST to $\varphi\{z/y\}$. Since $\varphi \succ_{RST} \emptyset$, φ is equivalent in RST to a Δ_0 -formula by the induction hypothesis. Hence so does $\varphi\{z/y\}$.
 - Suppose ψ is $z = x$ or $x = z$, where z is a variable different from x . We may assume that z is not y . Then $\psi\{t/x\}$ is equivalent in RST to $(\forall y \in z.\varphi) \wedge \neg \exists y(\varphi \wedge y \notin z)$. Since $\varphi \succ_{RST} \{y\}$ and $\varphi \succ_{RST} \emptyset$, φ and $\exists y(\varphi \wedge y \notin z)$ are equivalent in RST to Δ_0 -formulas by the external induction hypothesis for φ . It follows that so is $\psi\{t/x\}$.
 - Suppose ψ is $x \in z$, where z is a variable different from x . Let w be a fresh variable. Then $\psi\{t/x\}$ is logically equivalent to $\exists w \in z.w = t$. By the previous case, $w = t$ is equivalent in RST to a Δ_0 formula. Hence so is $\psi\{t/x\}$.
 - If ψ is $\neg\psi_1$ or $\psi_1 \wedge \psi_2$, or $\psi_1 \vee \psi_2$, then the claim for ψ follows from the induction hypothesis for ψ_1 and ψ_2 .
 - If ψ is of the form $\exists z \in w.\psi_1$, where both w and z are different from x , then the claim for ψ is immediate from the internal induction hypothesis for ψ_1 .

- Suppose ψ is of the form $\exists z \in x. \psi_1$ (where z is different from x). Since t is free for x in ψ , z does not occur free in φ , and we may assume that it does not occur in φ at all. Then $\psi\{t/x\}$ is equivalent in RST to $\exists z(\varphi\{z/y\} \wedge \psi_1\{t/x\})$. Since z does not occur in φ and $\varphi \succ_{RST} \{y\}$, also $\varphi\{z/y\} \succ_{RST} \{z\}$. Hence by the external induction hypothesis for φ and the internal induction hypothesis for ψ_1 , $\psi\{t/x\}$ is equivalent in RST to a Δ_0 formula.
- Suppose φ is atomic (and so $\varphi \succ_{RST} \emptyset$). Then φ is either $t_1 \in t_2$ or $t_1 = t_2$ for some terms t_1 and t_2 . Since $x \in y$ and $x = y$ are Δ_0 -formulas, it follows by applying the induction hypotheses for t_1 and t_2 that φ is equivalent to a Δ_0 -formula. Hence $\varphi \wedge \psi$ is equivalent to a Δ_0 -formula whenever ψ is.
- Suppose that φ is of the form $x \in x$, where x is a variable, Then $\varphi \succ_{RST} \{x\}$, and so we have to prove that $\exists x \in x. \psi$ is equivalent to a Δ_0 -formula. This is obvious.
- Suppose that φ is of the form $x \in t$, where $x \notin Fv(t)$. Since $\varphi \succ_{RST} \{x\}$ in this case, we have to prove that for every Δ_0 -formula ψ , $\exists x(x \in t \wedge \psi)$ is equivalent to a Δ_0 -formula. This follows from the induction hypothesis for t , since the last formula is $\theta\{t/z\}$, where z is a fresh variable, and θ is the Δ_0 -formula $\exists x(x \in z) \wedge \psi$ (note that since $x \notin Fv(t)$, t is free for z in θ).
- Suppose that φ is of the form $x = t$ or $t = x$, where x is not free in t . Since $\varphi \succ_{RST} \{x\}$ in this case, we have to prove that for every Δ_0 -formula ψ , $\exists x(x = t \wedge \psi)$ is equivalent to a Δ_0 -formula. By changing bound variables, we may assume that t is free for x in ψ . This and the fact that $x \notin Fv(t)$ together imply that $\exists x(x = t \wedge \psi)$ is logically equivalent to $\psi\{t/x\}$. This formula, in turn, is equivalent in RST to a Δ_0 -formula by our induction hypothesis for t .
- Suppose φ is $\neg\varphi_1$, where $\varphi_1 \succ_{RST} \emptyset$ (and so $\neg\varphi_1 \succ_{RST} \emptyset$). By induction hypothesis for φ , φ is equivalent in RST to a Δ_0 -formula. Hence so is $\neg\varphi \wedge \psi$ for every Δ_0 -formula ψ .
- Suppose φ is $\varphi_1 \vee \varphi_2$, where $\varphi_1 \succ_{RST} \{x_1, \dots, x_n\}$ and $\varphi_2 \succ_{RST} \{x_1, \dots, x_n\}$ (and so $\varphi \succ_{RST} \{x_1, \dots, x_n\}$). Then $\exists x_1 \dots x_n(\varphi \wedge \psi)$ is logically equivalent to $\exists x_1 \dots x_n(\varphi_1 \wedge \psi) \vee \exists x_1 \dots x_n(\varphi_2 \wedge \psi)$. Hence the induction hypothesis for φ_1 and φ_2 entails that $\exists x_1 \dots x_n(\varphi \wedge \psi)$ is equivalent in RST to a Δ_0 -formula whenever ψ is.
- Suppose φ is $\varphi_1 \wedge \varphi_2$, $\varphi_1 \succ_{RST} \{x_1, \dots, x_n\}$, $\varphi_2 \succ_{RST} \{y_1, \dots, y_k\}$, $\{y_1, \dots, y_k\} \cap Fv(\varphi_1) = \emptyset$ (so $\varphi \succ_{RST} \{x_1, \dots, x_n, y_1, \dots, y_k\}$). Then $\exists x_1 \dots x_n y_1 \dots y_k(\varphi \wedge \psi)$ is equivalent to $\exists x_1 \dots x_n(\varphi_1 \wedge \exists y_1 \dots y_k(\varphi_2 \wedge \psi))$. By applying the induction hypothesis twice, we get that $\exists x_1 \dots y_k(\varphi \wedge \psi)$ is equivalent in RST to a Δ_0 -formula whenever ψ is.
- Suppose φ is $\exists y \varphi_1$ where $\varphi_1 \succ_{RST} \{x_1, \dots, x_n, y\}$. Let ψ be a Δ_0 -formula. Then $\exists x_1 \dots x_n(\varphi \wedge \psi)$ is logically equivalent to the formula $\exists x_1 \dots x_n z(\varphi_1\{z/y\} \wedge \psi)$, where z is a fresh variable. Since $\varphi_1\{z/y\} \succ_{RST} \{x_1, \dots, x_n, z\}$, the induction hypothesis implies that $\exists x_1 \dots x_n z(\varphi_1\{z/y\} \wedge \psi)$ is equivalent in RST to a Δ_0 -formula. Hence so is $\exists x_1 \dots x_n(\varphi \wedge \psi)$. ◀

► **Corollary 39.** *If $\varphi \succ_{RST} \emptyset$ then φ is equivalent in RST to a Δ_0 -formula of σ_{ZF} .*

► **Note 40.** On the other hand, if $X \neq \emptyset$, then it can happen that $\varphi \succ_{RST} X$, but $\theta \not\succeq_{RST} X$, where θ is the Δ_0 -formula to which φ is equivalent according to the construction given in the last proof. Thus if φ is $x = \{y\}$, then θ is the $y \in x \wedge \forall z \in x. z = y$, so $\theta \not\succeq_{RST} \{x\}$, even though $\varphi \succ_{RST} \{x\}$. This problem cannot be solved by adding to the definition of \succ_{RST} the clause mentioned in Note 37, because $\forall y(y \in x \leftrightarrow \varphi)$ is not necessarily a Δ_0 -formula in case φ is. From the above theorem it follows that it is equivalent in RST to a Δ_0 -formula θ , but then again there seems to be no guarantee that $\theta \succ_{RST} \{x\}$.

7 Conclusion and Further Research

We have shown that the syntactic framework developed in [3, 5] for the semantic notions of dependent safety and absoluteness is complete in the case of *general* first-order logic in languages without function symbols. Therefore it promises to be rather adequate for the general theory of constructibility, decidability, and computability envisaged in [3]. The next stages of this research program will involve the following goals:

1. Extending the general theory of dependent safety for languages with function symbols.
2. The completeness result given in this paper is with respect to the class of *all* structures for a given signature. However, frequently we are mainly interested only with a subclass of that class. Two particularly important cases for which an extension of the general theory developed here is needed are:
 - a. The class of finite models.
 - b. The class of the models of some given theory.
3. For computability theory we might need to restrict our attention to *specific* central structures. Thus in section (5) we characterized the absolute formulas of the important structure $(\mathcal{N}, F_{\mathcal{N}})$. It is not clear whether the same can be done for other basic important structures, like the structure of hereditarily finite sets $\mathcal{HF} = (\mathbb{HF}, \langle \in \rangle)$ (where \in has its usual meaning, and $x \in y$ is safe with respect to $\{x\}$).
4. Providing concrete applications of our results in specific areas. This includes:
 - Database theory (e.g. Datalog extended with arithmetic).
 - MKM (Mathematical Knowledge Management), in particular: the formalization of scientifically applicable mathematics in a type-free, predicative setting ([5]).

References

- 1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- 2 H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.
- 3 A. Avron. Constructibility and decidability versus domain independence and absoluteness. *Theoretical Computer Science*, 394:144–158, 2008.
- 4 A. Avron. A framework for formalizing set theories based on the use of static set terms. In A. Avron, N. Dershowitz, and A. Rabinovich, editors, *Pillars of Computer Science*, volume 4800 of *LNCS*, pages 87–106. Springer, 2008.
- 5 A. Avron. A new approach to predicative set theory. In R. Schindler, editor, *Ways of Proof Theory*, onto series in mathematical logic, pages 31–63. onto verlag, 2010.
- 6 A. Avron and L. Cohen. Formalizing scientifically applicable mathematics in a definitional framework. *Journal of Formalized Reasoning*, 9(1):53–70, 2016.
- 7 A. Avron and L. Cohen. A minimal computational theory of a minimal computational universe. In *Proc. of LFCS 2018*, pages 37–54, 2018.
- 8 D. Cantone, E. Omodeo, and A. Policriti. *Set theory for computing: from decision procedures to declarative programming with sets*. Springer, 2001.
- 9 K. J. Devlin. *Constructibility*. Perspectives in Mathematical Logic. Springer-Verlag, 1984.
- 10 R. A. Di Paola. The recursive unsolvability of the decision problem for the class of definite formulas. *J. ACM*, 16:324–327, 1969.
- 11 S. Feferman. Why a little bit goes a long way: Logical foundations of scientifically applicable mathematics. In *PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association*, pages 442–455, 1992.
- 12 R. O. Gandy. Set-theoretic functions for elementary syntax. In *Axiomatic set theory, Part 2*, pages 103–126. AMS, Providence, Rhode Island, 1974.

- 13 R. B. Jensen. The fine structure of the constructible hierarchy. *Annals of Mathematical Logic*, 4:229–308, 1972.
- 14 K. Kunen. *Set Theory, An Introduction to Independence Proofs*. North-Holland, 1980.
- 15 A. O. Mendelzon and T. Milo. Formal models of web queries. In *Proceedings of the Sixteenth ACM Symposium on Principles of Database Systems*, pages 134–143, 1997.
- 16 R. Ramakrishnan, F. Bancilhon, and A. Silberschatz. Safety of recursive horn clauses with infinite relations. In *ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, San Diego*, 1987.
- 17 R. M. Smullyan. *The Incompleteness Theorems*. Oxford University Press, 1992.
- 18 R. Topor. Safe database queries with arithmetic relations. In *Proceedings of the 14th Australian Computer Science Conference*, pages 1–13, Sydney, 1991.
- 19 Rodney W. Topor. Domain-independent formulas and databases. *Theoretical Computer Science*, 52(3):281–306, 1987.
- 20 J.D. Ullman. *Principles of Database and Knowledge-base Systems*. Computer Science Press, 1988.

Combining Linear Logic and Size Types for Implicit Complexity

Patrick Baillot

Univ Lyon, CNRS, ENS de Lyon, Université Claude-Bernard Lyon 1, LIP
F-69342, Lyon Cedex 07, France

Alexis Ghyselen

ENS Paris-Saclay, 94230 Cachan, France

Abstract

Several type systems have been proposed to statically control the time complexity of lambda-calculus programs and characterize complexity classes such as FPTIME or FEXPTIME. A first line of research stems from linear logic and restricted versions of its !-modality controlling duplication. A second approach relies on the idea of tracking the size increase between input and output, and together with a restricted recursion scheme, to deduce time complexity bounds. However both approaches suffer from limitations : either a limited intensional expressivity, or linearity restrictions. In the present work we incorporate both approaches into a common type system, in order to overcome their respective constraints. Our system is based on elementary linear logic combined with linear size types, called sEAL, and leads to characterizations of the complexity classes FPTIME and $2k$ -FEXPTIME, for $k \geq 0$.

2012 ACM Subject Classification Theory of computation \rightarrow Lambda calculus, Theory of computation \rightarrow Linear logic, Theory of computation \rightarrow Turing machines, Software and its engineering \rightarrow Functional languages

Keywords and phrases Implicit computational complexity, λ -calculus, linear logic, type systems, polynomial time complexity, size types

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.9

Related Version Combining Linear Logic and Size Types for Implicit Complexity (Long Version), Patrick Baillot, Ghyselen Alexis, <https://hal.archives-ouvertes.fr/hal-01687224>, 2018

1 Introduction

Controlling the time complexity of programs is a crucial aspect of program development. Complexity analysis can be performed on the overall final program and some automatic techniques have been devised for this purpose. However, if the program does not meet our expected complexity bound it might not be easy to track which subprograms are responsible for the poor performance and how they should be rewritten in order to improve the global time bound. Can one instead investigate some methodologies to program while staying in a given complexity class? Can one carry such program construction without having to deal with explicit annotations for time bounds? These are some of the questions that have been explored by *implicit computational complexity*, a line of research which defines calculi and logical systems corresponding to various complexity classes, such as FP, FEXPTIME, FLOGSPACE . . .

A first success in implicit complexity was the recursion-theoretic characterization of FP [9]. This work on safe recursion leads to languages for polynomial time [18], for oracle functionals or for probabilistic computation [13, 25]. Among the other different approaches of implicit



© Patrick Baillot and Alexis Ghyselen;
licensed under Creative Commons License CC-BY
27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 9; pp. 9:1–9:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

complexity one can mention two important threads of work. The first one is issued from linear logic, which provides a decomposition of intuitionistic logic with a modality, $!$, accounting for duplication. By designing variants of linear logic with weak versions of the $!$ modality one obtains systems corresponding to different complexity classes, like light linear logic (LLL) for the class FP [15] and elementary linear logic (ELL) for the classes k -FEXPTIME, for $k \geq 0$. [15, 2, 14]. These logical systems can be seen as type systems for some variants of lambda-calculi. A key feature of these systems, and the main ingredient for proving their complexity properties, is that they induce a stratification of the typed program into levels. We will thus refer to them as *level-based systems*. Their advantage is that they deal with a higher-order language, and that they are also compatible with polymorphism. Unfortunately from a programming point of view they have a critical drawback: only few and very specific programs are actually typable, because the restrictions imposed to recursion by typing are in fact very strong... A second thread of work relies on the idea of tracking the size increase between the input and the output of a program. This approach is well illustrated by Hofmann's Non-size-increasing (NSI) type system [19] : here the types carry information about the input/output size difference, and the recursion is restricted in such a way that typed programs admit polynomial time complexity. An important advantage with respect to LLL is that the system is algorithmically more expressive, that is to say that far more programs are typable. This has triggered a fertile research line on type-based complexity analysis using ideas of amortized cost analysis [20, 17, 16]. Some aspects of higher-order have been adressed [22] but note that this approach deals with complexity analysis and not with the characterization of complexity classes. In particular it does not suggest disciplines to program within a given complexity class. A similar idea is also explored by the line of work on quasi-interpretations [10, 4], with a slightly different angle : here the kind of dependence between input and output size can be more general but the analysis is more of a semantic nature and in particular no type system is provided to derive quasi-interpretations. The type system $d\ell T$ of [3] can be thought of as playing this role of describing the dependence between input and output size, and it allows to derive time complexity bounds, even though these are not limited to polynomial bounds. Altogether we will refer to these approaches as *size-based systems*. However they also have a limitation: characterizations of complexity classes have not been obtained for full-fledged higher-order languages, but only for linear higher-order languages, that is to say languages in which functional arguments have to be used at most once (as in [19, 4]).

Problematic and methodology. So on the one hand level-based systems manage higher-order but have a poor expressivity, and on the other hand sized-based systems have a good expressivity but do not characterize complexity classes within a general higher-order language... On both sides some attempts have been made to repair these shortcomings but only with limited success: in [6] for instance LLL is extended to a language with recursive definitions, but the main expressivity problem remains; in [4] quasi-interpretations are defined for a higher-order language, but with a linearity condition on functional arguments. The goal of the present work is precisely to improve this situation by reconciling the level-based and the size-based approaches. From a practical point of view we want to design a system which would bring together the advantages of the two approaches. From a fundamental point of view we want to understand how the levels and the input/output size dependencies are correlated, and for instance if one of these two characteristics subsumes the other one.

One way to bridge these two approaches could be to start with a level-based system such as LLL, and try to extend it with more typing rules so as to integrate in it some size-based features. However a technical difficulty for that is that the complexity bounds for LLL and

variants of this system are usually obtained by following specific term reduction strategies such as the *level-by-level* strategy. Enriching the system while keeping the validity of such reduction strategies turns out to be very intricate. For instance this has been done in [6] for dealing with recursive definitions with pattern-matching, but at the price of technical and cumbersome reasonings on the reduction sequences. Our methodology to overcome this difficulty in the present work will be to choose a variant of linear logic for which we can prove the complexity bound by using a measure which decreases for *any* reduction step. So in this case there is no need for specific reduction strategy, and the system is more robust to extensions. For that purpose we use elementary linear logic (ELL), and more precisely the elementary lambda-calculus studied in [24].

Our language. Let us recall that ELL is essentially obtained from linear logic by dropping the two axioms $!A \multimap A$ and $!A \multimap !!A$ for the $!$ functor (the co-unit and co-multiplication of the comonad). Basically, if we consider the family of types $W \multimap !^i W$ (where W is a type for binary words), the larger the integer i , the more computational power we get... This results in a system that can characterize the classes $k\text{-FEXPTIME}$, for $k \geq 0$ [2]. The paper [24] gives a reformulation of the principles of ELL in an extended lambda-calculus with constructions for $!$. It also incorporates other features (references and multithreading) which we will not be interested in here. Our idea will be to enrich the elementary lambda-calculus by a kind of *bootstrapping*, consisting in adding more terms to the “basic” type $W \multimap W$. For instance we can think of giving to this type enough terms for representing all polynomial time functions. The way we implement this idea is by using a second language. We believe that several equivalent choices could be made for this second language, and here we adopt for simplicity a variant of the language $d\ell T$ from [3], a descendant of previous work on linear dependent types [23]. This language is a linear version of system T, that is to say a lambda-calculus with recursion, with types annotated with size expressions. Actually the type system of our second language can be thought of as a linear cousin of sized types [21, 1] and we call it $s\ell T$. So on the whole our global language can be viewed as a kind of two-layer system, the lower one used for tuning first-order intensional expressivity, and the upper one for dealing with higher-order computation and non-linear use of functional arguments. We will call it $sEAL$, for *sized Elementary affine logic* typed λ -calculus. We do not include polymorphism in $sEAL$ for the simplicity of exposition, but we are convinced that our results could be adapted to the polymorphic extension.

Roadmap. We will first define the language $s\ell T$ of sized linear types and investigate its properties (Sect. 2). Then we will recall the elementary lambda-calculus, define our enriched calculus $sEAL$, describe some examples of programs and study the reduction properties of this calculus (Sect. 3). After that we will establish the complexity results (Sect. 4).

2 Presentation of $s\ell T$ and Control of the Reduction Procedure

We present $s\ell T$ which is a linear λ -calculus with constructors for base types and a constructor for high-order primitive recursion. Types are enriched with a polynomial index describing the size of the value represented by a term, and this index imposes a restriction on recursions. With this, we are able to derive a weight on terms in order to control the number of reduction steps.

$$\begin{array}{l|l}
\text{if}(V, V') \text{ tt} \rightarrow V & (\lambda x.t) V \rightarrow t[V/x] \\
\text{if}(V, V') \text{ ff} \rightarrow V' & \text{let } x \otimes y = V \otimes V' \text{ in } t \rightarrow t[V/x][V'/y] \\
\text{ifn}(V, V') \text{ zero} \rightarrow V' & \text{ifn}(V, V') \text{ succ}(W) \rightarrow V W \\
\text{itern}(V, V') \text{ zero} \rightarrow V' & \text{itern}(V, V') \text{ succ}(W) \rightarrow \text{itern}(V, V V') W \\
\text{ifw}(V_0, V_1, V') \epsilon \rightarrow V' & \text{ifw}(V_0, V_1, V') \mathbf{s}_i(W) \rightarrow V_i W \\
\text{iterw}(V_0, V_1, V') \epsilon \rightarrow V' & \text{iterw}(V_0, V_1, V') \mathbf{s}_i(W) \rightarrow \text{iterw}(V_0, V_1, V_i V') W
\end{array}$$

■ **Figure 1** Base rules for $s\ell\mathcal{T}$.

2.1 Syntax of $s\ell\mathcal{T}$ and Type System

► **Definition 1** (Substitution). For an object t with a notion of free variable and substitution we write $t[t'/x]$ the term t in which free occurrences of x have been replaced by t' .

Terms. Terms and values of $s\ell\mathcal{T}$ are defined by the following grammars :

$$\begin{aligned}
t &:= x \mid \lambda x.t \mid t t' \mid t \otimes t' \mid \text{let } x \otimes y = t \text{ in } t' \mid \text{zero} \mid \text{succ}(t) \mid \text{ifn}(t, t') \mid \text{itern}(V, t) \mid \epsilon \\
&\mid \mathbf{s}_0(t) \mid \mathbf{s}_1(t) \mid \text{ifw}(t_0, t_1, t') \mid \text{iterw}(V_0, V_1, t) \mid \text{tt} \mid \text{ff} \mid \text{if}(t, t') \\
V &:= x \mid \lambda x.t \mid V \otimes V' \mid \text{zero} \mid \text{succ}(V) \mid \text{ifn}(V, V') \mid \text{itern}(V, V') \mid \epsilon \mid \mathbf{s}_0(V) \mid \mathbf{s}_1(V) \\
&\mid \text{ifw}(V_0, V_1, V') \mid \text{iterw}(V_0, V_1, V') \mid \text{tt} \mid \text{ff} \mid \text{if}(V, V')
\end{aligned}$$

We define free variables and free occurrences as usual and we work up to α -renaming. In the following, we will often use the notation \mathbf{s}_i to regroup the cases \mathbf{s}_0 and \mathbf{s}_1 . Here, we choose the alphabet $\{0, 1\}$ for simplification, but we could have taken any finite alphabet Σ and in this case, the constructors ifw and iterw would need a term for each letter.

The definitions of the constructors will be more explicit with their reductions rules and their types. For intuition, the constructor $\text{ifn}(t, t')$ can be seen as $\lambda n.\text{match } n \text{ with succ}(n') \mapsto t n' \mid 0 \mapsto t'$, and the constructor $\text{itern}(V, t)$ is such that $\text{itern}(V, t) \underline{n} \rightarrow^* V^n t$, if \underline{n} is the coding of the integer n , that is $\text{succ}^n(\text{zero})$.

Reductions. Base reductions in $s\ell\mathcal{T}$ are given by the rules described in Figure 1.

Note that in the iterw rule, the order in which we apply the steps functions is the reverse of the one for iterators we see usually. In particular, it does not correspond to the reduction defined in [3]. This is not a problem since we can compute the mirror of a word and the subject reduction is easier to prove with this definition. Those base reductions can be applied in contexts C defined by the following grammar : $C := [] \mid C t \mid V C \mid C \otimes t \mid t \otimes C \mid \text{let } x \otimes y = C \text{ in } t \mid \text{succ}(C) \mid \text{ifn}(C, t) \mid \text{ifn}(t, C) \mid \text{itern}(V, C) \mid \mathbf{s}_i(C) \mid \text{ifw}(C, t, t') \mid \text{ifw}(t, C, t') \mid \text{ifw}(t, t', C) \mid \text{iterw}(V_0, V_1, C) \mid \text{if}(C, t) \mid \text{if}(t, C)$.

Linear Types with Sizes. Base types are given by the following grammar :

$$U := W^I \mid N^I \mid B \quad I, J, \dots := a \mid n \in \mathbb{N}^* \mid I + J \mid I \cdot J$$

\mathbb{N}^* is the set of non-zero integers. I represents an *index* and a represents an *index variable*. We define for indexes the notions of free variables and free occurrences in the usual way and we work up to renaming of variables. We also define the substitution of a free variable in an index in the usual way. Then, we can generalize substitution to types, with for example $N^I[J/a] = N^{I[J/a]}$.

The intended meaning is that closed values of type N^I (resp. W^I) will be integers (resp. words) of size (resp. length) at most I .

► **Definition 2** (Order on Indexes). For two indexes I and J , we say that $I \leq J$ if for any valuation ϕ mapping free variables of I and J to non-zero integers, we have $I_\phi \leq J_\phi$. I_ϕ is I where free variables have been replaced by their value in ϕ , thus I_ϕ is a non-zero integer.

We now consider that if $I \leq J$ and $J \leq I$ then $I = J$ (ie we take the quotient set for the equivalence relation). Remark that by definition of indexes, we always have $1 \leq I$. For two indexes I and J , we say that $I < J$ if for any valuation ϕ mapping free variables of I and J to non-zero integers, we have $I_\phi < J_\phi$. This is not equivalent to $I \leq J$ and $I \neq J$, as we can see with $a \leq a \cdot b$.

Here we only consider polynomial indexes. This is a severe restriction w.r.t. linear dependent types, used for example in [12, 3], in which indexes can use any set of functions described by some rewrite rules. But in the present setting this is sufficient because we only want $\text{s}\ell\text{T}$ to characterize polynomial time computation.

► **Definition 3.** Types are given by the grammar $D, E, \dots := U \mid D \multimap D' \mid D \otimes D'$

We define a subtyping order \sqsubset on types given by the following rules :

- $B \sqsubset B$ and if $I \leq J$ then $N^I \sqsubset N^J$ and $W^I \sqsubset W^J$.
- $D_1 \multimap D'_1 \sqsubset D_2 \multimap D'_2$ iff $D_2 \sqsubset D_1$ and $D'_1 \sqsubset D'_2$.
- $D_1 \otimes D'_1 \sqsubset D_2 \otimes D'_2$ iff $D_1 \sqsubset D_2$ and $D'_1 \sqsubset D'_2$.

► **Definition 4 (Contexts).** *Variables contexts* are denoted Γ , with the shape $\Gamma = x_1 : D_1, \dots, x_n : D_n$. We say that $\Gamma \sqsubset \Gamma'$ when Γ and Γ' have exactly the same variables, and for $x : D$ in Γ and $x : D'$ in Γ' we have $D \sqsubset D'$. *Ground variables contexts*, denoted $d\Gamma$, are variables contexts in which all types are base types. We write $\Gamma = \Gamma', d\Gamma$ to denote the decomposition of Γ into a ground variable context $d\Gamma$ and a variable context Γ' in which types are non-base types. For a variable context without base types, we note $\Gamma = \Gamma_1, \Gamma_2$ when Γ is the concatenation of Γ_1 and Γ_2 , and Γ_1 and Γ_2 do not have any common variables.

We denote proofs as $\pi \triangleleft \Gamma \vdash t : D$ and we define an index $\omega(\pi)$ called the *weight* for such a proof. The idea is that the weight will be an upper-bound for the number of reduction steps of t . Note that since $\omega(\pi)$ is an index, this bound can depend of some index variables. The rules for those proofs are described by Figure 2. The rules for words and booleans can be found in the appendix 6.1, they can be deduced from the rules for integers. Observe that this system enforces a linear usage of variables of non-base types (see e.g. the rule for application in Fig. 2). Note that in the rule for `itern` described in Figure 2, the index variable a must be a fresh variable.

Example in $\text{s}\ell\text{T}$. We sketch here the multiplication in $\text{s}\ell\text{T}$, other examples can be found in the appendix 6.3. The multiplication can be written $\text{mult} = \lambda x. \text{itern}(\lambda y. \text{add } x \ y, \text{zero}) : N^I \multimap N^J \multimap N^{I+J}$, if we are given the term $\text{add} : N^I \multimap N^J \multimap N^{I+J}$.

$$\frac{\frac{x : N^I, y : N^{I-a} \vdash \text{add } x \ y : N^{I-a+I}}{x : N^I \vdash \lambda y. \text{add } x \ y : N^{I-a} \multimap N^{I-a}[a+1/a]} \quad x : N^I \vdash \text{zero} : N^I}{x : N^I \vdash \text{itern}(\lambda y. \text{add } x \ y, \text{zero}) : N^J \multimap N^{I+J}}$$

2.2 Subject Reduction and Upper Bound

In order to prove the subject reduction for $\text{s}\ell\text{T}$ and that the weight is a bound on the number of reduction steps of a term, we give some important intermediate lemmas. Other lemmas can be found in the appendix 6.2, and more details are available in [7], as for other sections in this paper. First, we show that values are indeed linked to normal forms. In particular, this theorem shows that a value of type integer is indeed of the form $\text{succ}(\text{succ}(\dots(\text{succ}(\text{zero}))\dots))$. This imposes that in this call-by-value calculus, when an argument is of type N , it is the encoding of an integer.

$$\begin{array}{c}
\pi \triangleleft \frac{D \sqsubset D'}{\Gamma, x : D \vdash x : D'} \quad \omega(\pi) = 1 \\
\pi \triangleleft \frac{\sigma \triangleleft \Gamma, x : D \vdash t : D'}{\Gamma \vdash \lambda x. t : D \multimap D'} \quad \omega(\pi) = 1 + \omega(\sigma) \\
\pi \triangleleft \frac{\sigma_1 \triangleleft \Gamma_1, d\Gamma \vdash t : D' \multimap D \quad \sigma_2 \triangleleft \Gamma_2, d\Gamma \vdash t' : D'}{\Gamma_1, \Gamma_2, d\Gamma \vdash t t' : D} \quad \omega(\pi) = \omega(\sigma_1) + \omega(\sigma_2) \\
\pi \triangleleft \frac{\sigma_2 \triangleleft \Gamma_2, d\Gamma \vdash t' : D' \quad \sigma_1 \triangleleft \Gamma_1, d\Gamma \vdash t : D}{\Gamma_1, \Gamma_2, d\Gamma \vdash t \otimes t' : D \otimes D'} \quad \omega(\pi) = \omega(\sigma_1) + \omega(\sigma_2) + 1 \\
\pi \triangleleft \frac{\sigma_2 \triangleleft \Gamma_2, d\Gamma, x : D, y : D' \vdash t' : D'' \quad \sigma_1 \triangleleft \Gamma_1, d\Gamma \vdash t : D \otimes D'}{\Gamma_1, \Gamma_2, d\Gamma \vdash \mathbf{let} \ x \otimes y = t \ \mathbf{in} \ t' : D''} \quad \omega(\pi) = \omega(\sigma_1) + \omega(\sigma_2) \\
\pi \triangleleft \frac{}{\Gamma \vdash \mathbf{zero} : \mathbf{N}^I} \quad \omega(\pi) = 0 \\
\pi \triangleleft \frac{J + 1 \leq I \quad \sigma \triangleleft \Gamma \vdash t : \mathbf{N}^J}{\Gamma \vdash \mathbf{succ}(t) : \mathbf{N}^I} \quad \omega(\pi) = \omega(\sigma) \\
\pi \triangleleft \frac{\sigma_1 \triangleleft \Gamma_1, d\Gamma \vdash t : \mathbf{N}^I \multimap D \quad \sigma_2 \triangleleft \Gamma_2, d\Gamma \vdash t' : D}{\Gamma_1, \Gamma_2, d\Gamma \vdash \mathbf{ifn}(t, t') : \mathbf{N}^I \multimap D} \quad \omega(\pi) = \omega(\sigma_1) + \omega(\sigma_2) + 1 \\
\pi \triangleleft \frac{D \sqsubset E \quad E[I/a] \sqsubset F \quad E \sqsubset E[a + 1/a] \quad \sigma_1 \triangleleft d\Gamma \vdash V : D \multimap D[a + 1/a] \quad \sigma_2 \triangleleft \Gamma, d\Gamma \vdash t : D[1/a]}{\Gamma, d\Gamma \vdash \mathbf{itern}(V, t) : \mathbf{N}^I \multimap F} \quad \omega(\pi) = I + \omega(\sigma_2) + I \cdot \omega(\sigma_1)[I/a]
\end{array}$$

■ **Figure 2** Type system for $s\ell\mathcal{T}$.

► **Theorem 5.** *Let t be a term in $s\ell\mathcal{T}$, if t is closed and has a typing derivation $\vdash t : D$ then t is normal if and only if t is a value V .*

Another important lemma is the one for subtyping.

► **Lemma 6 (Subtyping).** *If $\pi \triangleleft \Gamma \vdash t : D$ then for all Γ', D' such that $D \sqsubset D'$ and $\Gamma' \sqsubset \Gamma$, we have a proof $\pi' \triangleleft \Gamma' \vdash t : D'$ with $\omega(\pi') \leq \omega(\pi)$*

This lemma shows that we do not need an explicit rule for subtyping and subtyping does not harm the upper bound derived from typing. Moreover, this lemma is important in order to substitute variables, since the axiom rule allows subtyping.

We can now express the subject-reduction of the calculus and the fact that the weight of a proof strictly decreases during a reduction.

► **Theorem 7.** *Let $\tau \triangleleft \Gamma \vdash t_0 : D$, and $t_0 \rightarrow t_1$, then there is a proof $\tau' \triangleleft \Gamma \vdash t_1 : D$ such that $\omega(\tau') < \omega(\tau)$.*

The proof of this theorem can be found in [7]. The main difficulty is to prove the statement for base reductions. Base reductions that induce a substitution, like the usual β reduction, are proved by a substitution lemma. The other interesting cases are the rules for iterators. For such a rule, the subject reduction is given by a good use of the fresh variable given in the typing rule.

As the indexes can only define polynomials, the weight of a sequent can only be a polynomial on the index variables. And so, in $s\ell\mathcal{T}$, we can only define terms that work in time polynomial in their inputs.

Polynomial Indexes and Degree. For the following section on the elementary affine logic, we need to define a notion of degree of indexes and explicit some properties of this notion.

► **Definition 8.** The indexes can be seen as multi-variables polynomials, and we can define the *degree* of an index I by induction on I .

- $\forall n \in \mathbb{N}^*, d(n) = 0$
- For an index variable a , $d(a) = 1$
- $d(I + J) = \max(d(I), d(J))$
- $d(I \cdot J) = d(I) + d(J)$.

This definition of degree is primordial for the control of reductions in sEAL, that we present in the following section.

3 Elementary Affine Logic and Sizes

We work on an elementary affine lambda calculus based on [24] without multithreading and side-effects, that we present here. In order to solve the problem of intensional expressivity of this calculus, we enrich it with constructors for integers, words and booleans, and some iterators on those types following the usual constraint on iteration in elementary affine logic (EAL). Then, using the fact that the proof of correctness in [24] is robust enough to support functions computable in polynomial time with type $\mathbb{N} \multimap \mathbb{N}$ (see Section 6.4 in the appendix), we enrich EAL with the polynomial time calculus defined previously. We call this new language sEAL (EAL with sizes). More precisely, we add the possibility to use first-order s ℓ T terms in this calculus in order to work on those base types, particularly we can then do controlled iterations for those types. We then adapt the measure used in [24] to sEAL to find an upper-bound on the number of reductions for a term.

3.1 An EAL-Calculus

First, let us present a λ -calculus for the elementary affine logic. In this calculus, any sequence of reduction terminates in elementary time. The keystone of this proof is the use of the modality “!” , called *bang*, inspired by linear logic. In order to have this bound, there are some restrictions in the calculus like linearity (or *affinity* if we allow weakening) and an important notion linked with the “!” is used, the *depth*. We follow the presentation from [24] and we encode the usual restrictions in a type system.

Syntax. Terms are given by the grammar: $M := x \mid \lambda x.M \mid M M' \mid !M \mid \mathbf{let} !x = M \mathbf{in} M'$

The constructor $\mathbf{let} !x = M \mathbf{in} M'$ binds the variable x in M' . We define as usual the notion of free variables, free occurrences and substitution.

The semantic of this calculus is given by the two following rules

$$(\lambda x.M) M' \rightarrow M[M'/x] \quad \mathbf{let} !x = !M \mathbf{in} M' \rightarrow M'[M/x].$$

Those rules can be applied in any contexts.

Type System. We add to this calculus a polymorphic type system that also restrains the possible terms we can write. Types are given by the grammar $T := \alpha \mid T \multimap T' \mid !T \mid \forall \alpha.T$

► **Definition 9 (Contexts).** *Linear variables contexts* are denoted Γ , with the shape $\Gamma = x_1 : T_1, \dots, x_n : T_n$. We write Γ_1, Γ_2 the disjoint union between Γ_1 and Γ_2 . *Global variables contexts* are denoted Δ , with the shape $\Delta = x_1 : T_1, \dots, x_n : T_n, y_1 : [T'_1], \dots, y_m : [T'_m]$. We say that $[T]$ is a *discharged type*, as we could see in light linear logic [15, 26]. When we need to separate the discharged types from the others, we will write $\Delta = \Delta'', [\Delta']$. In this case, if $[\Delta'] = y_1 : [T'_1], \dots, y_m : [T'_m]$, then we note $\Delta' = y_1 : T'_1, \dots, y_m : T'_m$.

$$\begin{array}{c}
\frac{}{\Gamma, x : T \mid \Delta \vdash x : T} \text{ (Lin Ax)} \\
\frac{\Gamma, x : T \mid \Delta \vdash M : T'}{\Gamma \mid \Delta \vdash \lambda x.M : T \multimap T'} (\lambda) \\
\frac{\emptyset \mid \Delta \vdash M : T}{\Gamma \mid \Delta', [\Delta] \vdash !M : !T} (! \text{ Intro}) \\
\frac{\Gamma \mid \Delta \vdash M : T \quad \alpha \text{ fresh in } \Gamma, \Delta}{\Gamma \mid \Delta \vdash M : \forall \alpha.T} (\forall \text{ Intro})
\end{array}
\qquad
\begin{array}{c}
\frac{}{\Gamma \mid \Delta, x : T \vdash x : T} \text{ (Glob Ax)} \\
\frac{\Gamma \mid \Delta \vdash M : T' \multimap T \quad \Gamma' \mid \Delta \vdash M' : T'}{\Gamma, \Gamma' \mid \Delta \vdash M M' : T} \text{ (App)} \\
\frac{\Gamma' \mid \Delta \vdash M : !T \quad \Gamma \mid \Delta, x : [T] \vdash M' : T'}{\Gamma, \Gamma' \mid \Delta \vdash \mathbf{let} !x = M \mathbf{in} M' : T'} (! \text{ Elim}) \\
\frac{\Gamma \mid \Delta \vdash M : \forall \alpha.T}{\Gamma \mid \Delta \vdash M : T[T'/\alpha]} (\forall \text{ Elim})
\end{array}$$

■ **Figure 3** Type system for the EAL-calculus.

Typing judgments have the shape $\Gamma \mid \Delta \vdash M : T$.

The rules are given in Figure 3. Observe that all the rules are multiplicative for Γ , and the “! Intro” rule erases linear contexts, non-discharged types and transforms discharged types into usual types. With this, we can see that some restrictions appears in a typed term. First, in $\lambda x.M$, x occurs at most once in M , and moreover, there is no “! Intro” rule behind the axiom rule for x . Then, in $\mathbf{let} !x = M \mathbf{in} M'$, x can be duplicated, but there is exactly one “! Intro” rule behind each axiom rule for x . For example, with this type system, we can not type terms like $\lambda x.!x$, $\lambda f, x.f (f x)$ or $\mathbf{let} !x = M \mathbf{in} x$.

With this type system, we obtain as a consequence of the results exposed in [24] that any sequence of reductions of a typed term terminates in elementary time. This proof relies on the notion of depth linked with the modality “!” and a measure on terms bounding the number of reduction for this term. We will adapt those two notions in the following part on sEAL, but for now, let us present some terms and encoding in this EAL-calculus.

Examples of Terms in EAL and Church Integers. First, a useful term proving the functoriality of $! : \mathit{fonct} = \lambda f, x. \mathbf{let} !g = f \mathbf{in} \mathbf{let} !y = x \mathbf{in} !(g y) : \forall \alpha, \alpha'. !(\alpha \multimap \alpha') \multimap !\alpha \multimap !\alpha'$.

Integers can be encoded in this calculus, using the type $\mathbf{N} = \forall \alpha. !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$. For example, 3 is described by the term $\underline{3} = \lambda f. \mathbf{let} !g = f \mathbf{in} !(\lambda x. g (g (g x))) : \mathbf{N}$.

With this encoding, addition and multiplication can be defined, with type $\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$.
 $\mathit{add} = \lambda n, m, f. \mathbf{let} !f' = f \mathbf{in} \mathbf{let} !g = n !f' \mathbf{in} \mathbf{let} !h = m !f' \mathbf{in} !(\lambda x. h (g x))$
 $\mathit{mult} = \lambda n, m, f. \mathbf{let} !g = f \mathbf{in} n(m !g)$

And finally, one can also define an iterator using integers.

$\mathit{iter} = \lambda f, x, n. \mathit{fonct} (n f) x : \forall \alpha. !(\alpha \multimap \alpha) \multimap !\alpha \multimap \mathbf{N} \multimap !\alpha$ with $\mathit{iter} !M !M' \underline{n} \rightarrow^* !(M^n M')$.

Intensional Expressivity. Those examples show that this calculus suffers from limitation. First, we need to work with Church integers, because of a lack of data structure. Furthermore, we need to be careful with the modality, and this can be sometimes a bit tricky, as one can remark with the addition. And finally if we want to do an iteration, we are forced to work with types with bangs. This implies that each time we need to use an iteration, we are forced to add a bang in the final type. Typically this prevents from iterating a function which has itself been defined by iteration. It has been proved [5] that polynomial and exponential complexity classes can be characterized in this calculus, by fixing types. For example, with a type for words \mathbf{W} and booleans \mathbf{B} we have that $!\mathbf{W} \multimap !!\mathbf{B}$ characterizes polynomial time computation. However, because of the restrictions mentioned above some natural polynomial time programs cannot be typed with the type $!\mathbf{W} \multimap !!\mathbf{B}$. We say that this calculus has a limited intensional expressivity. One goal of this paper is to try to lessen this problem, and for that, we now present an enriched version of this calculus, sEAL, using the language sLT.

3.2 Syntax and Type System for sEAL

Notation. Let us first give some notations on terms and vectors.

► **Definition 10** (Applications). For an object with a notion of application M and an integer n , we write $M^n M'$ to denote n applications of M to M' . In particular, $M^0 M' = M'$

We also define for a word w , given objects M_a for all letter a , $M^w M'$. This is defined by induction on words with $M^\epsilon M' = M'$ and $M^{aw'} M' = M_a (M^{w'} M')$

► **Definition 11** (Vectors). In the following we will work with vectors of \mathbb{N}^{n+1} , for $n \in \mathbb{N}$. We introduce here some notations on those vectors. We usually denote vectors by $\mu = (\mu(0), \dots, \mu(n))$. When there is no ambiguity with the value of n , for $0 \leq k \leq n$, we note $\mathbb{1}_k$ the vector μ with $\mu(k) = 1$ and $\forall i, 0 \leq i \leq n, i \neq k, \mu(i) = 0$. We extend this notation for $k > n$. In this case, $\mathbb{1}_k$ is the zero-vector. Let $\mu_0 \in \mathbb{N}^{n+1}$ and $\mu_1 \in \mathbb{N}^{m+1}$. We denote $\mu = (\mu_0, \mu_1) \in \mathbb{N}^{m+n+2}$ the vector with $\forall i, 0 \leq i \leq n, \mu(i) = \mu_0(i)$ and $\forall i, 0 \leq i \leq m, \mu(i+n+1) = \mu_1(i)$. Let $\mu_0, \mu_1 \in \mathbb{N}^{n+1}$. We write $\mu_0 \leq \mu_1$ when $\forall i, 0 \leq i \leq n, \mu_0(i) \leq \mu_1(i)$. And we write $\mu_0 < \mu_1$ when $\mu_0 \leq \mu_1$ and $\mu_0 \neq \mu_1$. We also write $\mu_0 \leq_{lex} \mu_1$ for the lexicographic order on vectors. For $k \in \mathbb{N}$, when there is no ambiguity with the value of n , we write \tilde{k} the vector μ such that $\forall i, 0 \leq i \leq n, \mu(i) = k$.

Terms and Reductions. Terms of sEAL are defined by the following grammar :

$$\begin{aligned} M := & x \mid \lambda x.M \mid M M' \mid !M \mid \text{let } !x = M \text{ in } M' \mid M \otimes M' \mid \text{let } x \otimes y = M \text{ in } M' \\ & \mid \text{zero} \mid \text{succ}(M) \mid \text{ifn}(M, M') \mid \text{iter}_N^!(M, M') \mid \text{tt} \mid \text{ff} \mid \text{if}(M, M') \mid \epsilon \mid \mathbf{s}_0(M) \mid \mathbf{s}_1(M) \\ & \mid \text{ifw}(M_0, M_1, M) \mid \text{iter}_W^!(M_0, M_1, M) \mid [\lambda x_n \dots x_1.t](M_1, \dots, M_n) \end{aligned}$$

Note that the t used in $[\lambda x_n \dots x_1.t](M_1, \dots, M_n)$ refers to terms defined in sℓT. This notation means that we call the function t defined in sℓT with arguments M_1, \dots, M_n . Moreover, n can be any integer, even 0. Constructors for iterations directly follow from the ones we can define usually in EAL for Church integers or Church words, as we could see in the previous section on EAL. Once again, we often write \mathbf{s}_i to denote \mathbf{s}_0 or \mathbf{s}_1 , and the choice of the alphabet $\{0, 1\}$ is arbitrary, we could have used any finite alphabet. As usual, we work up to α -isomorphism and we do not explicit the renaming of variables.

► **Definition 12** (Base type values). We note \underline{v} for base type values, defined by the grammar $\underline{v} := \text{zero} \mid \text{succ}(\underline{v}) \mid \epsilon \mid \mathbf{s}_i(\underline{v}) \mid \text{tt} \mid \text{ff}$.

In particular, if n is an integer and w is a binary word, we note \underline{n} for the base value $\text{succ}^n(\text{zero})$, and $\underline{w} = \underline{w_1} \dots \underline{w_n}$ for the base value $\mathbf{s}_{w_1}(\dots \mathbf{s}_{w_n}(\epsilon) \dots)$. We define the *size* $|\underline{v}|$ of \underline{v} by $|\text{zero}| = |\epsilon| = |\text{tt}| = |\text{ff}| = 1$ and $|\text{succ}(\underline{v})| = |\mathbf{s}_i(\underline{v})| = 1 + |\underline{v}|$.

Base reductions are defined by the rules given in Figure 4. Note that for some of these rules, for example the last one, \underline{v} can denote either the sℓT term or the sEAL term.

Those reductions can be extended to any contexts, and so we have $M \rightarrow M'$ if there is a context C and a base reduction $M_0 \rightarrow M'_0$ such that $M = C(M_0)$ and $M' = C(M'_0)$. However, the scope of those contexts does not allow context reduction in sℓT. For reduction in sℓT, we use the last reduction rule.

Types. Types are usual types for intuitionistic linear logic enriched with some base types for booleans, integers and words. Base types are given by the grammar : $A := B \mid N \mid W$. Types are given by the grammar : $T := A \mid T \multimap T' \mid !T \mid T \otimes T'$

► **Definition 13** (Contexts and Type System). *Linear variables contexts* are denoted Γ and *global variables contexts* are denoted Δ . They are defined in the same way as in the previous part on the EAL-calculus. Typing judgments have the usual shape of dual contexts judgments $\pi \triangleleft \Gamma \mid \Delta \vdash M : T$. For such a proof π , and $i \in \mathbb{N}$, we define a *weight* $\omega_i(\pi) \in \mathbb{N}$.

$$\begin{array}{c}
(\lambda x.M) M' \rightarrow M[M'/x] \\
\text{let } x \otimes y = M \otimes M' \text{ in } N \rightarrow N[M/x][M'/y] \\
\text{ifn}(M, M') \text{ succ}(N) \rightarrow M N \\
\text{ifw}(M_0, M_1, M) \epsilon \rightarrow M \\
\text{iter}_W^!(M_0, !M_1, !M') \underline{w} \rightarrow !(M^w M') \\
\text{if}(M, M') \text{ ff} \rightarrow M' \\
[\lambda x_n \dots x_1.t](M_1, \dots, M_{n-1}, \underline{v}) \rightarrow [\lambda x_{n-1} \dots x_1.t[v/x_n]](M_1, \dots, M_{n-1}) \\
[\underline{v}]() \rightarrow \underline{v}
\end{array}
\left|
\begin{array}{c}
\text{let } !x = !M \text{ in } M' \rightarrow M'[M/x] \\
\text{ifn}(M, M') \text{ zero} \rightarrow M' \\
\text{iter}_N^!(M, !M') \underline{n} \rightarrow !(M^n M') \\
\text{ifw}(M_0, M_1, M) \text{ s}_i(N) \rightarrow M_i N \\
\text{if}(M, M') \text{ tt} \rightarrow M \\
\text{if } t \rightarrow t' \text{ in } \text{s}\ell\mathbb{T}, [t]() \rightarrow [t']()
\end{array}
\right.$$

■ **Figure 4** Base rules for sEAL.

$$\begin{array}{c}
\pi \triangleleft \frac{}{\Gamma, x : T \mid \Delta \vdash x : T} \quad \mu_n(\pi) = \mathbb{1}_0 \\
\pi \triangleleft \frac{}{\Gamma \mid \Delta, x : T \vdash x : T} \quad \mu_n(\pi) = \mathbb{1}_0 \\
\pi \triangleleft \frac{\sigma \triangleleft \Gamma, x : T \mid \Delta \vdash M : T'}{\Gamma \mid \Delta \vdash \lambda x.M : T \multimap T'} \quad \mu_n(\pi) = \mu_n(\sigma) + \mathbb{1}_0 \\
\pi \triangleleft \frac{\sigma \triangleleft \Gamma \mid \Delta \vdash M : T' \multimap T \quad \tau \triangleleft \Gamma' \mid \Delta \vdash M' : T'}{\Gamma, \Gamma' \mid \Delta \vdash M M' : T} \quad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0 \\
\pi \triangleleft \frac{\sigma \triangleleft \emptyset \mid \Delta \vdash M : T}{\Gamma \mid \Delta', [\Delta] \vdash !M : !T} \quad \mu_n(\pi) = (1, \mu_{n-1}(\sigma)) \\
\pi \triangleleft \frac{\sigma \triangleleft \Gamma' \mid \Delta \vdash M : !T \quad \tau \triangleleft \Gamma \mid \Delta, x : [T] \vdash M' : T'}{\Gamma, \Gamma' \mid \Delta \vdash \text{let } !x = M \text{ in } M' : T'} \quad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0 \\
\pi \triangleleft \frac{\sigma \triangleleft \Gamma \mid \Delta \vdash M : T \quad \tau \triangleleft \Gamma' \mid \Delta \vdash M' : T'}{\Gamma, \Gamma' \mid \Delta \vdash M \otimes M' : T \otimes T'} \quad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0 \\
\pi \triangleleft \frac{\sigma \triangleleft \Gamma' \mid \Delta \vdash M : T \otimes T' \quad \tau \triangleleft \Gamma, x : T, y : T' \mid \Delta \vdash M' : T''}{\Gamma, \Gamma' \mid \Delta \vdash \text{let } x \otimes y = M \text{ in } M' : T''} \quad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0
\end{array}$$

■ **Figure 5** Type and measure for generic constructors in sEAL.

► **Definition 14** (Measure and Depth). For all $k, n \in \mathbb{N}$, we note $\mu_n^k(\pi) = (\omega_k(\pi), \dots, \omega_n(\pi))$, with the convention that if $k > n$, then $\mu_n^k(\pi)$ is the null-vector. We write $\mu_n(\pi)$ to denote the vector $\mu_n^0(\pi)$. In the definitions given in the type system, instead of defining $\omega_i(\pi)$ for all i , we define $\mu_n(\pi)$ for all n , from which one can recover the weights. We will often call $\mu_n(\pi)$ the *measure* of the proof π . The *depth* of a proof (or a typed term) is the greatest integer i such that $\omega_i(\pi) \neq 0$. It is always defined for any proof.

The idea behind the definition of measure is to show that with a reduction step, this measure strictly decreases for the lexicographic order and we can control the growing of the weights. The rules are given on Figures 5, 6 and 7, and the rules for words and booleans can be found in the appendix 6.5.

The rules given in figure 5 represent the usual constructors in EAL. Those rules impose some restrictions in the use of variables similar to the one described in the previous section on classical EAL. Remark that the constructors for base types values such as **zero** and **succ** given in Figure 6 influence the weight only in position 1 and not 0 like the others constructors.

For the rule given by Figure 7, some explanations are necessary. The premise for t is a proof τ in $\text{s}\ell\mathbb{T}$. In this proof, we add on each base types A_i an index, more precisely an index variable a_i . There is here an abuse of notation, since in $\text{s}\ell\mathbb{T}$ there is no indexes on the

$$\begin{array}{l}
\pi \triangleleft \frac{}{\Gamma \mid \Delta \vdash \mathbf{zero} : \mathbf{N}} \quad \mu_n(\pi) = \mathbb{1}_1 \\
\pi \triangleleft \frac{\sigma \triangleleft \Gamma \mid \Delta \vdash M : \mathbf{N}}{\Gamma \mid \Delta \vdash \mathbf{succ}(M) : \mathbf{N}} \quad \mu_n(\pi) = \mu_n(\sigma) + \mathbb{1}_1 \\
\pi \triangleleft \frac{\sigma \triangleleft \Gamma \mid \Delta \vdash M : \mathbf{N} \multimap T \quad \tau \triangleleft \Gamma' \mid \Delta \vdash M' : T}{\Gamma, \Gamma' \mid \Delta \vdash \mathbf{ifn}(M, M') : \mathbf{N} \multimap T} \quad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0 \\
\pi \triangleleft \frac{\sigma \triangleleft \Gamma \mid \Delta \vdash M : !(T \multimap T) \quad \tau \triangleleft \Gamma' \mid \Delta \vdash M' : !T}{\Gamma, \Gamma' \mid \Delta \vdash \mathbf{iter}_N^!(M, M') : \mathbf{N} \multimap !T} \quad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0
\end{array}$$

■ **Figure 6** Type and measure for constructors on integers in sEAL.

$$\begin{array}{l}
\pi \triangleleft \frac{\forall i, (1 \leq i \leq k), \sigma_i \triangleleft \Gamma_i \mid \Delta \vdash M_i : A_i \quad \tau \triangleleft x_1 : A_1^{a_1}, \dots, x_k : A_k^{a_k} \vdash_{s\ell T} t : A^I}{\Gamma, \Gamma_1, \dots, \Gamma_k \mid \Delta \vdash [\lambda x_k \dots x_1. t](M_1, \dots, M_k) : A} \\
\mu_n(\pi) = \sum_{i=1}^k \mu_n(\sigma_i) + k(d(\omega(\tau)) + I) \cdot \mathbb{1}_0 + ((\omega(\tau) + I)[1/b_1] \cdots [1/b_l] + 1) \cdot \mathbb{1}_1 \\
\text{where } \{b_1, \dots, b_l\} = FV(\omega(\tau)) \cup FV(I).
\end{array}$$

■ **Figure 7** Typing rule and measure for the sℓT call in sEAL.

boolean type \mathbf{B} . So when $A_i = \mathbf{B}$, we just do not put any index on the type \mathbf{B} . The same goes for the type A , if A is the boolean type \mathbf{B} , then there is no index I , and we just replace in the measure I by 1. The previous section gives us a weight $\omega(\tau)$ for this proof in sℓT. Let us now comment on the definition of $\mu_n(\pi)$. The degrees of $\omega(\tau)$ and I influence the weight at position 0, and their values when all free variables are replaced by 1 influence the weight at position 1. Having the degree at position 0 will allow us the replacement of the arguments x_i by their values given by M_i , and the measure at position 1 will allow us to bound the number of reductions in sℓT and the size of the output. Furthermore, when $k = 0$, the term $[t]()$ influences only the weight at position 1, as constructors for base types.

3.3 Example: Testing Satisfiability of a Propositional Formula

Some examples of sEAL terms, like towers of exponentials, can be found in the appendix 6.6. We sketch here the construction of a term for deciding the SAT problem. Some other examples are given in the appendix, like testing the satisfiability of quantified boolean formulas (QBF_k) and deciding the subset-sum problem.

The term for SAT has type $\mathbf{N} \otimes \mathbf{W} \multimap !\mathbf{B}$ and given a formula on conjunctive normal form encoded in the type $\mathbf{N} \otimes \mathbf{W}$, it checks its satisfiability. The modality in front of the output $!\mathbf{B}$ shows that we used a non-polynomial computation, or more precisely an iteration in EAL, as expected of a term for satisfiability.

We encode formula in conjunctive normal form in the type $\mathbf{N} \otimes \mathbf{W}$, representing the number of distinct variables in the formula and the encoding of the formula by a word on the alphabet $\Sigma = \{0, 1, \#, |\}$. A literal is represented by the number of the corresponding variable written in binary and the first bit determines if the literal is positive or negative. Then $\#$ and $|$ are used as separator for literals and clauses.

For example, the formula $(x_1 \vee x_0 \vee x_2) \wedge (x_3 \vee \bar{x}_0 \vee \bar{x}_1) \wedge (\bar{x}_2 \vee x_0 \vee \bar{x}_3)$ is represented by $\underline{4 \otimes \#11\#10\#110\#111\#00\#01\#010\#10\#011}$

Intermediate terms in sℓT. For the sake of simplicity, we sometimes omit to describe all terms in \mathbf{ifw} or \mathbf{iterw} , especially for the letters $\#$ and $|$, when they are not important.

First, we can easily define a term $occ_a : W^I \multimap N^I$ that gives the number of occurrences of $a \in \Sigma$ in a word. In the appendix 6.3, some important terms are defined. We have a term $Cbinarytounary : N^I \multimap W^J \multimap N^I$ such that $Cbinarytounary \underline{n} \underline{w}$ computes the minimum between n and the unary representation of the binary integer w . We also have a term that gives the n^{th} bit (from right) of a binary word as a boolean $n^{th} : W^I \multimap N^I \multimap B$. And finally, we have a term $Extract_a : W^I \multimap W^I \otimes W^I$ that separates a word $w = w_0 a w_1$ in $w_0 \otimes w_1$ such that w_1 does not contain any a . This function will allow us to extract the last clause/literal of a word representing a formula.

A valuation is represented by a binary word with a length equal to the number of variable, such that the n^{th} bit of the word represents the boolean associated to the n^{th} variable.

We define a term $ClausestoBool : N^I \multimap W^J \multimap W^K \multimap B$ such that, given the number of variables, a valuation and a word representing a clause, this term outputs the truth value of this clause using the valuation.

$ClausestoBool = \lambda n, w_v, w_c. \text{let } w \otimes b = \text{itern}(\lambda w' \otimes b'. \text{let } w_0 \otimes w_1 = \text{Extract}_{\#} w' \text{ in } w_0 \otimes (or \ b' \ (LittoBool \ n \ w_v \ w_1)), w_c \otimes \text{ff}) \ (occ_{\#} w_c) \ \text{in } b$

With $LittoBool : N^I \multimap W^J \multimap W^K \multimap B$ converting a literal into the boolean given by the valuation : $LittoBool = \lambda n, w_v, w_l. \text{ifw}(\lambda w'. n^{th} \ w_v \ (Cbinarytounary \ n \ w'), \lambda w'. \text{not} \ (n^{th} \ w_v \ (Cbinarytounary \ n \ w')), \text{ff}) \ w_l$.

With this we can check if a clause is true given a certain valuation. We can define in the same way a term $FormulatoBool : N^I \multimap W^J \multimap W^K \multimap B$.

Testing all different valuations. Now all we have to do is to test this term with all possible valuations. If n is the number of variables, all possible valuations are described by all the binary integer from 0 to $2^n - 1$. Then we only need to use the iterator in $s\ell T$ with base type-inputs in order to check if one valuation satisfies the formula. We use a constructor for iteration defined in the appendix 6.3 : $REC(V, t) \underline{n} \rightarrow^* V \underline{n-1} (V \underline{n-2} (\dots (V \ \text{zero} \ t) \dots))$. We can then give the term for SAT :

$SAT = \lambda n \otimes w. \text{let } !r = \text{iter}_{N}^!(\lambda n_0 \otimes n_1. \text{succ}(n_0) \otimes [\text{double}](n_1)), !(\underline{0} \otimes \underline{1}) \ n \ \text{in}$
 $\text{let } !w_f = \text{coerc } w \ \text{in } !(\text{let } n \otimes \text{exp} = r \ \text{in } [\lambda n, \text{exp}, w_f.$

$REC(\lambda \text{val}, b. \text{or } b \ (FormulatoBool \ n \ (Cunarytobinary \ n \ \text{val}) \ w_f), \text{ff}) \ \text{exp}](n, \text{exp}, w_f))$.

The first iteration computes both 2^n and a copy of n . This technique is important as it shows that the linearity of EAL for base variables is not too constraining for the iteration. In the last line the term is a big “or” on the term $FormulatoBool$ applied to different valuations. And with that we have $SAT : N \otimes W \multimap !B$.

3.4 Subject Reduction and Measure

In this section, we show that we can bound the number of reduction steps of a typed term using the measure. This is done by showing that a reduction preserves some properties on the measure, and then give an explicit integer bound that will strictly decrease after a reduction. This proof uses the same logic as the one from [24]. The relation \mathcal{R} defined in the following is a generalization of the usual requirements exposed in elementary linear logic in order to control reductions.

Let us first express substitution lemmas for sEAL. There are 3 cases to consider, linear variables and discharged and non-discharged global variables.

► **Lemma 15 (Linear Substitution).** *If $\pi \triangleleft \Gamma_1, x : T' \mid \Delta \vdash M : T$ and $\sigma \triangleleft \Gamma_2 \mid \Delta \vdash M' : T'$ then we have a proof $\pi' \triangleleft \Gamma_1, \Gamma_2 \mid \Delta \vdash M[M'/x] : T$. Moreover, for all $n, \mu_n(\pi') \leq \mu_n(\pi) + \mu_n(\sigma)$.*

The proof comes from the fact that rules are multiplicative for Γ , and so x only appears in one of the premises for each rule. Thus the proof σ is used only once in the new proof π' .

► **Lemma 16** (General substitution). *If $\pi \triangleleft \Gamma \mid \Delta, x : T' \vdash M : T$ and $\sigma \triangleleft \emptyset \mid \Delta \vdash M' : T'$ and the number of occurrences of x in M is less than K , then we have a proof $\pi' \triangleleft \Gamma \mid \Delta \vdash M[M'/x] : T$. Moreover, for all n , $\mu_n(\pi') \leq \mu_n(\pi) + K \cdot \mu_n(\sigma)$.*

This time, the non-linearity of the variable x induces a duplication of the proof σ , that's why the measure $\mu_n(\sigma)$ is also duplicated.

► **Lemma 17** (Discharged substitution lemma). *If $\pi \triangleleft \Gamma \mid \Delta', [\Delta], x : [T'] \vdash M : T$ and $\sigma \triangleleft \emptyset \mid \Delta \vdash M' : T'$ then we have a proof $\pi' \triangleleft \Gamma \mid \Delta', [\Delta] \vdash M[M'/x] : T$. Moreover, for all n , $\mu_n(\pi') \leq (\omega_0(\pi), (\mu_n^1(\pi) + \omega_1(\pi) \cdot \mu_{n-1}(\sigma)))$.*

The proof of this lemma relies directly on the previous one. Indeed, a variable with a discharged type can be used only after crossing a (!-Intro) rule, and then the upper bound on $\mu_n(\pi')$ comes from the previous lemma since the number of occurrences of x in M is less than $\omega_1(\pi)$.

Then, let us give two important definition, t_α and \mathcal{R} , in order to derive the upper bound on the number of reduction in sEAL.

► **Definition 18** (t_α). We define a family of tower functions $t_\alpha(x_1, \dots, x_n)$ on vectors of integers by induction on n , where we assume $\alpha \geq 1$ and $x_i \geq 2$ for all i :

$$t_\alpha() = 0 \text{ and } t_\alpha(x_1, \dots, x_n) = (\alpha \cdot x_n)^{2^{t_\alpha(x_1, \dots, x_{n-1})}} \text{ for } n \geq 1$$

► **Definition 19** (\mathcal{R}). We define a relation on vectors denoted \mathcal{R} . Intuitively, we want $\mathcal{R}(\mu, \mu')$ to express the fact that a proof of measure μ has been reduced to a proof of measure μ' . Let $\mu, \mu' \in \mathbb{N}^{n+1}$. We have $\mathcal{R}(\mu, \mu')$ if and only if :

1. $\mu \geq \tilde{2}$ and $\mu' \geq \tilde{2}$.
2. $\mu' \leq_{lex} \mu$. Thus, we write $\mu = (\omega_0, \dots, \omega_n)$ and $\mu' = (\omega_0, \dots, \omega_{i_0-1}, \omega'_{i_0}, \dots, \omega'_n)$, with $\omega_{i_0} > \omega'_{i_0}$.
3. There exists $d \in \mathbb{N}, 1 \leq d \leq (\omega_{i_0} - \omega'_{i_0})$ such that $\forall j > i_0, \omega'_j \leq \omega_j \cdot (\omega_{i_0+1})^{d-1}$

The first condition with $\tilde{2}$, that can also be seen in the definition of t_α , makes calculation easier, since with this condition, exponentials and multiplications conserve the strict order between integers. This does not harm the proof, since we can simply add $\tilde{2}$ to each vector we will consider. We can then connect those two definitions :

► **Theorem 20.** *Let $\mu, \mu' \in \mathbb{N}^{n+1}$ and $\alpha \geq n, \alpha \geq 1$. If $\mathcal{R}(\mu, \mu')$ then $t_\alpha(\mu') < t_\alpha(\mu)$*

It shows that if we want to ensure that a certain integer defined with t_α strictly decreases for a reduction, it is sufficient to work with the relation \mathcal{R} .

We can now state the subject reduction of sEAL and we show that the measure allows us to construct a bound on the number of reductions.

► **Theorem 21.** *Let $\tau \triangleleft \Gamma \mid \Delta \vdash M_0 : T$ and $M_0 \rightarrow M_1$. Let α be an integer equal or greater than the depth of τ . Then there is a proof $\tau' \triangleleft \Gamma \mid \Delta \vdash M_1 : T$ such that $\mathcal{R}(\mu_\alpha(\tau) + \tilde{2}, \mu_\alpha(\tau') + \tilde{2})$. Moreover, the depth of τ' is smaller than the depth of τ .*

The proof uses the substitution lemma for reductions in which substitution appears, and for the others constructors, one can see that the measure given in the type system for sEAL is following this idea of the relation \mathcal{R} , e.g., in the reduction $[\lambda x_n \dots x_1.t](M_1, \dots, M_{n-1}, \underline{v}) \rightarrow [\lambda x_{n-1} \dots x_1.t[\underline{v}/x_n]](M_1, \dots, M_{n-1})$, the degree that appears at position 0 is here to compensate the growing of the measure at position 1. Now using the previous results, we can easily conclude our bound on the number of reductions.

► **Theorem 22.** *Let $\pi \triangleleft \Gamma \mid \Delta \vdash M : T$. Denote $\alpha = \max(\text{depth}(\pi), 1)$, then $t_\alpha(\mu_\alpha(\pi) + \tilde{2})$ is a bound on the number of reductions from M .*

4 Complexity Results: Characterization of 2k-EXP and 2k-FEXP

Now that we have proved the preceding theorem, we have obtained a bound on the number of reduction steps from a term. More precisely, this bound shows that between two consecutive weights ω_{i+1} and ω_i , there is a difference of 2 in the height of the tower of exponentials. This will allow us to give a characterization of the classes 2k-EXP for $k \geq 0$, and each modality “!” in the type of a term will induce a difference of 2 in the height of the tower of exponentials. With exactly the same method, we also have a characterization of the classes 2k-FEXP for $k \geq 0$.

Restricted Reductions and Values. First, we show that the precedent bound on the number of reductions in Theorem 22 can be improved. Indeed, if we restrict the possible reductions, we obtain a more precise bound.

► **Definition 23** (Reductions up to a Certain Depth). For $i \in \mathbb{N}$, we define the i -reductions, that we note \rightarrow_i :

- $\forall i \geq 1, [t]() \rightarrow_i [t']()$ if $t \rightarrow t'$ in $\text{s}\ell\mathcal{T}$. Moreover, $[y]() \rightarrow_i y$
- For the other base reductions $M \rightarrow M'$, we have $\forall i \in \mathbb{N}, M \rightarrow_i M'$
- For all $i \in \mathbb{N}$, if $M \rightarrow_i M'$ then $!M \rightarrow_{i+1} !M'$
- For all others constructors, the index i stays the same. For example for the application, we have for all $i \in \mathbb{N}$, if $M \rightarrow_i M'$ then $M N \rightarrow_i M' N$.

Now, we can find a more precise measure to bound the number of i -reductions. The proof is very similar to the proof of theorem 21 and 22.

► **Lemma 24.** *Let $i \in \mathbb{N}$, $\tau \triangleleft \Gamma \mid \Delta \vdash M_0 : T$ and $M_0 \rightarrow_i M_1$. Then there is a proof $\tau' \triangleleft \Gamma \mid \Delta \vdash M_1 : T$ such that $\mathcal{R}(\mu_i(\tau) + \tilde{2}, \mu_i(\tau') + \tilde{2})$*

► **Theorem 25.** *Let $\pi \triangleleft \Gamma \mid \Delta \vdash M : T$ and $\alpha = \max(i, 1)$. Then $t_\alpha(\mu_i(\pi) + \tilde{2})$ is a bound on the number of i -reductions from M .*

► **Definition 26** (Values Associated to Restricted Reductions). We give the form of closed normal terms for i -reductions. For that, we define for all $i \in \mathbb{N}$, *closed i -values* V^i by the following grammar :

$$\begin{aligned}
 V^0 &:= M \\
 \forall i \geq 1, V^i &:= \lambda x.M \mid !V^{i-1} \mid V_0^i \otimes V_1^i \mid \mathbf{zero} \mid \mathbf{succ}(V^i) \mid \mathbf{ifn}(V_0^i, V_1^i) \mid \mathbf{iter}_N^!(V_0^i, V_1^i) \mid \\
 \mathbf{tt} \mid \mathbf{ff} \mid \mathbf{if}(V_0^i, V_1^i) \mid \epsilon \mid \mathbf{s}_i(V^i) \mid \mathbf{ifw}(V_0^i, V_1^i, V_2^i) \mid \mathbf{iter}_W^!(V_0^i, V_1^i, V_2^i).
 \end{aligned}$$

We can then prove the following lemma:

► **Lemma 27.** *Let M be a term. If M is closed and has a typing derivation then, for all $i \in \mathbb{N}$, if M is normal for i -reductions then M is a i -value V^i .*

The proof can be found in [7]. From the previous results, we now have that, from a typed term M , we can reach the normal form for i -reductions for M in less than $t_i(\mu_i(\pi) + \tilde{2})$ reductions, and this form is an i -value.

A Characterization of $2k$ -EXP. Now, we sketch how the type $!W \multimap^{k+1} B$ can characterize the class $2k$ -EXP for $k \geq 0$. Recall that 2_k^x is defined by $2_0^x = x$ and $2_{k+1}^x = 2^{2_k^x}$. The class k -EXP is the class of problem solvable by a Turing machine that works in time $2_k^{p(n)}$ on an entry of size n , where p is a polynomial. First we show that the number of reductions for such a term is bounded by a tower of exponentials of height $2k$.

► **Lemma 28.** *Let $\pi \triangleleft \cdot \mid \cdot \vdash t : !W \multimap^{k+1} B$. Let w be a word of size $|w|$. We can compute the result of $t \ !w$ in less than a $2k$ -exponential tower in the size of w .*

Observe that the result of this computation is of type $!^{k+1} B$, and a $(k+2)$ -value of type $!^{k+1} B$ is exactly of the form $!^{k+1} \mathbf{t} \mathbf{t}$ or $!^{k+1} \mathbf{f} \mathbf{f}$. So it is enough to only consider $(k+2)$ -reductions to compute the result, by lemma 27. The measure μ_n of $t \ !w$ is $\mu_n = \mu_n(\pi) + 2 \cdot \mathbf{1}_0 + |w| \cdot \mathbf{1}_2$. By theorem 25, we can bound the number of reductions from $t \ !w$ by $t_{k+2}(\mu_{k+2} + \tilde{2})$. By definition, in $t_{k+2}(\mu_{k+2} + \tilde{2})$, we can see that the weight at position 2, where the size of w appears, is at height $2k$. This concludes the proof of lemma 28.

Now we have to prove that we can simulate a Turing-machine in our calculus. This proof is usual in implicit complexity [5, 2]. A sketch of this proof can be found in the appendix, section 6.7. With this, using the lemma 28, we obtain the following theorem

► **Theorem 29.** *Terms of type $!W \multimap^{k+1} B$ characterize the class $2k$ -EXP.*

As explained previously, this theorem can be expanded for the classes $2k$ -FEXP, that is the class of function from words to words that can be computed by a one-tape Turing machine running with a time at most $2_{2k}^{p(|w|)}$ on a word w . For a more precise definitions of such classes, see [5]. This characterization uses the same proof by replacing $!W \multimap^{k+1} B$ by $!W \multimap^{k+1} W$.

Moreover, in EAL, we can characterize k -EXP with the type $!W \multimap^{k+1} B$. The difference with sEAL can be explained by the fact that in EAL, in the type $N \multimap N$ we only have polynomials of degree 1 (polynomials in general have the type $!N \multimap !N$), whereas in our case, polynomials have the type $N \multimap N$.

5 Conclusion

We believe that our main contribution is to define a new methodology to combine size-based and level-based type systems, which we have illustrated here with the example of sℓT and EAL, but we think is of more general interest. In the present particular setting of sEAL we can wonder which enrichment we can add to EAL while keeping the properties, for instance: new data-types (lists, trees), the possibility to freely duplicate base types ... We should also investigate type inference techniques, by drawing inspiration from linear dependent types [11, 3] and EAL [8]. But more importantly we would like to explore to which other systems we could apply this methodology:

- First can we define a similar system in which we could move up one level of $!$ and stay in polynomial time? We conjecture that this could be obtained with EAL but replacing sℓT with a system of indexes of degree at most 1, instead of polynomial indexes. In this case we believe that the type $!W \multimap !B$ would correspond to PTIME. An alternative choice could be to use a Non-size-increasing types system [19] instead of sℓT.
- Can we define a system in which all levels stay in FPTIME? Beside the condition on indexes (degree at most 1) we would also need for that purpose to replace EAL with another level-based system. Light linear logic [15] is a natural candidate, but we would need to find a measure-based argument for its complexity bound, which is a challenging objective.

References

- 1 Martin Avanzini and Ugo Dal Lago. Automating sized-type inference for complexity analysis. *PACMPL*, 1(ICFP):43:1–43:29, 2017.
- 2 Patrick Baillot. On the expressivity of elementary linear logic: Characterizing ptime and an exponential time hierarchy. *Information and Computation*, 241:3–31, 2015.
- 3 Patrick Baillot, Gilles Barthe, and Ugo Dal Lago. Implicit computational complexity of subrecursive definitions and applications to cryptographic proofs (long version). Research report, ENS Lyon, 2015. URL: <https://hal.archives-ouvertes.fr/hal-01197456>.
- 4 Patrick Baillot and Ugo Dal Lago. Higher-order interpretations and program complexity. *Information and Computation*, 248:56–81, 2016.
- 5 Patrick Baillot, Erika De Benedetti, and Simona Ronchi Della Rocca. Characterizing polynomial and exponential complexity classes in elementary lambda-calculus. In *IFIP International Conference on Theoretical Computer Science*, pages 151–163. Springer, 2014.
- 6 Patrick Baillot, Marco Gaboardi, and Virgile Mogbil. A polytime functional language from light linear logic. In *European Symposium on Programming*, pages 104–124. Springer, 2010.
- 7 Patrick Baillot and Alexis Ghyselen. Combining linear logic and size types for implicit complexity (long version). hal-01687224, [Research Report], 2018.
- 8 Patrick Baillot and Kazushige Terui. A feasible algorithm for typing in elementary affine logic. In *TLCA*, volume 5, pages 55–70. Springer, 2005.
- 9 Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the polytime functions. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 283–293. ACM, 1992.
- 10 Guillaume Bonfante, Jean-Yves Marion, and Jean-Yves Moyen. Quasi-interpretations a way to control resources. *Theoretical Computer Science*, 412(25):2776–2796, 2011.
- 11 Ugo Dal Lago and Barbara Petit. The geometry of types. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL’13, Proceedings*, pages 167–178. ACM, 2013.
- 12 Ugo Dal Lago and Barbara Petit. Linear dependent types in a call-by-value scenario. *Science of Computer Programming*, 84:77–100, 2014.
- 13 Ugo Dal Lago and Paolo Parisen Toldin. A higher-order characterization of probabilistic polynomial time. In *International Workshop on Foundational and Practical Aspects of Resource Analysis*, pages 1–18. Springer, 2011.
- 14 Vincent Danos and Jean-Baptiste Joinet. Linear logic and elementary time. *Information and Computation*, 183(1):123–137, 2003.
- 15 Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.
- 16 Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate amortized resource analysis. In *38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL’11, Proceedings*. ACM, 2011.
- 17 Jan Hoffmann and Martin Hofmann. Amortized resource analysis with polynomial potential. In *19th Euro. Symp. on Prog.(ESOP’10)*, pages 287–306. Springer, 2010.
- 18 Martin Hofmann. A mixed modal/linear lambda calculus with applications to bellantoni-cook safe recursion. In *International Workshop on Computer Science Logic*, pages 275–294. Springer, 1997.
- 19 Martin Hofmann. Linear types and non-size-increasing polynomial time computation. *Information and Computation*, 183(1):57–85, 2003.
- 20 Martin Hofmann and Steffen Jost. Static prediction of heap space usage for first-order functional programs. In *30th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL’11, Proceedings*, pages 185–197. ACM, 2003.

- 21 John Hughes, Lars Pareto, and Amr Sabry. Proving the correctness of reactive systems using sized types. In *Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 410–423, 1996.
- 22 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 ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL, 2010*, pages 223–236, 2010.
- 23 Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. *Logical Methods in Computer Science*, 8(4), 2011.
- 24 Antoine Madet and Roberto M Amadio. An elementary affine λ -calculus with multithreading and side effects. In *International Conference on Typed Lambda Calculi and Applications*, pages 138–152. Springer, 2011.
- 25 John Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 725–733. IEEE, 1998.
- 26 Kazushige Terui. Light affine lambda calculus and polytime strong normalization. In *Logic in Computer Science, 2001. Proceedings. 16th Annual IEEE Symposium on*, pages 209–220. IEEE, 2001.

6 Appendix

6.1 Type System for Words and boolean in $s\ell T$

$$\begin{array}{l}
\pi \triangleleft \frac{}{\Gamma \vdash \epsilon : W^I} \qquad \omega(\pi) = 0 \\
\pi \triangleleft \frac{\sigma \triangleleft \Gamma \vdash t : W^J \quad J+1 \leq I}{\Gamma \vdash \mathbf{s}_i(t) : W^I} \qquad \omega(\pi) = \omega(\sigma) \\
\pi \triangleleft \frac{\sigma_1 \triangleleft \Gamma_1, d\Gamma \vdash t_1 : W^I \multimap D \quad \sigma_0 \triangleleft \Gamma_0, d\Gamma \vdash t_0 : W^I \multimap D \quad \sigma \triangleleft \Gamma, d\Gamma \vdash t' : D}{\Gamma_0, \Gamma_1, \Gamma, d\Gamma \vdash \mathbf{ifw}(t_0, t_1, t') : W^I \multimap D} \qquad \omega(\pi) = \omega(\sigma_1) + \omega(\sigma_0) + \omega(\sigma) + 1 \\
\pi \triangleleft \frac{D \sqsubset E \quad E \sqsubset E[a+1/a] \quad \sigma_1 \triangleleft d\Gamma \vdash V_1 : D \multimap D[a+1/a] \quad E[I/a] \sqsubset F \quad \sigma_0 \triangleleft d\Gamma \vdash V_0 : D \multimap D[a+1/a] \quad \sigma \triangleleft \Gamma, d\Gamma \vdash t : D[1/a]}{\Gamma, d\Gamma \vdash \mathbf{iterw}(V_0, V_1, t) : N^I \multimap F} \qquad \omega(\pi) = I + \omega(\sigma) + I \cdot (\omega(\sigma_1) + \omega(\sigma_0))[I/a] \\
\pi \triangleleft \frac{}{\Gamma \vdash \mathbf{tt}(or \mathbf{ff}) : B} \qquad \omega(\pi) = 0 \\
\pi \triangleleft \frac{\sigma_1 \triangleleft \Gamma_1, d\Gamma \vdash t : D \quad \sigma_2 \triangleleft \Gamma_2, d\Gamma \vdash t' : D}{\Gamma_1, \Gamma_2, d\Gamma \vdash \mathbf{if}(t, t') : B \multimap D} \qquad \omega(\pi) = \omega(\sigma_1) + \omega(\sigma_2) + 1
\end{array}$$

6.2 Some Intermediate Lemmas for the Subject Reduction

Index Variable Substitution and Subtyping. We give some intermediate lemmas in order to prove the subject reduction theorem. Some intuition and more detailed proofs can be found in the technical report [7]

► **Lemma 30** (Weakening). *Let Δ, Γ be disjoint typing contexts, and $\pi \triangleleft \Gamma \vdash t : D$ then we have a proof $\pi' \triangleleft \Gamma, \Delta \vdash t : D$ with $\omega(\pi) = \omega(\pi')$.*

► **Lemma 31** (Index substitution). *Let I be an index.*

1. *Let J_1, J_2 be indexes such that $J_1 \leq J_2$ then $J_1[I/a] \leq J_2[I/a]$*
2. *Let D, D' be types such that $D \sqsubset D'$ then $D[I/a] \sqsubset D'[I/a]$*
3. *If $\pi \triangleleft \Gamma \vdash t : D$ then $\pi[I/a] \triangleleft \Gamma[I/a] \vdash t : D[I/a]$*
4. *$\omega(\pi[I/a]) = \omega(\pi)[I/a]$*

► **Lemma 32** (Monotonic index substitution). *Take J_1, J_2 such that $J_1 \leq J_2$.*

1. *Let I be an index, then $I[J_1/a] \leq I[J_2/a]$.*
2. *For any proof π , $\omega(\pi[J_1/a]) \leq \omega(\pi[J_2/a])$.*
3. *Let E be a type. If $E \sqsubset E[a+1/a]$ then $E[J_1/a] \sqsubset E[J_2/a]$ and if $E[a+1/a] \sqsubset E$ then $E[J_2/a] \sqsubset E[J_1/a]$*

► **Lemma 33.** *If $\pi \triangleleft \Gamma, d\Gamma \vdash V : U$ then we have a proof $\pi' \triangleleft d\Gamma \vdash V : U$ with $\omega(\pi) = \omega(\pi')$. Moreover, $\omega(\pi') \leq 1$.*

Term Substitution Lemma. In order to prove the subject reduction of the calculus, we explicit what happens during a substitution of a value in a term. There are two cases, first the substitution of variables with base types, that is to say duplicable variables, and then the substitution of variables with a non-base type for which the type system imposes linearity.

► **Lemma 34** (Value Substitution). *If $\pi \triangleleft \Gamma_1, d\Gamma, x : D' \vdash t : D$ and $\sigma \triangleleft \Gamma_2, d\Gamma \vdash V : D'$ then we have a proof $\pi' \triangleleft \Gamma_1, \Gamma_2, d\Gamma \vdash t[V/x] : D$. Moreover, if D' is a base type then $\omega(\pi') \leq \omega(\pi)$. Otherwise, $\omega(\pi') \leq \omega(\pi) + \omega(\sigma)$.*

This is proved by induction on π . For the base type case, we use lemma 33 to show that Γ_2 can be ignored, and then as $d\Gamma$ is duplicable, the proof is rather direct. For the non-base case, in multiplicative rules such as application and *if*, the property holds by the fact that x only appears in one of the premises, and so $\omega(\sigma)$ appears only once in the total weight.

6.3 Examples in sℓT

Reverse of a word, and mirror iterator. We can compute the reverse of a word $(a_0 a_1 \dots a_n \mapsto a_n \dots a_1 a_0)$ with the term $rev = \mathbf{iterw}(\lambda w. \mathbf{s}_0(w), \lambda w. \mathbf{s}_1(w), \epsilon) : \mathbf{W}^I \multimap \mathbf{W}^I$.

Now we define $ITERW(V_0, V_1, t) = \lambda w. (\mathbf{iterw}(V_0, V_1, t)(rev w))$ that is the iterator on words with the right order $(ITERW(V_0, V_1, t) \mathbf{s}_{i_1}(\mathbf{s}_{i_2}(\dots \mathbf{s}_{i_n}(\epsilon) \dots))) \rightarrow^* V_{i_1}(V_{i_2}(\dots V_{i_n}(t) \dots))$. The typing rule we can make for this constructor is exactly the same as the one for \mathbf{iterw} .

Iterator with base type argument. We show that for integers we can construct a term $REC(V, t)$ such that $REC(V, t) \underline{n} \rightarrow^* V \underline{n-1} (V \underline{n-2} (\dots (V \mathbf{zero} t) \dots))$.
 $REC(V, t) = \lambda n. \mathbf{let} \ x \otimes y = (\mathbf{intern}(\lambda r \otimes n'. (V n' r) \otimes \mathbf{succ}(n'), t \otimes \mathbf{zero}) \ n) \ \mathbf{in} \ x$

We can give this constructor a typing rule close to the one for the iteration, with an additional argument in the step term of type \mathbf{N}^a . This constructor can also be defined for words.

Addition for unary words. The addition can be written in sℓT. We give a sketch of the proof tree.

$$\frac{\frac{\frac{\frac{\frac{\mathbf{N}^{I+a} \sqsubset \mathbf{N}^{I+a}}{x : \mathbf{N}^I, y : \mathbf{N}^{I+a} \vdash y : \mathbf{N}^{I+a} \quad I+a+1 \leq I+a+1}}{x : \mathbf{N}^I, y : \mathbf{N}^{I+a} \vdash \mathbf{succ}(y) : \mathbf{N}^{I+a+1}}}{x : \mathbf{N}^I \vdash \lambda y. \mathbf{succ}(y) : \mathbf{N}^{I+a} \multimap \mathbf{N}^{I+a}[a+1/a]} \quad \dots}{x : \mathbf{N}^I \vdash \mathbf{intern}(\lambda y. \mathbf{succ}(y), x) : \mathbf{N}^J \multimap \mathbf{N}^{I+J}}}{\pi_{add}(I, J) \triangleleft \cdot \vdash \lambda x. \mathbf{intern}(\lambda y. \mathbf{succ}(y), x) : \mathbf{N}^I \multimap \mathbf{N}^J \multimap \mathbf{N}^{I+J}}$$

$add = \lambda x. \mathbf{intern}(\lambda y. \mathbf{succ}(y), x), \pi_{add}(I, J) \triangleleft \cdot \vdash add : \mathbf{N}^I \multimap \mathbf{N}^J \multimap \mathbf{N}^{I+J}$. And the rules give us, for two integers n and m , $add \underline{n} \underline{m} \rightarrow \mathbf{intern}(\lambda y. \mathbf{succ}(y), \underline{n}) \underline{m} \rightarrow^* (\lambda y. \mathbf{succ}(y))^m \underline{n} \rightarrow^* \underline{n+m}$. The weight of this term is $\omega_{add}(I, J) = 1+J+1+J \cdot (1+1)[J/a] = 3J+2$

Addition on binary integers. Now, we define some terms working on integers written in binary, with type W^I . First, we define an addition on binary integers in $s\mathcal{L}\mathcal{T}$ with a control on the number of bits. More precisely, we give a term $Cadd : N^I \multimap W^{J_1} \multimap W^{J_2} \multimap W^I$ such that $Cadd\ n\ w_1\ w_2$ outputs the least significant n bits of the sum $w_1 + w_2$. For example, $Cadd\ 3\ 101\ 110 = 011$, and $Cadd\ 5\ 101\ 110 = 01011$. This will usually be used with a n greater than the expected number of bits, the idea being that those extra 0 can be useful for some other programs. The term follows the usual idea for addition: the result is computed bit by bit, and we keep track of the carry. For simplification, we do not give an explicit term but we show that we have to use conditionals and work on each cases one by one.

$Cadd = \lambda n, w_1, w_2. \mathbf{let}\ c' \otimes r' \otimes w'_1 \otimes w'_2 = \mathbf{itern}(\lambda c \otimes r \otimes w \otimes w'. \mathbf{match}\ c, w, w' \text{ with } (\mathbf{ff}, \epsilon, \epsilon) \mapsto \mathbf{ff} \otimes \mathbf{s}_0(r) \otimes \epsilon \otimes \epsilon \mid \dots \mid (\mathbf{tt}, \mathbf{s}_1(v), \mathbf{s}_1(v')) \mapsto \mathbf{tt} \otimes \mathbf{s}_1(r) \otimes v \otimes v', \mathbf{ff} \otimes \epsilon \otimes (\mathbf{rev}\ w_1) \otimes (\mathbf{rev}\ w_2))\ n \text{ in } r.$

For the typing of this term, we use in the iteration the type $B \otimes W^a \otimes W^{J_1} \otimes W^{J_2}$, with c representing the carry, r the current result, and w, w' the binary integers that we read from right to left.

Unary integers to binary integers. We define a term $Cunarytobinary : N^I \multimap N^J \multimap W^I$ such that on the input n, n' , this term computes the least n significant bit of the representation of n' in binary : $Cunarytobinary = \lambda n. \mathbf{itern}(\lambda w. Cadd\ n\ w\ (\mathbf{s}_1(\epsilon)), Cadd\ n\ \epsilon\ \epsilon)$

Binary integers to unary integers. We would like a way to compute the unary integer for a given binary integer. However, this function is exponential in the size of its input, so it is impossible to write such a function in $s\mathcal{L}\mathcal{T}$. Nevertheless, given an additional information bounding the size of this unary word, we can give a term $Cbinarytounary : N^I \multimap W^J \multimap N^I$ such that on an input n, w this term computes the minimum between n and the unary representation of w . First we describe a term $min : N^I \multimap N^J \multimap N^I$. $min = \lambda n, n'. \mathbf{let}\ r_0 \otimes n_0 = (\mathbf{itern}(\lambda r_1 \otimes n_1. \mathbf{ifn}(\lambda p. \mathbf{succ}(r_1) \otimes p, r_1 \otimes \mathbf{zero})\ n_1, \mathbf{zero} \otimes n')\ n) \text{ in } r_0$

In order to type this term, we use in the iteration the type $N^a \otimes N^J$. Remark that this term allows us to erase the index J . Now that we have this term, we can define the following term $Cbinarytounary = \lambda n. \mathbf{iterw}(\lambda n'. min\ n\ (\mathbf{mult}\ n' \ \underline{2}), \lambda n'. min\ n\ \mathbf{succ}(\mathbf{mult}\ n' \ \underline{2}), \mathbf{zero})$

Some examples on words. We can define a term that gives the n^{th} bit (from right) of a binary word as a boolean :

$n^{th} = \lambda w, n. \mathbf{ifw}(\lambda w'. \mathbf{ff}, \lambda w'. \mathbf{tt}, \mathbf{ff}) ((\mathbf{itern}(\mathbf{pred}, \mathbf{rev}\ w))\ n) : W^I \multimap N^I \multimap B$, with $\mathbf{pred} : W^I \multimap W^I = \mathbf{ifw}(\lambda w. w, \lambda w. w, \epsilon)$.

We can also define a term of type $W^I \multimap W^I \otimes W^I$ that separates a word $w = w_0 a w_1$ in $w_0 \otimes w_1$ such that w_1 does not contain any a .

$Extract_a = \lambda w. \mathbf{let}\ b' \otimes w'_0 \otimes w'_1 = \mathbf{ITERW}(V_0, V_1, V_{\#}, V_{|}, V_{\epsilon})\ w \text{ in } w'_0 \otimes w'_1$
with $V_a = \lambda b \otimes w_0 \otimes w_1. \mathbf{if}(\mathbf{tt} \otimes s_a(w_0) \otimes w_1, \mathbf{tt} \otimes w_0 \otimes w_1)\ b$
 $\forall c \neq a, V_c = \lambda b \otimes w_0 \otimes w_1. \mathbf{if}(\mathbf{tt} \otimes s_c(w_0) \otimes w_1, \mathbf{ff} \otimes w_0 \otimes s_c(w_1))\ b$
 $V_{\epsilon} = \mathbf{ff} \otimes \epsilon \otimes \epsilon$

For the intuition on this term, the boolean b' used in the iteration is a boolean that indicates if we have already read the letter “a” previously.

States. A state is a tensor of boolean for which we can have a match case. More precisely, for $n \in \mathbb{N}^*$, we define by induction the type $B^n = B \otimes B^{n-1}$ with $B^1 = B$. B^n describes states of size n . In the following, we will ignore the term for the associativity of the tensor.

In order to precise the decomposition, we will note $\text{let } x_D \otimes y_{D'} = t \text{ in } t'$ to explicit the decomposition when it is ambiguous.

There are 2^n base states of size n , given by the 2^n possibilities of associating n times tt or ff . Moreover, there is a constructor to do a match-case on those states, $\text{case}_n(t_{0^n}, \dots, t_{1^n})$. We will consider in order to simplify the notations that those indexes are the integers from 0 to $2^n - 1$ written in binary, with 1 referring to tt . We define it by induction, and give the typing.

For $n = 1$, $\text{case}_1(t_0, t_1) = \text{if}(t_1, t_0)$ and for $n \geq 0$:
 $\text{case}_{n+1}(t_{0^{n+1}}, \dots, t_{1^{n+1}}) = \lambda s. \text{let } s'_{\mathbb{B}^n} \otimes x_{\mathbb{B}} = s \text{ in } \text{case}_n(t'_{0^n}, \dots, t'_{1^n}) s'$ with, for all boolean word i , $t'_i = \text{if}(t_{i1}, t_{i0}) x$.

With this definition, by noting $i = b_1 \dots b_n$ the state and the boolean word, we have $\text{case}_n(t_{0^n}, \dots, t_{1^n}) (b_1 \dots b_n) \rightarrow^* \text{case}_{n-1}(t_{0^{n-1}b_n}, \dots, t_{1^{n-1}b_n}) (b_1 \dots b_{n-1}) \rightarrow^* t_i$

Moreover, we can deduce this rule:

$$\frac{\forall i, 0 \leq i \leq 2^n - 1, \Gamma_i, d\Gamma \vdash t_i : D}{\Gamma_0, \dots, \Gamma_{2^n-1}, d\Gamma \vdash \text{case}_n(t_0, \dots, t_{2^n-1}) : \mathbb{B}^n \multimap D}$$

6.4 Adding Polynomial Time Functions in EAL

Here we explain very informally how we can add polynomial time functions in the calculus defined in [24], keeping the same kind of proof relying on the measure.

Suppose given a function f from integers to integers. We define a new constructor f in the classical EAL-calculus, and a new reduction rule $f \underline{n} \rightarrow f(n)$, saying that f applied to the encoding of the integer n is reduced to the encoding of the integer $f(n)$. We add a cost to this reduction, depending on the integer n , that we call $C_f(n)$. We give a typing rule for this constructor, f has type $\mathbb{N} \multimap \mathbb{N}$.

If this function f is a polynomial time computable function, we can bound the cost function $C_f(n)$ by a polynomial function $(n+2)^d$ for a certain d , and we can also bound the size of $f(n)$ by the cost, and so $f(n) \leq (n+2)^d$. Now if we look at the reduction rule, if we call $\mu(f)$ the measure for f , we go from $\mu(f) + (1, n+1)$ to $(0, (n+2)^d)$, if we want to take in consideration the cost, we can add it in the measure, and suppose that in the right part of the reduction we have the measure $(0, 2(n+2)^d)$. Now, see that if $\mu(f) = (d, 1)$, this reduction follows the relation \mathcal{R} defined in section 3, and with that we can deduce that this construction works with the measure.

6.5 Type System for Words and Boolean in sEAL

$$\begin{array}{l} \pi \triangleleft \frac{}{\Gamma \mid \Delta \vdash \epsilon : \mathbb{W}} \quad \mu_n(\pi) = \mathbb{1}_1 \\ \pi \triangleleft \frac{\sigma \triangleleft \Gamma \mid \Delta \vdash M : \mathbb{W}}{\Gamma \mid \Delta \vdash \mathbf{s}_i(M) : \mathbb{W}} \quad \mu_n(\pi) = \mu_n(\sigma) + \mathbb{1}_1 \\ \pi \triangleleft \frac{\sigma_1 \triangleleft \Gamma_1 \mid \Delta \vdash M_1 : \mathbb{W} \multimap T \quad \sigma_0 \triangleleft \Gamma_0 \mid \Delta \vdash M_0 : \mathbb{W} \multimap T \quad \sigma \triangleleft \Gamma \mid \Delta \vdash M' : T}{\Gamma_0, \Gamma_1, \Gamma \mid \Delta \vdash \text{ifw}(M_0, M_1, M') : \mathbb{W} \multimap T} \quad \mu_n(\pi) = \mu_n(\sigma_0) + \mu_n(\sigma_1) + \mu_n(\sigma) + \mathbb{1}_0 \\ \pi \triangleleft \frac{\sigma_1 \triangleleft \Gamma_1 \mid \Delta \vdash M_1 : !(T \multimap T) \quad \sigma_0 \triangleleft \Gamma_0 \mid \Delta \vdash M_0 : !(T \multimap T) \quad \sigma \triangleleft \Gamma \mid \Delta \vdash M : !T}{\Gamma_0, \Gamma_1, \Gamma \mid \Delta \vdash \text{iter}_W^!(M_0, M_1, M) : \mathbb{W} \multimap !T} \quad \mu_n(\pi) = \mu_n(\sigma_0) + \mu_n(\sigma_1) + \mu_n(\sigma) + \mathbb{1}_0 \\ \pi \triangleleft \frac{}{\Gamma \mid \Delta \vdash \text{tt}(\text{or ff}) : \mathbb{B}} \quad \mu_n(\pi) = \mathbb{1}_1 \\ \pi \triangleleft \frac{\sigma \triangleleft \Gamma \mid \Delta \vdash M : T \quad \tau \triangleleft \Gamma' \mid \Delta \vdash M' : T}{\Gamma, \Gamma' \mid \Delta \vdash \text{if}(M, M') : \mathbb{B} \multimap T} \quad \mu_n(\pi) = \mu_n(\sigma) + \mu_n(\tau) + \mathbb{1}_0 \end{array}$$

6.6 Examples in sEAL

We give some examples of terms in sEAL, first some terms we can usually see for the elementary affine logic, and then we give the term for computing tower of exponentials.

Some general results and notations on sEAL.

- For base types A we have the coercion $A \multimap !A$. For example, for words, this is given by the term $coerc_w = \mathit{iter}_W^!(!(\lambda w'.s_0(w')),!(\lambda w'.s_1(w')),!\epsilon)$, with $coerc_w \underline{w} \rightarrow^* !\underline{w}$
- We write $\lambda x \otimes y.M$ for the term $\lambda c.\mathit{let} \ x \otimes y = c \ \mathit{in} \ M$.

Polynomials and Tower of Exponentials in sEAL Recall that we defined polynomials in sℓT. With this we can define polynomials in EAL with type $\mathbf{N} \multimap \mathbf{N}$ using the sℓT call. Moreover, using the iteration in EAL, we can define a tower of exponential.

We can compute the function $k \mapsto 2^{2^k}$ in sEAL with type $\mathbf{N} \multimap !\mathbf{N}$

$$\frac{\frac{\frac{n : \mathbf{N} \mid \cdot \vdash n : \mathbf{N} \quad x_1 : \mathbf{N}^{a_1} \vdash_{\text{sℓT}} \mathit{mult} \ x_1 \ x_1 : \mathbf{N}^{a_1 \cdot a_1}}{n : \mathbf{N} \mid \cdot \vdash [\lambda x_1.\mathit{mult} \ x_1 \ x_1](n) : \mathbf{N}}}{\cdot \mid \vdash \lambda n.[\lambda x_1.\mathit{mult} \ x_1 \ x_1](n) : \mathbf{N} \multimap \mathbf{N}}}{\cdot \mid \vdash !(\lambda n.[\lambda x_1.\mathit{mult} \ x_1 \ x_1](n)) : !(\mathbf{N} \multimap \mathbf{N})} \quad \cdot \mid \vdash !\underline{2} : !\mathbf{N}}{\cdot \mid \vdash \mathit{exp} = \mathit{iter}_N^!(!\lambda n.[\lambda x_1.\mathit{mult} \ x_1 \ x_1](n), !\underline{2}) : \mathbf{N} \multimap !\mathbf{N}}$$

With $\mathit{iter}_N^!(!\lambda n.[\lambda x_1.\mathit{mult} \ x_1 \ x_1](n), !\underline{2}) \underline{k} \rightarrow^* !((\lambda n.[\lambda x_1.\mathit{mult} \ x_1 \ x_1](n))^k \underline{2}) \rightarrow^* !(2^{2^k})$.

For an example of measure, for the subproof

$\pi \triangleleft \cdot \mid \vdash \lambda n.[\lambda x_1.\mathit{mult} \ x_1 \ x_1](n) : \mathbf{N} \multimap \mathbf{N}$, we have $\mathit{depth}(\pi) = 1$ and as the weight for $\sigma \triangleleft x_1 : \mathbf{N}^{a_1} \vdash_{\text{sℓT}} \mathit{mult} \ x_1 \ x_1 : \mathbf{N}^{a_1 \cdot a_1}$ is $\omega(\sigma) = 4 + a_1 + 3a_1^3$, we can deduce $\mu(\pi) = (1 + 1 + 1 \cdot (d(\omega(\sigma)) + a_1 \cdot a_1) + 1), 1 + (\omega(\sigma) + a_1 \cdot a_1)[1/a_1] = (6, 10)$

If we define, $2_0^x = x$ and $2_{k+1}^x = 2^{2_k^x}$, with the use of polynomials, we can represent the function $n \mapsto 2_{2_k}^{P(n)}$ for all $k \geq 0$ and polynomial P with a term of type $\mathbf{N} \multimap !^k \mathbf{N}$.

Some other big examples, such as QBF_k and the $SUBSET_SUM$ problem can be found in the technical report [7]

6.7 Simulation of a Turing Machine in sEAL

The first thing we prove is the existence of a term in sℓT to simulate n steps of a deterministic Turing-machine on a word w . We give here the intuition of the encoding, and a more detailed explanation on how to work with this encoding can be found in the technical report [7].

Suppose given two variables $w : W^{a_w}$ and $n : \mathbf{N}^{a_n}$, we note $Conf_b$ the type $W^{a_w+b} \otimes \mathbf{B} \otimes W^{a_w+b} \otimes \mathbf{B}^q$, with q an integer and \mathbf{B}^q being q tensors of booleans. This type represents a configuration on a Turing machine after b steps, with \mathbf{B}^q coding the state, and then $\underline{w_0} \otimes b \otimes \underline{w_1}$ represents the tape, with b being the position of the head, w_0 represents the reverse of the word before b , and w_1 represents the word after b . We can then define multiple term in sℓT with this encoding. First we have a term init such that $w : W^{a_w}, n : \mathbf{N}^{a_n} \vdash \mathit{init} : Conf_1$ and init computes the initial configuration of the Turing machine. Then, we have a term step with $\cdot \vdash \mathit{step} : Conf_b \multimap Conf_{b+1}$ that computes the result of the transition function from a configuration to the next one, and finally we have a term final with $\cdot \vdash \mathit{final} : Conf_b \multimap \mathbf{B}$ verifying if the final configuration is accepted or not. Now that we have that, if we can compute an integer n bounding the number of steps of a Turing-machine on an entry w , then we can effectively simulate the Turing-machine in our calculus using a sℓT call. The height of the tower of exponential we can compute in this calculus is closely linked to the difference of ! modalities between the input and the output. You can see this with the examples in the appendix 6.6. This shows that, by using a ! modality, we can increase the integer n we can compute and thus increase the working time of the Turing-machine we want to simulate.

Beyond Admissibility: Dominance Between Chains of Strategies


Nicolas Basset

Université Grenoble Alpes
Grenoble, France
bassetni@univ-grenoble-alpes.fr

Ismaël Jecker

Department of Computer Science, Université Libre de Bruxelles
Brussels, Belgium
ismael.jecker@ulb.ac.be

Arno Pauly

Department of Computer Science, Swansea University
Swansea, UK
arno.m.pauly@gmail.com
 <https://orcid.org/0000-0002-0173-3295>

Jean-François Raskin

Department of Computer Science, Université Libre de Bruxelles
Brussels, Belgium
jraskin@ulb.ac.be

Marie Van den Bogaard

Department of Computer Science, Université Libre de Bruxelles
Brussels, Belgium
marie.van.den.bogaard@ulb.ac.be

Abstract

Admissible strategies, i.e. those that are not dominated by any other strategy, are a typical rationality notion in game theory. In many classes of games this is justified by results showing that any strategy is admissible or dominated by an admissible strategy. However, in games played on finite graphs with quantitative objectives (as used for reactive synthesis), this is not the case.

We consider increasing chains of strategies instead to recover a satisfactory rationality notion based on dominance in such games. We start with some order-theoretic considerations establishing sufficient criteria for this to work. We then turn our attention to generalised safety/reachability games as a particular application. We propose the notion of maximal uniform chain as the desired dominance-based rationality concept in these games. Decidability of some fundamental questions about uniform chains is established.

2012 ACM Subject Classification Theory of computation → Solution concepts in game theory, Theory of computation → Automata extensions

Keywords and phrases dominated strategies, admissible strategies, games played on finite graphs, reactive synthesis, reachability games, safety games, cofinal, order theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.10

Related Version A full version of the paper is available at [3], <https://arxiv.org/abs/1805.11608>.



© Nicolas Basset, Ismaël Jecker, Arno Pauly, Jean-François Raskin, and Marie Van den Bogaard; licensed under Creative Commons License CC-BY

27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 10; pp. 10:1–10:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This work was partially supported by the ERC inVEST (279499), the ARC project “Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond” (Fédération Wallonie-Bruxelles), the EOS project “Verifying Learning Artificial Intelligence Systems” (FNRS-FWO), and the Belgian FNRS PDR project “Subgame perfection in graph games” (T.0088.18). J.-F. Raskin is Professeur Francqui de Recherche funded by the Francqui foundation.

Acknowledgements We thank Gilles Geeraerts and Guillermo A. Pérez for several fruitful discussions.

1 Introduction

The canonical model to formalize the reactive synthesis problem are two-player win/lose perfect information games played on finite (directed) graphs [22, 1]. In recent years, more general objectives and multiplayer games have been studied (see e.g. [17] or [7] and additional references therein). When moving beyond two-player win/lose games, the traditional solution concept of a *winning strategy* needs to be updated by another notion. The game-theoretic literature offers a variety of concepts of *rationality* to be considered as candidates.

The notion we focus on here is *admissibility*: roughly speaking, judging strategies according to this criterion allows to deem rational only strategies that are not *worse* than any other strategy (ie, that are not *dominated*). In this sense, admissible strategies represent maximal elements in the whole set of strategies available to a player. One attractive feature of admissibility, or more generally, dominance based rationality notions is that they work on the level of an individual agent. Unlike e.g. to justify Nash equilibria, no common rationality, shared knowledge or any other assumptions on the other players are needed to explain why a specific agent would avoid dominated strategies.

The study of admissibility in the context of games played on graphs was initiated by Berwanger in [4] and subsequently became an active research topic (e.g. [12, 9, 2, 8, 10], see related work below). In [4], Berwanger established in the context of perfect-information games with boolean objectives that admissibility is the *good* criterion for rationality: every strategy is either admissible or dominated by an admissible strategy.

Unfortunately, this fundamental property does not hold when one considers quantitative objectives. Indeed, as soon as there are three different possible payoffs, one can find instances of games where a strategy is neither dominated by an admissible strategy, nor admissible itself (see Example 1). This third payoff actually allows for the existence of infinite domination sequences of strategies, where each element of the sequence dominates its predecessor and is dominated by its successor in the chain. Consequently, no strategy in such a chain is admissible. However, it can be the case that no admissible strategy dominates the elements of the chain. In the absence of a *maximal element* above these strategies, one may ask why they should be discarded in the quest of a rational choice. They may indeed represent a type of behaviour that is rational but not captured by the admissibility criterion.

Our contributions. To formalize this behaviour, we study increasing chains of strategies (Definition 3). A chain is weakly dominated by some other chain, if every strategy in the first is below some strategy in the second. The question then arises whether every chain is below a maximal chain. Based on purely order-theoretic argument, a sufficient criterion is given in Theorem 11. However, Corollary 17 shows that our sufficient criterion does not apply to all games of interests. We can avoid the issue by restricting to some countable class of strategies, e.g. just the regular, computable or hyperarithmetical ones (Corollary 19).

We test the abstract notion in the concrete setting of generalised safety/reachability games (Definition 21). Based on the observation that the crucial behaviour captured by chains of strategies, but not by single strategies is *Repeat this action a large but finite number of times*, we introduce the notion of a parameterized automaton (Definition 28), which essentially has just this ability over the standard finite automata. We then show that any finite memory strategy is below a maximal chain or strategy realized by a parameterized automaton (Theorem 31).

Finally, we consider some algorithmic properties of chains and parameterized automata in generalised safety/reachability games. It is decidable in PTIME whether a parameterized automaton realizes a chain of strategies (Theorem 35). It is also decidable in PTIME whether the chain realized by one parameterized automaton dominates the chain realized by another (Theorem 36).

Most proofs are omitted in the paper due to space restrictions. The appendix contains a selection of those. For the full account, we refer to the arXiv version [3].

Related work. As mentioned above, the study of dominance and admissibility for games played on graphs was initiated by Berwanger in [4]. Faella analyzed several criteria for how a player should play a win/lose game on a finite graph that she cannot win, eventually settling on the notion of admissible strategy [15].

Admissibility in quantitative perfect-information sequential games played on graphs was studied in [9]. Concurrent games were considered in [2]. In [8], games with imperfect information, but boolean objectives were explored. The study of decision problems related to admissibility (as we do in Subsection 4.3) was advanced in [12]. The complexity of decision problems related to dominance in normal form games has received attention, see [21] for an overview. For the role of admissibility for synthesis, we refer to [10].

Our Subsection 3.1 involves an investigation of cofinal chains in certain partially ordered sets. A similar theme (but with a different focus) is present in [25].

2 Background

2.1 Games on finite graphs

A *turn-based multiplayer game* \mathcal{G} on a finite graph G is a tuple $\mathcal{G} = \langle P, G, (p_i)_{i \in P} \rangle$ where:

- P is the non-empty finite set of players of the game,
- $G = \langle V, E \rangle$ where the finite set V of vertices of G is equipped with a $|P|$ -partition $\uplus_{i \in P} V_i$, and $E \subseteq V \times V$ is the edge relation of G ,
- for each player i in P , p_i is a *payoff function* that associates to every infinite path in G a payoff in \mathbb{R} .

Outcomes and histories. An *outcome* ρ of \mathcal{G} is an infinite path in G , that is, an infinite sequence of vertices $\rho = (\rho_k)_{k \in \mathbb{N}} \in V^\omega$, where for all $k \in \mathbb{N}$, $(\rho_k, \rho_{k+1}) \in E$. The set of all possible outcomes in \mathcal{G} is denoted $\mathbf{Out}(\mathcal{G})$. A finite prefix of an outcome is called a *history*. The set of all histories in \mathcal{G} is denoted $\mathbf{Hist}(\mathcal{G})$. For an outcome $\rho = (\rho_k)_{k \in \mathbb{N}}$ and an integer ℓ , we denote by $\rho_{\leq \ell}$ the history $(\rho_k)_{0 \leq k \leq \ell}$. The *length* of the history $\rho_{\leq \ell}$, denoted $|\rho_{\leq \ell}|$ is $\ell + 1$. Given an outcome or a history ρ and a history h , we write $h \subseteq_{\text{pref}} \rho$ if h is a prefix of ρ , and we denote by $h^{-1} \cdot \rho$ the unique outcome (or history) such that $\rho = h.(h^{-1} \cdot \rho)$. Given an outcome ρ or a history h and $k \in \mathbb{N}$ (respectively $k < |h|$), we denote by ρ_k (respectively h_k) the $k + 1$ -th vertex of ρ (respectively of h). For a history h , we define the last vertex of h to be $\text{last}(h) := h_{|h|-1}$ and its first vertex $\text{first}(h) := h_0$. For a vertex $v \in V$, its set of *successors* is $E_v = \{v' \in V \mid (v, v') \in E\}$.

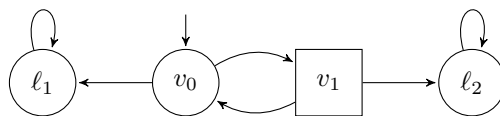
Strategy profiles and payoffs. A *strategy* of a player i is a function σ_i that associates to each history h such that $\text{last}(h) \in V_i$, a successor state $v \in E_{\text{last}(h)}$. A tuple of strategies $(\sigma_i)_{i \in P'}$ where $P' \subseteq P$, one for each player in P' is called a *profile* of strategies. Usually, we focus on a particular player i , thus, given a profile $(\sigma_i)_{i \in P}$, we write σ_{-i} to designate the collection of strategies of players in $P \setminus \{i\}$, and the complete profile is written (σ_i, σ_{-i}) . The set of all strategies of player i is denoted $\Sigma_i(\mathcal{G})$, while $\Sigma(\mathcal{G}) = \prod_{i \in P} \Sigma_i(\mathcal{G})$ is the set of all profiles of strategies in the game \mathcal{G} and $\Sigma_{-i}(\mathcal{G})$ is the set of all profiles of all players except Player i . As we consider games with perfect information and deterministic transitions, any complete profile $\sigma_P = (\sigma_i)_{i \in P}$ yields, from any history h , a unique outcome, denoted $\mathbf{Out}_h(\mathcal{G}, \sigma_P)$. Formally, $\mathbf{Out}_h(\mathcal{G}, \sigma_P)$ is the outcome ρ such that $\rho_{\leq |h|-1} = h$ and for all $k \geq |h| - 1$, for all $i \in P$, it holds that $\rho_{k+1} = \sigma_i(\rho_{\leq k})$ if $\rho_k \in V_i$. The set of outcomes (resp. histories) *compatible* with a strategy σ of player i after a history h is $\mathbf{Out}_h(\mathcal{G}, \sigma_i) = \{\rho \in \mathbf{Out}(\mathcal{G}) \mid \exists \sigma_{-i} \in \Sigma_{-i}(\mathcal{G}) \text{ such that } \rho = \mathbf{Out}_h(\mathcal{G}, (\sigma_i, \sigma_{-i}))\}$ (resp. $\mathbf{Hist}_h(\sigma) = \{h \in \mathbf{Hist}(\mathcal{G}) \mid \exists \rho \in \mathbf{Out}_h(\mathcal{G}, \sigma_i), n \in \mathbb{N} \text{ such that } h = \rho_{\leq n}\}$). Each outcome ρ yields a *payoff* $p_i(\rho)$ for each Player i . We denote with $p_i(h, \sigma, \tau)$ the payoff of a profile of strategies (σ, τ) after a history h .

Usually, we consider games instances such that players start to play at a fixed vertex. Thus, we call an *initialized game* a pair (\mathcal{G}, v_0) of a game \mathcal{G} and a vertex $v_0 \in V$. When the initial vertex v_0 is clear from context, we speak directly from \mathcal{G} , $\mathbf{Out}(\mathcal{G}, \sigma_P)$ and $p_i(\sigma_P)$ instead of (\mathcal{G}, v_0) , $\mathbf{Out}_{v_0}(\mathcal{G}, \sigma_P)$ and $p_i(v_0, \sigma_P)$.

Dominance relation. In order to compare different strategies of a player i in terms of payoffs, we rely on the notion of dominance between strategies: A strategy $\sigma \in \Sigma_i$ is *weakly dominated* by a strategy $\sigma' \in \Sigma_i$ at a history h compatible with σ and σ' , denoted $\sigma \preceq_h \sigma'$, if for every $\tau \in \Sigma_{-i}$, we have $p_i(h, \sigma, \tau) \leq p_i(h, \sigma', \tau)$. We say that σ is weakly dominated by σ' , denoted $\sigma \preceq \sigma'$ if $\sigma \preceq_{v_0} \sigma'$, where v_0 is the initial state of \mathcal{G} . A strategy $\sigma \in \Sigma_i$ is *dominated* by a strategy $\sigma' \in \Sigma_i$, at a history h compatible with σ and σ' , denoted $\sigma \prec_h \sigma'$, if $\sigma \preceq_h \sigma'$ and there exists $\tau \in \Sigma_{-i}$, such that $p_i(h, \sigma, \tau) < p_i(h, \sigma', \tau)$. We say that σ is dominated by σ' , denoted $\sigma \prec \sigma'$ if $\sigma \prec_{v_0} \sigma'$, where v_0 is the initial state of \mathcal{G} . Strategies that are not dominated by any other strategies are called *admissible*: A strategy $\sigma \in \Sigma_i$ is *admissible* (respectively *from* h) if $\sigma \not\prec \sigma'$ (resp. $\sigma \not\prec_h \sigma'$) for every $\sigma' \in \Sigma_i$.

Antagonistic and Cooperative Values. To study the rationality of different behaviours in a game \mathcal{G} , it is useful to be able to know, for a player i , a fixed strategy $\sigma \in \Sigma_i$ and any history h , the worst possible payoff Player i can obtain with σ from h (i.e., the payoff he will obtain assuming the other players play *antagonistically*), as well as the best possible payoff Player i can hope for with σ from h (i.e., the payoff he will obtain assuming the other players play *cooperatively*). The first value is called the *antagonistic value of the strategy* σ of Player i at history h in \mathcal{G} and the second value is called the *cooperative value of the strategy* σ of Player i at history h in \mathcal{G} . They are formally defined as $\text{aVal}_i(\mathcal{G}, h, \sigma) := \inf_{\tau \in \Sigma_{-i}} p_i(\mathbf{Out}_h(\sigma, \tau))$ and $\text{cVal}_i(\mathcal{G}, h, \sigma) := \sup_{\tau \in \Sigma_{-i}} p_i(\mathbf{Out}_h(\sigma, \tau))$.

Prior to any choice of strategy of Player i , we can define, for any history h , the *antagonistic value of h* for Player i as $\text{aVal}_i(\mathcal{G}, h) := \sup_{\sigma \in \Sigma_i} \text{aVal}_i(\mathcal{G}, h, \sigma)$ and the *cooperative value of h* for Player i as $\text{cVal}_i(\mathcal{G}, h) := \sup_{\sigma \in \Sigma_i} \text{cVal}_i(\mathcal{G}, h, \sigma)$. Furthermore, one can ask, from a history h , what is the maximal payoff one can obtain *while* ensuring the antagonistic value of h . Thus, we define the *antagonistic-cooperative value of h* for Player i as $\text{acVal}_i(\mathcal{G}, h) := \sup\{\text{cVal}_i(\mathcal{G}, h, \sigma) \mid \sigma \in \Sigma_i \text{ and } \text{aVal}_i(\mathcal{G}, h, \sigma) \geq \text{aVal}_i(\mathcal{G}, h)\}$. From now on, we will omit to precise \mathcal{G} when it is clear from the context.



■ **Figure 1** The *Help-me?*-game.

An initialized game (\mathcal{G}, v_0) is *well-formed for Player i* if, for every history $h \in \mathbf{Hist}_{v_0}(\mathcal{G})$, there exists a strategy $\sigma \in \Sigma_i$ such that $\mathbf{aVal}_i(h, \sigma) = \mathbf{aVal}(h)$, and a strategy $\sigma' \in \Sigma_i$ such that $\mathbf{cVal}_i(h, \sigma') = \mathbf{cVal}(h)$. In other words, at every history h , Player i has a strategy that ensures the payoff $\mathbf{aVal}_i(h)$, and a strategy that allows the other players to cooperate to yield a payoff of $\mathbf{cVal}_i(h)$.

In the following, we will always focus on the point of view of one player i , thus we will sometimes refer to him as the *protagonist* and assume it is the first player, while the other players $-i$ can be seen as a coalition and abstracted to a single player, that we will call the *antagonist*. Furthermore, we will omit the subscript i to refer to the protagonist when we use the notations $\mathbf{aVal}_i, \mathbf{cVal}_i, \mathbf{acVal}_i, p_i$, etc..

► **Example 1.** Consider the game depicted in Figure 1. The protagonist owns the circle vertices. The payoffs are defined as follows for the protagonist :

$$p(\rho) = \begin{cases} 0 & \text{if } \rho = (v_0v_1)^\omega, \\ 1 & \text{if } \rho = (v_0v_1)^n v_0 \ell_1^\omega \text{ where } n \in \mathbb{N}, \\ 2 & \text{if } \rho = (v_0v_1)^n \ell_2^\omega \text{ where } n \in \mathbb{N}. \end{cases}$$

Let us first look at the possible behaviours of the protagonist in this game, when he makes no assumption on the payoff function of the antagonist. He can choose to be “optimistic” and opt to try (at least for some time, or forever) to go to v_1 in the hope that the antagonist will cooperate to bring him to ℓ_2 , or settle from the start and go directly to ℓ_1 , not counting on any help from the antagonist. We denote by s_k the strategy that prescribes to choose v_1 as the successor vertex at the first k visits of v_0 , and ℓ_1 at the $k + 1$ -th visit, while s_ω denotes the strategy that prescribes v_1 at every visit of v_0 .

Fix $k \in \mathbb{N}$. Then, $s_k \prec s_{k+1}$: Indeed, for all $\tau \in \Sigma_{-i}$, if $p(s_k, \tau) = 2$, then there exists $j \leq k$ such that $\tau((v_0v_1)^j) = \ell_2$. As s_k and s_{k+1} agree up to $(v_0v_1)^k q_0$, we have that $\mathbf{Out}(s_{k+1}, \tau) = (v_0v_1)^j \ell_2^\omega = \mathbf{Out}(s_k, \tau)$, thus $p(s_{k+1}, \tau) = 2$ as well. Furthermore, consider a strategy τ such that $\tau((v_0v_1)^j) = v_0$ for all $j \leq k$ and $\tau((v_0v_1)^{k+1}) = \ell_2$. Then $p(s_k, \tau) = 1$ while $p(s_{k+1}, \tau) = 2$. Finally, consider the strategy τ such that $\tau((v_0v_1)^k) = v_0$ for all $k \in \mathbb{N}$. Then $p(s_k, \tau) = 1 = p(s_{k+1}, \tau)$. Hence, $s_k \prec s_{k+1}$. In addition, we observe that s_ω is admissible: for any strategy s_k , the strategy τ of the antagonist that moves to ℓ_2 at the $k + 1$ -th visit of v_1 yields a payoff of 1 against strategy s_k but 2 against strategy s_ω . Thus, $s_\omega \not\prec s_k$ for any $k \in \mathbb{N}$.

Quantitative vs Boolean setting. Remark that in the boolean variant of the *Help-me?* game considered in Example 1, where the payoff associated with the vertex ℓ_1 is 0 and the payoff associated with the vertex ℓ_2 is 1, every strategy s_k for $k \in \mathbb{N}$ is in fact dominated by s_ω , as s_k and s_ω both yield payoff 0 against τ such that $\tau((v_0v_1)^k) = v_0$ for all $k \in \mathbb{N}$. In fact, Berwanger in [4], showed that boolean games with ω -regular objectives enjoy the following fundamental property: every strategy is either admissible, or dominated by an admissible strategy. The existence of an admissible strategy in any such game follows as an immediate corollary.

Let us now illustrate how admissibility fails to capture fully the notion of rational behaviour in the quantitative case. Firstly, recall that the existence of admissible strategies is not guaranteed in this setting (see for instance the examples given in [9]). In [9], the authors identified a class of games for which the existence of admissible strategies (for Player i) is guaranteed: well-formed games (for Player i). However, even in such games, the desirable fundamental property that holds for boolean games is not assured to hold anymore. In fact, this is already true for quantitative well-formed games with only three different payoffs and really simple payoff functions. Indeed, consider again the *Help-me?* game in Figure 1. Remark that it is a well-formed game for the protagonist. We already showed that any strategy s_k is dominated by the strategy s_{k+1} . Thus, none of them is admissible. The only admissible strategy is s_ω . It is easy to see that $s_k \not\preceq s_\omega$ for any $k \in \mathbb{N}$: Let $\tau \in \Sigma_{-i}$ be such that $\tau((v_0 v_1)^k) = v_0$ for all $k \in \mathbb{N}$. Then $p(s_k, \tau) = 1 > 0 = p(s_\omega, \tau)$. To sum up, we see that there exists an infinite sequence $(s_k)_{k \in \mathbb{N}}$ of strategies such that none of its elements is dominated by the only admissible strategy s_ω . However, the sequence $(s_k)_{k \in \mathbb{N}}$ is totally ordered by the dominance relation. Based on these observations, we take the approach to not only consider single strategies, but also such ordered sequences of strategies, that can represent a type of rational behaviour not captured by the admissibility concept.

2.2 Order theory

In this paragraph we recall the standard results from order theory that we need (see e.g. [19]).

A *linear order* is a total, transitive and antisymmetric relation. A linearly ordered set $(R, <)$ is a *well-order*, if every subset of R has a minimal element w.r.t. $<$. The ordinals are the canonical examples of well-orders, in as far as any well-order is order-isomorphic to an ordinal. The ordinals themselves are well-ordered by the relation $<$ where $\alpha \leq \beta$ iff α order-embeds into β . The first infinite ordinal is denoted by ω , and the first uncountable ordinal by ω_1 .

A *partial order* is a transitive and reflexive relation. Let (X, \preceq) be a partially ordered set (*poset* for short). A *chain* in (X, \preceq) is a subset of X that is totally ordered by \preceq . An *increasing chain* is an ordinal-indexed family $(x_\beta)_{\beta < \alpha}$ of elements of X such that $\beta < \gamma < \alpha \Rightarrow x_\beta \prec x_\gamma$. If we only have that $\beta < \gamma$ implies $x_\beta \preceq x_\gamma$, we speak of a *weakly increasing chain*. We are mostly interested in (weakly) increasing chains in this paper, and will thus occasionally suppress the words *weakly increasing* and only speak about *chains*.

A subset Y of a partially ordered set (X, \preceq) is called *cofinal*, if for every $x \in X$ there is a $y \in Y$ with $x \preceq y$. A consequence of the axiom of choice is that every chain contains a cofinal increasing chain, which is one reason for our focus on increasing chains. It is obvious that having multiple maximal elements prevents the existence of a cofinal chain, but even a lattice can fail to admit a cofinal chain. An example we will go back to is $\omega_1 \times \omega$ (cf. [19]).

If (X, \preceq) admits a cofinal chain, then its *cofinality* (denoted by $\text{cof}(X, \preceq)$) is the least ordinal α indexing a cofinal increasing chain in (X, \preceq) . The possible values of the cofinality are 1 or infinite regular cardinals (it is common to identify a cardinal and the least ordinal of that cardinality). In particular, a countable chain can only have cofinality 1 or ω . The first uncountable cardinal \aleph_1 is regular, and $\text{cof}(\omega_1) = \omega_1$.

We will need the probably most-famous result from order theory:

► **Lemma 2 (Zorn's Lemma).** *If every chain in (X, \preceq) has an upper bound, then every element of X is below a maximal element.*

3 Increasing chains of strategies

3.1 Ordering chains

In this subsection, we study the poset of increasing chains in a given poset (X, \preceq) . We denote by $\text{IC}(X, \preceq)$ the set of increasing chains in (X, \preceq) . Our intended application will be that (X, \preceq) is the set of strategies for the protagonist in a game ordered by the dominance relation. However, in this subsection we are not exploiting any properties specific to the game-setting. Instead, our approach is purely order-theoretic.

► **Definition 3.** We introduce an order \sqsubseteq on $\text{IC}(X, \preceq)$ by defining:

$$(x_\beta)_{\beta < \alpha} \sqsubseteq (y_\gamma)_{\gamma < \delta} \quad \text{if} \quad \forall \beta < \alpha \exists \gamma < \delta \quad x_\beta \preceq y_\gamma$$

Note that \sqsubseteq is a partial order. Let \doteq denote the corresponding equivalence relation. We will occasionally write short IC for $(\text{IC}(X, \preceq), \sqsubseteq)$.

Inspired by our application to dominance between strategies in games, we will refer to both \preceq and \sqsubseteq as the *dominance* relation, and might express e.g. $(x_\beta)_{\beta < \alpha} \sqsubseteq (y_\gamma)_{\gamma < \delta}$ as $(x_\beta)_{\beta < \alpha}$ is dominated by $(y_\gamma)_{\gamma < \delta}$, or $(y_\gamma)_{\gamma < \delta}$ dominates $(x_\beta)_{\beta < \alpha}$. There is no risk to confuse whether \preceq or \sqsubseteq is meant, since $x \preceq y$ iff $(x)_{\beta < 1} \sqsubseteq (y)_{\gamma < 1}$. Continuing the identification of $x \in X$ and $(x)_{\beta < 1} \in \text{IC}$, we will later also speak about a single strategy dominating a chain or vice versa.

The central notion we are interested in will be that of a maximal chain:

► **Definition 4.** $A \in \text{IC}$ is called *maximal*, if $A \sqsubseteq B$ for $B \in \text{IC}$ implies $B \sqsubseteq A$.

We desire situations where every chain in IC is either maximal or below a maximal chain. Noting that this goal is precisely the conclusion of Zorn's Lemma (Lemma 2), we are led to study chains of chains; for if every chain of chains is bounded, Zorn's Lemma applies. Since (IC, \sqsubseteq) is a poset just as (X, \preceq) is, notions such as cofinality apply to chains of chains just as they apply to chains. We will gather a number of lemmas we need to clarify when chains of chains are bounded.

In a slight abuse of notation, we write $(x_\beta)_{\beta < \alpha} \subseteq (y_\gamma)_{\gamma < \delta}$ iff $\{x_\beta \mid \beta < \alpha\} \subseteq \{y_\gamma \mid \gamma < \delta\}$. Clearly, $(x_\beta)_{\beta < \alpha} \subseteq (y_\gamma)_{\gamma < \delta}$ implies $(x_\beta)_{\beta < \alpha} \sqsubseteq (y_\gamma)_{\gamma < \delta}$. We can now express cofinality by noting that $(x_\beta)_{\beta < \alpha}$ is cofinal in $(y_\gamma)_{\gamma < \delta}$ iff $(x_\beta)_{\beta < \alpha} \subseteq (y_\gamma)_{\gamma < \delta}$ and $(y_\gamma)_{\gamma < \delta} \sqsubseteq (x_\beta)_{\beta < \alpha}$. We recall that the cofinality of $(y_\gamma)_{\gamma < \delta}$ (denoted by $\text{cof}((y_\gamma)_{\gamma < \delta})$) is the least ordinal α such that there exists some $(x_\beta)_{\beta < \alpha}$ which is cofinal in $(y_\gamma)_{\gamma < \delta}$.

► **Lemma 5.** If $(x_\beta)_{\beta < \alpha} \doteq (y_\gamma)_{\gamma < \delta}$, then there is some $(y'_\lambda)_{\lambda < \alpha'} \subseteq (y_\gamma)_{\gamma < \delta}$ with $\alpha' \leq \alpha$ and $(y'_\lambda)_{\lambda < \alpha'} \doteq (y_\gamma)_{\gamma < \delta}$.

► **Corollary 6.** $\text{cof}((y_\gamma)_{\gamma < \delta})$ is equal to the least ordinal α such that there exists $(x_\beta)_{\beta < \alpha}$ with $(x_\beta)_{\beta < \alpha} \doteq (y_\gamma)_{\gamma < \delta}$.

► **Corollary 7.** For every chain $(y_\gamma)_{\gamma < \delta}$ there exists an equivalent chain $(x_\beta)_{\beta < \alpha}$ such that $\alpha = 1$ or α is an infinite regular cardinal. In particular, if δ is countable, then $(y_\gamma)_{\gamma < \delta}$ is equivalent to a singleton or some chain $(x_n)_{n < \omega}$.

We briefly illustrate the concepts introduced so far in the game setting. Notice that for a game \mathcal{G} and a Player i , the pair $(\Sigma_i(\mathcal{G}), \preceq)$ is indeed a partially ordered set. We can thus consider the set $\text{IC}(\Sigma_i(\mathcal{G}), \preceq)$ of increasing chains of strategies in \mathcal{G} .

► **Example 8.** Recall the *Help-me?* game of Figure 1 and consider the set (Σ_i, \preceq) of strategies of the protagonist partially ordered by the weak dominance relation. Any single strategy is an increasing chain, indexed by the ordinal 1. We already noted that the strategy s_ω is admissible, thus the chain consisting of s_ω is maximal with respect to \sqsubseteq . Furthermore, the sequence of strategies $(s_k)_{k < \omega}$ is an increasing chain. Indeed, we know that for any $k < \omega$, we have $s_k \prec s_{k+1}$. It is a maximal one: in fact, since the set of strategies of the protagonist solely consists of the strategies of this chain and s_ω , and as $s_k \not\preceq s_\omega$ for any $k < \omega$, we get that any chain $(\sigma_\beta)_{\beta < \alpha}$ such that $(s_k)_{k < \omega} \sqsubseteq (\sigma_\beta)_{\beta < \alpha}$ satisfies $(\sigma_\beta)_{\beta < \alpha} \subseteq (s_k)_{k < \omega}$. Thus, $(\sigma_\beta)_{\beta < \alpha} \sqsubseteq (s_k)_{k < \omega}$. Let $(\sigma_\beta)_{\beta < \alpha}$ be an increasing chain indexed by the ordinal α . First, remark that $\alpha \leq \omega$. If $\alpha < \omega$, then the cofinality of $(\sigma_\beta)_{\beta < \alpha}$ is 1 as $(\sigma_\beta)_{\beta < \alpha}$ is equivalent to the strategy $\sigma_{\alpha-1}$: every strategy of $(\sigma_\beta)_{\beta < \alpha}$ is weakly dominated by $\sigma_{\alpha-1}$, and as the strategy $\sigma_{\alpha-1}$ is included in the increasing chain $(\sigma_\beta)_{\beta < \alpha}$, it is weakly dominated by $(\sigma_\beta)_{\beta < \alpha}$. If $\alpha = \omega$, then the cofinality of $(\sigma_\beta)_{\beta < \alpha}$ is ω : As for every finite chain $(\sigma'_{\beta'})_{\beta' < \alpha'}$ with $1 < \alpha' < \omega$, there exists $n < \omega$ such that $(\sigma'_{\beta'})_{\beta' < \alpha'} \sqsubseteq \sigma_n$, and thus $(\sigma_\beta)_{\beta < \alpha}$ is not (weakly) dominated by $(\sigma'_{\beta'})_{\beta' < \alpha'}$. Moreover, we have that $(\sigma_\beta)_{\beta < \alpha} \dot{=} (s_k)_{k < \omega}$ and is thus maximal. Indeed, since $(\sigma_\beta)_{\beta < \alpha}$ is a chain that is not a singleton, we already know that $(\sigma_\beta)_{\beta < \alpha} \subseteq (s_k)_{k < \omega}$, that is $(\sigma_\beta)_{\beta < \alpha} \sqsubseteq (s_k)_{k < \omega}$. Let now $k < \omega$. As $(\sigma_\beta)_{\beta < \alpha}$ is an increasing chain and $\alpha = \omega$, we have that there exists $n < \omega$ and $k' \geq k$ such that $\sigma_n = s_{k'}$. Thus, $s_k \preceq \sigma_n$ since $(s_k)_{k < \omega}$ is an increasing chain. Hence, we also have $(s_k)_{k < \omega} \sqsubseteq (\sigma_\beta)_{\beta < \alpha}$.

Now we are ready to prove the main technical result of this section 3.1, which identifies the potential obstructions for each chain in IC to have an upper bound:

► **Lemma 9.** *The following are equivalent:*

1. *If $((x_\beta^\gamma)_{\beta < \alpha_\gamma})_{\gamma < \delta}$ is an increasing chain in IC, then it has an upper bound in IC.*
2. *If $((x_\beta^\gamma)_{\beta < \alpha_\gamma})_{\gamma < \delta}$ is an increasing chain in IC with $\alpha \neq \delta$, $\text{cof}((x_\beta^\gamma)_{\beta < \alpha}) = \alpha > 1$ and $\text{cof}(((x_\beta^\gamma)_{\beta < \alpha})_{\gamma < \delta}) = \delta > 1$, then it has an upper bound in IC.*

Let us illustrate the problem of extending Lemma 9 by an example:

► **Example 10** ([19, Example 1]). Let $(X, \preceq) = \omega_1 \times \omega$, i.e. the product order of the first uncountable ordinal and the first infinite ordinal. Consider the chain of chains given by $x_n^\gamma = (\gamma, n)$, this corresponds to the case $\alpha = \omega$, $\delta = \omega_1$ in Lemma 9. If this chain of chains had an upper bound, then $\omega_1 \times \omega$ would need to admit a cofinal chain. However, this is not the case.

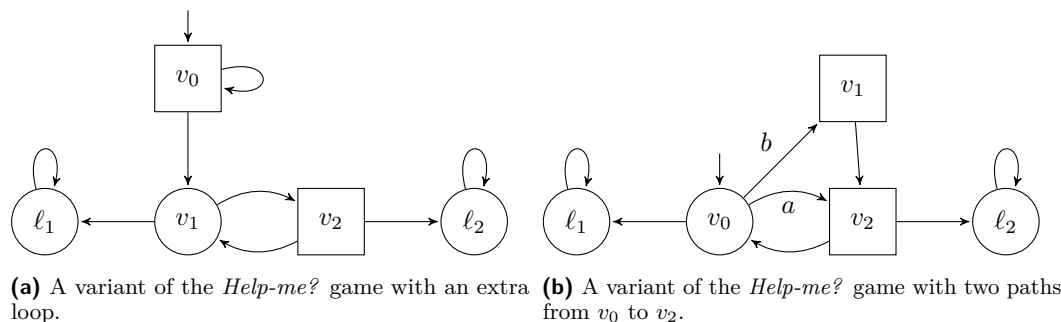
However, we can guarantee the existence of a maximal chain above any chain when there is no uncountable increasing chain of increasing chains.

► **Theorem 11.** *If all increasing chains of elements in IC (i.e., increasing chains of increasing chains of elements of (X, \preceq)) have a countable number of elements, then for every $A \in \text{IC}$ there exists a maximal $B \in \text{IC}$ with $A \sqsubseteq B$.*

Proof. We first argue that Condition 2 in Lemma 9 is vacuously true. As all increasing chains in IC are countable, the only possible value $\delta > 1$ for $\delta = \text{cof}(((x_\beta^\gamma)_{\beta < \alpha})_{\gamma < \delta})$ is $\delta = \omega$. As (X, \preceq) embeds into IC, if all chains in IC are countable, then so are all chains in (X, \preceq) . This tells us that the only possible value for α is $\alpha = \omega$. But then $\alpha \neq \delta$ cannot be satisfied.

By Lemma 9, Condition 1 follows. We can then apply Zorn's Lemma (Lemma 2) to conclude the claim. ◀

A small modification of the example shows that we cannot replace the requirement that IC has only countable increasing chains in Theorem 11 with the simpler requirement that (X, \preceq) has only countable increasing chains:



■ **Figure 2** Two variants of the *Help-me?* game.

► **Example 12.** Let $X = \omega_1 \times \omega$, and let $(\alpha, n) \prec (\beta, m)$ iff $\alpha \leq \beta$ and $n < m$. Then (X, \prec) has only countable increasing chains, but IC still has the chain of chains given by $x_n^\gamma = (\gamma, n)$ as in Example 10.

3.2 Uncountably long chains of chains

Unfortunately, we can design a game such that there exists an uncountable increasing chain of increasing chains. Thus the existence of a maximal element above any chain is not guaranteed by Theorem 11. In fact, we will see that the chain of chains of uncountable length we construct is not below any maximal chain.

► **Example 13.** We consider a variant of the *Help-me?* game (Example 1), depicted in Figure 2a. The strategies of the protagonist in this game can be described by functions $f : \mathbb{N} \rightarrow \mathbb{N} \cup \{\infty\}$ describing how often the protagonist is willing to repeat the second loop (between v_1 and v_2) given the number of repetitions the antagonist made in the first loop (at v_0). With the same reasoning as in Example 1 we find that the strategy corresponding to a function g dominates the strategy corresponding to f iff $\forall n \in \mathbb{N} f(n) = \infty \Leftrightarrow g(n) = \infty$ and $\forall n \in \mathbb{N} f(n) \leq g(n)$.

► **Definition 14.** Let $\mathbb{N}^{\mathbb{N}}$ denote the set of functions $f : \mathbb{N} \rightarrow \mathbb{N}$. For $f, g \in \mathbb{N}^{\mathbb{N}}$, let $f \leq g$ denote that $\forall n \in \mathbb{N} f(n) \leq g(n)$.

► **Observation 15.** There is an embedding of $(\mathbb{N}^{\mathbb{N}}, \leq)$ into the strategies of the game in Example 13 ordered by dominance such that no strategy in the range of embedding is dominated by a strategy outside the range of the embedding.

► **Proposition 16** ⁽¹⁾. For every chain $(f_n)_{n \in \mathbb{N}}$ in $(\mathbb{N}^{\mathbb{N}}, \leq)$ there exists a chain of chains $((f_n^\alpha)_{n < \omega})_{\alpha < \omega_1}$ of length ω_1 with $(f_n^0)_{n < \omega} \supseteq (f_n)_{n < \omega}$.

► **Corollary 17.** *The game in Example 13 has uncountably long chains of chains not below any maximal chains.*

Proof. Combine Observation 15 and Proposition 16. ◀

¹ This result is adapted from an answer by user *Deedlit* on math.stackexchange.org [16].

3.3 Chains over countable posets (X, \preceq)

Our proof of Proposition 16 crucially relied on functions of type $f : \mathbb{N} \rightarrow \mathbb{N}$ with arbitrarily high rate of growth. In concrete applications such functions would typically be unwelcome. In fact, for almost all classes of games of interest in (theoretical) computer science, a countable collection of strategies suffices for the players to attain their attainable goals. Restricting to computable strategies often makes sense. Many games played on finite graphs are even finite-memory determined (see [18] for how this extends to the quantitative case), and thus strategies implementable by finite automata are all that need to be considered.

Restricting consideration to a countable set of strategies indeed circumvents the obstacle presented by Proposition 16. The reason is that the cardinality of the length of a chain of chains cannot exceed that of the underlying partially ordered set (X, \preceq) :

► **Proposition 18.** For any increasing chain $((x_\beta^\gamma)_{\beta < \alpha})_{\gamma < \delta}$ in $\text{IC}(X, \preceq)$ we find that $|\delta| \leq |X|$.

Proof. Let $X_\gamma = \{x \in X \mid \exists \beta < \alpha \ x \preceq x_\beta^\gamma\}$. We find that $X_{\gamma_1} \subsetneq X_{\gamma_2}$ for any $\gamma_1 < \gamma_2 < \delta$ as a direct consequence of $(x_\beta^{\gamma_1})_{\beta < \alpha} \sqsubset (x_\beta^{\gamma_2})_{\beta < \alpha}$. Pick for each $\gamma < \delta$ some $y_\gamma \in X_{\gamma+1} \setminus X_\gamma$. Then $y : \delta \rightarrow X$ is an injection, establishing $|\delta| \leq |X|$. ◀

► **Corollary 19.** If (X, \preceq) is countable, then any increasing chain is maximal or below a maximal chain.

Proof. Proposition 18 shows that Theorem 11 applies. ◀

► **Example 20.** We return to the *Help-me?* game (Example 1, Figure 1). With the analysis done in Example 8, we have seen that any increasing chain C is either maximal or such that $C \sqsubseteq (\sigma_n)_{n < \omega}$, which is maximal. This fact can be derived directly from Corollary 19 as the number of strategies in \mathcal{G} is countable. Note also that the seemingly irrelevant loop we added in Figure 2a has a fundamental impact on the behaviour of chains of strategies!

4 Generalised safety/reachability games

► **Definition 21.** A *generalised safety/reachability game* (for Player i) $\mathcal{G} = \langle P, G, L, (p_i)_{i \in P} \rangle$ is a turn-based multiplayer game on a finite graph such that:

- $L \subseteq V$ is a finite set of *leaves*,
- for each $\ell \in L$, we have that $(\ell, v) \in E$ if, and only if $v = \ell$, that is, each leaf is equipped with a self-loop, and no other outgoing transition,
- for each $\ell \in L$, there exists an associated payoff $n_\ell \in \mathbb{Z}$ such that: for each outcome ρ , we have $p_i(\rho) = \begin{cases} n_\ell & \text{if } \rho \in V^* \ell^\omega, \\ 0 & \text{otherwise.} \end{cases}$

The traditional reachability games can be recovered as the special case where all leaves are associated with the same positive payoff, whereas the traditional safety games are those generalised safety/reachability games with a single negative payoff attached to leaves. This class was studied under the name *chess-like* games in [5, 6].

Generalised safety/reachability games are *well-formed* for Player i . Furthermore, they are prefix-independent, that is, for any outcome ρ and history h , we have that $p_i(h\rho) = p_i(\rho)$. Without loss of generality, we consider that there is either a unique leaf $\ell(n) \in L$ or no leaf for each possible payoff $n \in \mathbb{Z}$.

It follows from the transfer theorem in [18] (in fact, already from the weaker transfer theorem in [13]) that generalised safety/reachability games are finite memory determined. With a slight modification, we see that for any history h and strategy σ , there exists a

finite-memory strategy σ' such that $cVal(h, \sigma') = cVal(h, \sigma)$ and $aVal(h, \sigma') = aVal(h, \sigma)$. We shall thus restrict our attention to finite memory strategies, of which there are only countably many. We then obtain immediately from Corollary 19:

► **Corollary 22.** *In a generalised safety/reachability game, every increasing chain comprised of finite memory strategies is either maximal or dominated by a maximal such chain.*

If our goal is only to obtain a dominance-related notion of rationality, then for generalised safety/reachability games we can be satisfied with maximal chains comprised of finite memory strategies. However, for applications, it would be desirable to have a concrete understanding of these maximal chains. For this, having used Zorn's Lemma in the proof of their existence surely is a bad omen!

After collecting some useful lemmas on dominance in generalised safety/reachability games in Section 4.1, we will introduce the notion of *uniform chains* in Section 4.2. These are realized by automata of a certain kind, and thus sufficiently concrete to be amenable to algorithmic manipulations.

4.1 Dominance in generalised safety/reachability games

Given a generalised safety/reachability game \mathcal{G} and two strategies σ_1 and σ_2 of Player i , we can provide a criterion to show that σ_1 is not dominated by σ_2 :

► **Lemma 23.** *Let σ_1 and σ_2 be two strategies of Player i in a generalised safety/reachability game \mathcal{G} . Then, $\sigma_1 \not\preceq \sigma_2$ if, and only if, there exists a history h compatible with σ_1 and σ_2 such that $\text{last}(h) \in V_i$, $\sigma_1(h) \neq \sigma_2(h)$ and $cVal(h, \sigma_1) > aVal(h, \sigma_2)$.*

Intuitively, if there is no history where the two strategies disagree, they are in fact equivalent, and if, at every history where they disagree, the best payoff σ_1 can achieve (that is, $cVal(h, \sigma_1)$) is less than the one σ_2 can ensure (that is, $aVal(h, \sigma_2)$), then $\sigma_1 \preceq \sigma_2$. On the other hand, if they disagree at a history h and the best payoff σ_1 can achieve is strictly greater than the one σ_2 can ensure, then there exist a strategy of the antagonist that will yield exactly these payoffs against σ_1 and σ_2 respectively, which means that $\sigma_1 \not\preceq \sigma_2$. This result follows from the proof of Theorem 11 in [9]. The proof adapted to our setting can be found in the appendix.

We call such a history h a *non-dominance witness* of σ_1 by σ_2 . The existence of non-dominance witnesses allows us to conclude that in generalised safety/reachability games, all increasing chains are countable (not just those comprised of finite memory strategies).

► **Corollary 24.** *If $(\sigma_\beta)_{\beta < \alpha}$ is an increasing chain in generalised safety/reachability game, then α is countable.*

Proof. Assume that a history h is a witness of non-dominance of σ_2 by σ_1 , and of σ_3 by σ_2 , but not of σ_1 by σ_2 or σ_2 by σ_3 . Then $cVal(h, \sigma_2) > aVal(h, \sigma_1)$, $cVal(h, \sigma_3) > aVal(h, \sigma_2)$, $cVal(h, \sigma_1) \leq aVal(h, \sigma_2)$ and $cVal(h, \sigma_2) \leq aVal(h, \sigma_3)$. It follows that $aVal(h, \sigma_1) < aVal(h, \sigma_3)$ and $cVal(h, \sigma_1) < cVal(h, \sigma_3)$. Thus, if there are k different possible values, then any increasing chain of strategies using h as witness of non-dominance between them can have length at most $2k - 1$.

But if there were an uncountably long increasing chain, by the pigeon hole principle it would have an uncountably long subchain where all non-dominance witnesses in the reverse direction are given by the same history. ◀

10:12 Dominance Between Chains of Strategies

As we only handle countable chains, in the following we use the usual notation $(\sigma_n)_{n \in \mathbb{N}}$ to index chains.

The following lemma states that we can also extract witnesses for a strategy to be non-maximal (non-admissible or strictly dominated):

► **Lemma 25.** *Let \mathcal{G} be a generalised safety/reachability game and σ a strategy of Player i . The strategy σ is not admissible if, and only if there exists a history h compatible with σ such that $\text{aVal}(h, \sigma) \leq \text{cVal}(h, \sigma) \leq \text{aVal}(h) \leq \text{acVal}(h)$ where at least one inequality is strict.*

This result is a reformulation of Theorem 11 in [9] catered to our context and with a focus on the non-admissibility rather than on admissibility (see the arXiv version [3] for a proof adapted to our setting).

► **Definition 26.** Call a history h as in Lemma 25 a non-admissibility witness for σ . Call σ preadmissible, if for every non-admissibility witness hv of σ we find that $h = h'vh''$ with $\text{aVal}(h'v, \sigma) = \text{aVal}(h'v)$ and $\text{cVal}(h'v, \sigma) = \text{acVal}(h'v)$.

While a preadmissible strategy may fail to be admissible, it is not possible to improve upon it the first time it enters some vertex. Only when returning to a vertex later it may make suboptimal choices. Moreover, before a dominated choice is possible at a vertex, previously both the antagonistic and the antagonistic-cooperative value were realized at that vertex by the preadmissible strategy.

► **Lemma 27.** *In a generalised safety/reachability game, every strategy is either preadmissible or dominated by a preadmissible strategy.*

Proof sketch. Essentially, we can change how a strategy behaves locally on those histories that are an obstacle to it being preadmissible by replacing by a finite memory strategy that realizes the antagonistic and the antagonistic-cooperative value there. ◀

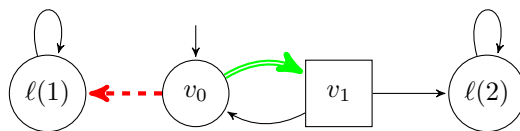
4.2 Parameterized automata and uniform chains

Let a *parameterized automaton* be a Mealy automaton that in addition can access a single counter in the following way: In a counter-access-state, a transition is chosen based on whether the counter value is 0 or not. Furthermore, in these counter-access-states, when the counter value is greater than 0, the counter is decremented by 1, otherwise, it stays at 0. In the remaining states, only one transition is possible and the counter value is not affected.

► **Definition 28.** A *parameterized automaton* for Player $i \in P$ over a game graph $G = (V, E)$ is a tuple $\mathcal{M} = (M, M_C, m_0, V, \mu, \nu)$ where:

- M is a non-empty finite set of memory states and $M_C \subseteq M$ is the set of *counter-access* states,
- m_0 is the initial memory state,
- V is the set of vertices of G ,
- $\mu : M \times V \times \mathbb{N} \rightarrow M \times \mathbb{N}$ is the memory and counter update function,
- $\nu : M \times V_i \times \mathbb{N} \rightarrow V$ is the move choice function for Player i , such that $(v, \nu(m, v, n)) \in E$ for all $m \in M$ and $v \in V_i$ and $n \in \mathbb{N}$.

The memory and counter-update function μ respects the following conditions: for each $m \in M \setminus M_C$, and $v \in V$, there exists $m' \in M$ such that $\mu(m, v, n) = (m', n)$ for all $n \in \mathbb{N}$. for each $m \in M_C$, and $v \in V$, there exists $m' \in M$ such that $\mu(m, v, n) = (m', n - 1)$ for all $n > 0$ and $m'' \in M$ such that $\mu(m, v, 0) = (m'', 0)$. The move choice function ν respects



■ **Figure 3** Product of the *Help-me?* game with parameterized automaton with a single memory state realizing $(s_k)_{k \in \mathbb{N}}$.

the following conditions: for each $m \in M \setminus M_C$, and $v \in V_i$, there exists $v' \in V$ such that $\nu(m, v, n) = v'$ for all $n \in \mathbb{N}$. For each $m \in M_C$, and $v \in V_i$, there exists $v', v'' \in V$ such that $\nu(m, v, n) = v'$ for all $n > 0$ and $\nu(m, v, 0) = v''$.

To ease presentation and understanding, we call transitions that decrement the counter *green* transitions, the transitions only taken when the counter value is 0 *red* transitions, and the ones that do not depend on the counter value *black* transitions. This classification between *green*, *red* and *black* transitions extends naturally to the edges of the product $\mathcal{M} \times G$ (that is, the graph with set of vertices $M \times V$ and edges induced by the functions μ and ν).

Parameterized automata can be seen as a collection of finite Mealy automata, one for each initialization of the counter. Thus, we say that a parameterized automaton \mathcal{M} realizes a *sequence* of finite-memory strategies $(\sigma_n)_{n \in \mathbb{N}}$. In the remainder of the paper, we focus on chains realized by parameterized automata:

► **Definition 29.** Let a chain $(\sigma_n)_{n \in \mathbb{N}}$ of strategies be called a *uniform* chain if there is a parameterized automaton \mathcal{M} that realizes σ_n if the counter is initialized with the value n . If $(\sigma_n)_{n \in \mathbb{N}}$ is maximal for \sqsubseteq amongst the increasing chains comprised of finite memory strategies, we call it a maximal uniform chain.

► **Example 30.** The *Help-me?* game from Figure 1 is clearly a generalised safety/reachability game with two leaves. The chain of strategies $(s_k)_{k \in \mathbb{N}}$ exposed in Example 1 is a uniform chain, as it is realized by the parameterized automaton that loops k times when its counter is initialized with value k . Figure 3 shows the product between this parameterized automaton and the game graph. The green (doubled) edge corresponds to the transition to take when the counter value is greater than 0 and should be decremented, while the red (dashed) edge corresponds to the transition to take when the counter value is 0.

The following theorem shows us that uniform chains indeed suffice to realize any rational behaviour in the sense of maximal chains:

► **Theorem 31.** *In a generalised safety/reachability game, every dominated finite memory strategy is dominated by an admissible finite memory strategy or by a maximal uniform chain.*

Theorem 31 cannot be extended to state that every chain comprised of finite memory strategies is below an admissible strategy or a maximal uniform chain. Note that there are only countably many uniform chains.

► **Example 32.** There is a generalised safety/reachability game where there are uncountably many incomparable maximal chains of finite memory strategies.

Proof. Consider the game depicted in Figure 2b. For any $p \in \{a, b\}^\omega$, define a chain of finite memory strategies by letting the n -strategy be *loop n times while playing the symbols from $p_{\leq n}$, then quit*. For each p , we obtain a different maximal chain. ◀

4.3 Algorithmic properties

In this section, we prove two decidability results concerning parametrized automata.

First, we prove that we can decide whether the sequence of strategies realized by a parameterized automaton is a chain. Note that this decision problem is not trivial: not every parameterized automaton realizes an (increasing) chain of strategies. For instance, if we switch the red and green transitions in the automaton/game graph product of figure 3, the sequence of strategies realized consists of s_ω when the counter is initialized with value 0, and s_0 when it is initialized with any other value. As $s_\omega \not\preceq s_0$, it is not a chain.

Second, we demonstrate that we can compare uniform chains: given two parametrized automata defining chains of strategies, we can decide whether one is dominated by the other. We begin by proving that strategies realized by Mealy automata are comparable.

► **Lemma 33.** *Let \mathcal{G} be a generalised safety/reachability game, let σ and σ' be finite-memory strategies realized by the finite Mealy automata \mathcal{M} and \mathcal{M}' . It is decidable in PTIME whether $\sigma \preceq \sigma'$.*

Proof sketch. We construct the game \mathcal{G}' of perfect information for two players, *Challenger* and *Prover*, such that Prover wins the game if and only if $\sigma \preceq \sigma'$. The goal of Challenger is to show that there exists a non-dominance witness of σ by σ' , that is, according to Lemma 23, a history h compatible with σ and σ' such that $\text{last}(h) \in V_i$, $\sigma(h) \neq \sigma'(h)$ and $\text{cVal}(h, \sigma) > \text{aVal}(h, \sigma')$. The game can be decomposed into the following phases:

- first, Challenger chooses a path \tilde{h} in $\mathcal{M} \times G \times \mathcal{M}'$ such that \tilde{h} has no successor in $\mathcal{M} \times G \times \mathcal{M}'$. This guarantees that h is compatible with σ and σ' , and that $\sigma(h) \neq \sigma'(h)$.
- Challenger then announces two values: c and a , such that $c > a$.
- Prover now can choose to contest either value c or value a .
- If Prover chooses to contest c , the game proceeds to a subgame \mathcal{C} , where Challenger has to find a continuation path in $(\mathcal{M} \times G)$ that yields a payoff c , to prove that $\text{cVal}(h, \sigma) \geq c$.
- If Prover chooses to contest a , the game proceeds to a subgame \mathcal{A} , where Challenger has to find a valid continuation path in $(\mathcal{M}' \times G)$ that yields a payoff a , to prove that $\text{aVal}(h, \sigma') \leq a$.

Informally, if $\sigma \not\preceq \sigma'$, Challenger is able to select correctly a non-dominance witness h of σ by σ' , and the two values $c = \text{cVal}(h, \sigma)$ and $a = \text{aVal}(h, \sigma')$ such that $c > a$. Thus, he can follow in \mathcal{G}' the path \tilde{h} corresponding to h , then continue, depending on the choice of Prover, to follow either a continuation of h that yields a payoff c with strategy σ or a continuation of h that yields a payoff a with strategy σ' . Symmetrically, if $\sigma \preceq \sigma'$, then for any history h compatible with σ and σ' where $\sigma(h) \neq \sigma'(h)$, we have that $\text{cVal}(h, \sigma) \leq \text{aVal}(h, \sigma')$. Thus any choice of pair (c, a) with $c > a$ by Challenger is faulty: either $c > \text{cVal}(h, \sigma)$, in which case Prover can let the game proceed to \mathcal{C} , and Challenger will fail to expose a continuation of h that yields a payoff c with strategy σ , or $a < \text{aVal}(h, \sigma')$, in which case Prover can let the game proceed to \mathcal{A} , and Challenger will fail to expose a continuation of h that yields a payoff a with strategy σ' . As the game graph we construct for this *Prover* game has a size polynomial in the size of the strategy automata and the game graph, and as solving this game amounts to solving a polynomially bounded number of reachability and safety subgames, we obtain that the question whether $\sigma \preceq \sigma'$ is decidable in PTIME. ◀

We now expose equivalences between the decision problems we are interested in, and properties (P₁), (P₂) and (P₃) that can be decided with the use of Lemma 33.

► **Proposition 34.** Let \mathcal{G} be a generalised safety/reachability game over a graph G . Let \mathcal{M} be a Mealy automaton realizing a finite memory strategy M , and let \mathcal{S} and \mathcal{T} be parameterized automata realizing sequences $(S_n)_{n \in \mathbb{N}}$ and $(T_n)_{n \in \mathbb{N}}$ of finite memory strategies. Then:

1. Let $N_{\prec} = |G||\mathcal{S}|$.
Then $(S_n)_{n \in \mathbb{N}}$ is a chain if and only if (P₁) $S_i \preceq S_{i+1}$ for every $1 \leq i \leq N_{\prec}$.
2. Let $N_T = |G||\mathcal{T}|(|\mathcal{M}| + 1) + 1$, and suppose that $(T_n)_{n \in \mathbb{N}}$ is a chain.
Then $M \not\sqsubseteq (T_n)_{n \in \mathbb{N}}$ if and only if (P₂) $M \not\preceq T_{N_T}$.
3. Let $N_S = |G||\mathcal{S}|(2|\mathcal{T}| + 1)$, and suppose that $(S_n)_{n \in \mathbb{N}}$ and $(T_n)_{n \in \mathbb{N}}$ are chains.
Then $(S_n)_{n \in \mathbb{N}} \not\sqsubseteq (T_n)_{n \in \mathbb{N}}$ if and only if (P₃) $S_{N_S} \not\preceq (T_n)_{n \in \mathbb{N}}$.

Proof sketch. Note that for every item, the backward implication is straightforward. The proof of each forward implication relies on the study of the loops that appear in witnesses of non dominance, whose existence is guaranteed by Lemma 23. For item 1, we prove that, given a witness of non-dominance of T_i by T_{i+1} for any integer $i > N_{\prec}$, we are able to construct a witness of non-dominance of T_j by T_{j+1} for some $j \leq N_{\prec}$ by exposing loops that can be pumped down.

To prove item 2, we show that since $(T_n)_{n \in \mathbb{N}}$ is a chain, $M \not\sqsubseteq (T_n)_{n \in \mathbb{N}}$ if and only if M is not dominated by T_N for arbitrarily large N . If M is dominated by T_{N_T} , we exhibit a loop in a witness of non dominance, which, once pumped, allows us to create witnesses of non dominance of M by T_N for arbitrarily large N , yielding the desired result.

Finally, item 3 is proved as follows. Since $(S_n)_{n \in \mathbb{N}}$ and $(T_n)_{n \in \mathbb{N}}$ are chains, $(S_n)_{n \in \mathbb{N}} \not\sqsubseteq (T_n)_{n \in \mathbb{N}}$ if and only if there exists an integer N such that $S_N \not\preceq (T_n)_{n \in \mathbb{N}}$. Once again, we show that if such an N exists, there is at least one that is smaller than N_S . ◀

Since the property P₁ can be decided in PTIME by applying Lemma 33 with adequately chosen Mealy automata as parameters, we obtain the following theorem.

► **Theorem 35.** *Given a generalised safety/reachability game and a parameterized automaton, we can decide in PTIME whether the automaton realizes a chain of strategies.*

Similarly, the property P₂ can be decided in PTIME by applying Lemma 33 with \mathcal{M} and the Mealy automaton corresponding to the strategy T_{N_T} as parameters. Moreover, by Proposition 34.2, the problem of deciding property P₃ can be reduced in polynomial time to the problem of deciding property P₂. Therefore Proposition 34.3 implies our final decidability result.

► **Theorem 36.** *Given a generalised safety/reachability game and two parameterized automata realizing uniform chains of strategies, we can decide in PTIME whether the chain realized by the first is dominated by the one from the second.*

5 Conclusion and outlook

In quantitative games with more than three possible payoffs, there are strategies that are dominated but not dominated by any admissible strategy. Example 1 suggests that chains of strategies could provide a suitable framework to circumvent this issue. Abstract order-theoretic considerations revealed that in the most general case, this does not work. However, if we restrict to countable collections of strategies, every chain is below a maximal chain. This restriction is very natural, as it covers all computable strategies.

We explored the abstract approach in the concrete setting of generalized safety/reachability games. Here, parameterized automata can give a very concrete meaning to chains of strategies. Several fundamental algorithmic questions are decidable in PTIME. There are

more algorithmic questions to investigate: first and foremost, deciding, given a parameterized automaton, whether the chain realized is maximal or not, is a relevant question left open.

Moreover, our results on this class of games mostly rely on the prefix-independence and finite-range of the payoff function, and on the restriction to finite-memory strategies. Thus, it seems achievable to extend our approach to other classes of games that enjoy these properties, such as quantitative extensions of parity or Muller games, in the sense of [20] and [24]. A more ambitious objective would be to tackle more general classes of games, starting by dropping the finite-range hypothesis to encompass, for instance, mean-payoff games [14].

Finally, in the boolean case, in addition to the fundamental property that a strategy is either admissible or dominated by an admissible strategy, the admissibility notion exhibits other good properties. Indeed, in [4], the author proves that, in games with ω -regular winning conditions on finite graphs, the set of admissible strategies is itself an ω -regular set. Furthermore, as shown in [11], assuming all the players are rational (that is, play admissible strategies) yields robust and resilient solutions for strategy synthesis.

This synthesis problem remains to be investigated in the quantitative setting.

References

- 1 Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In *ICALP'89*, volume 372 of *LNCS*. Springer, 1989.
- 2 Nicolas Basset, Gilles Geeraerts, Jean-François Raskin, and Ocan Sankur. Admissibility in concurrent games. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 123:1–123:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.123.
- 3 Nicolas Basset, Ismaël Jecker, Arno Pauly, Jean-François Raskin, and Marie Van den Boogaard. Beyond admissibility: Dominance between chains of strategies. arXiv 1805.11608, 2018. URL: <https://arxiv.org/abs/1805.11608>.
- 4 Dietmar Berwanger. Admissibility in infinite games. In *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, pages 188–199, 2007. doi:10.1007/978-3-540-70918-3_17.
- 5 Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. On Nash equilibria and improvement cycles in pure positional strategies for chess-like and backgammon-like n -person games. *Discrete Mathematics*, 312(4):772–788, 2012. doi:10.1016/j.disc.2011.11.011.
- 6 Endre Boros, Vladimir Gurvich, and Emre Yamangil. Chess-like games may have no uniform nash equilibria even in mixed strategies. *Game Theory*, 2013. doi:10.1155/2013/534875.
- 7 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In *Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2016.
- 8 Romain Brenguier, Arno Pauly, Jean-François Raskin, and Ocan Sankur. Admissibility in Games with Imperfect Information (Invited Talk). In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *LIPICs*, pages 2:1–2:23. Schloss Dagstuhl, 2017. doi:10.4230/LIPICs.CONCUR.2017.2.
- 9 Romain Brenguier, Guillermo A. Pérez, Jean-François Raskin, and Ocan Sankur. Admissibility in quantitative graph games. In *36th IARCS Annual Conference on Foundations of*

- Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, pages 42:1–42:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.42.
- 10 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. *Acta Inf.*, 54(1):41–83, 2017.
 - 11 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. *Acta Inf.*, 54(1):41–83, 2017.
 - 12 Romain Brenguier, Jean-François Raskin, and Mathieu Sassolas. The complexity of admissibility in omega-regular games. In *CSL-LICS '14, 2014*. ACM, 2014. doi:10.1145/2603088.2603143.
 - 13 Thomas Brihaye, Veronique Bruyère, and Julie De Pril. Equilibria in quantitative reachability games. In *Proc. of CSR*, volume 6072 of *LNCS*. Springer, 2010. doi:10.1007/978-3-642-13182-0_7.
 - 14 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, Jun 1979. doi:10.1007/BF01768705.
 - 15 Marco Faella. Admissible strategies in infinite games over graphs. In *MFCS 2009*, volume 5734 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2009.
 - 16 Deedlit (<https://math.stackexchange.com/users/21465/deedlit>). Height of a certain order. Mathematics Stack Exchange. URL:<https://math.stackexchange.com/q/2436983> (version: 2017-09-20). URL: <https://math.stackexchange.com/q/2436983>.
 - 17 Orna Kupferman. On high-quality synthesis. In S. Alexander Kulikov and J. Gerhard Woeginger, editors, *11th International Computer Science Symposium in Russia, CSR 2016*, pages 1–15. Springer International Publishing, 2016. doi:10.1007/978-3-319-34171-2_1.
 - 18 Stéphane Le Roux and Arno Pauly. Extending finite memory determinacy. *Information and Computation*, 201X. doi:10.1016/j.ic.2018.02.024.
 - 19 George Markowsky. Chain-complete posets and directed sets with applications. *Algebra Universalis*, 6:53–68, 1976. doi:10.1007/BF02485815.
 - 20 Soumya Paul and Sunil Easaw Simon. Nash equilibrium in generalised muller games. In *FSTTCS*, volume 4 of *LIPICs*, pages 335–346. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.
 - 21 Arno Pauly. The computational complexity of iterated elimination of dominated strategies. *Theory of Computing Systems*, pages 52–75, 2016. doi:10.1007/s00224-015-9637-1.
 - 22 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989. doi:10.1145/75277.75293.
 - 23 H.J. Prömel, W. Thumser, and B. Voigt. Fast growing functions based on Ramsey theorems. *Discrete Mathematics*, 95(1):341–358, 1991. doi:0012-365X(91)90346-4.
 - 24 Stéphane Le Roux. From winning strategy to nash equilibrium. *Math. Log. Q.*, 60(4-5):354–371, 2014.
 - 25 Wang Shang-Zhi and Li Bo-Yu. On the minimal cofinal subsets of a directed quasi-ordered set. *Discrete Mathematics*, 48(2):289–306, 1984. doi:10.1016/0012-365X(84)90189-4.

Here we present some, but not all omitted proofs. For the complete account, we refer to the arXiv version at [3].

A Proofs omitted from Section 3

► **Lemma 37.** *If $(x_\beta)_{\beta < \alpha} \sqsubseteq (y_\gamma)_{\gamma < \delta}$ and $\alpha < \text{cof}((y_\gamma)_{\gamma < \delta})$, then there exists $\gamma_0 < \delta$ such that*

$$(x_\beta)_{\beta < \alpha} \sqsubseteq (y_{\gamma_i})_{i < 1}$$

Proof of Lemma 9. It is clear that 2 is a special case of 1. We thus just need to show that any potential obstruction to 1 can be assumed to have the form in 2.

10:18 Dominance Between Chains of Strategies

By replacing each $(x_\beta^\gamma)_{\beta < \alpha_\gamma}$ with some suitable cofinal increasing chain if necessary, we can assume that $\text{cof}((x_\beta^\gamma)_{\beta < \alpha_\gamma}) = \alpha_\gamma$ for all $\gamma < \delta$.

Consider $\{(x_\beta^\gamma)_{\beta < \alpha_\gamma} \mid \exists \gamma' > \gamma \ \alpha_\gamma < \alpha_{\gamma'}\}$. If this set is cofinal in $((x_\beta^\gamma)_{\beta < \alpha_\gamma})_{\gamma < \delta}$, then for each γ inside that set pick some witness γ' , and let y_γ be the witness obtained from Lemma 37. Now $\{y_\gamma \mid \exists \gamma' > \gamma \ \alpha_\gamma < \alpha_{\gamma'}\}$ is the desired upper bound.

If the set from the paragraph above is not cofinal, then there exists some $\delta' < \delta$ such that for $\delta' \leq \gamma < \gamma' < \delta$ we always have that $\alpha_\gamma \geq \alpha_{\gamma'}$. As the α_γ are ordinals, decreases can happen only finitely many times. Thus, by moving to a suitable cofinal subset we can safely assume that all α_γ are equal to some fixed α .

Again by moving to a suitable cofinal subset, we can assume that $\text{cof}(((x_\beta^\gamma)_{\beta < \alpha})_{\gamma < \delta}) = \delta$. If $\delta = 1$, the statement is trivial. If $\alpha = 1$, then $(x_0^\gamma)_{\gamma < \delta}$ is the desired upper bound. It remains to handle the case $\alpha = \delta > 1$.

We construct some function $f : \alpha \rightarrow \alpha$, such that the desired upper bound $(y_\epsilon)_{\epsilon < \alpha}$ is of the form $y_\epsilon = x_{f(\epsilon)}^\epsilon$. We proceed as follows: Set $f(0) = 0$. Once $f(\zeta)$ has been defined for all $\zeta < \epsilon$, pick for each $\zeta < \epsilon$ some $g(\zeta)$ such that $x_{f(\zeta)}^\zeta \preceq x_{g(\zeta)}^\epsilon$ and $x_\epsilon^\zeta \preceq x_{g(\zeta)}^\epsilon$. As $\epsilon < \alpha$, it cannot be that $\{x_{g(\zeta)}^\epsilon \mid \zeta < \epsilon\}$ is cofinal in $\{x_\beta^\epsilon \mid \beta < \alpha\}$. Thus, it has some upper bound, and we define $f(\epsilon)$ such that $x_{f(\epsilon)}^\epsilon$ is such an upper bound. ◀

Proof of Proposition 16. For each countable limit ordinal α , we fix² some fundamental sequence $(\alpha[m])_{m < \omega}$ of ordinals with $\alpha[m] < \alpha$ and $\sup_{m \in \omega} \alpha[m] = \alpha$.

Let $f_n^0(k) = \max\{f(k), k\}$. Let $f_n^{\alpha+1}(k) = \max_{j \leq k} (f_{n+j}^\alpha)(k) + 1$, and for limit ordinals α , let $f_n^\alpha(k) = \max_{m \leq n+k} f_n^{\alpha[m]}(k)$.

Claim: If $\alpha \leq \beta$, then $(f_n^\alpha)_{n < \omega} \sqsubseteq (f_m^\beta)_{m < \omega}$.

Proof. It suffices to show that if $\alpha \leq \beta$, then $f_n^\alpha \leq f_n^\beta$ for all n greater than some t . If $\beta = \alpha + 1$, this is immediate already for $t = 0$. For β a limit ordinal, we note that $f_n^{\beta[m]} \leq f_n^\beta$ for $n \geq m$.

The claim then follows by induction over β . Recall that if β is a limit ordinal and $\alpha < \beta$, then there is some $m \in \omega$ with $\alpha \leq \beta[m]$. Since for any given α, β , the ordinals γ between α and β we will need to inspect in the induction form a decreasing chain, there are only finitely many such ordinals. In particular, the maximum of all thresholds t we encounter is well-defined. ◀

Claim: If $\alpha > \beta$, then $(f_n^\alpha)_{n < \omega} \not\sqsubseteq (f_m^\beta)_{m < \omega}$.

Proof. Due to transitivity of \sqsubseteq and the previous claim, it suffices to show that $(f_m^{\alpha+1})_{m < \omega} \not\sqsubseteq (f_n^\alpha)_{n < \omega}$. Write $g_n = f_n^\alpha$. Assume the contrary, i.e. that for all $n < \omega$ there exists some $m < \omega$ such that for all $k \in \mathbb{N}$ and for all $j \leq k$ we have that $g_{n+j}(k) + 1 \leq g_m(k)$. In particular, for $n = 0$ we would have that $\forall k \in \mathbb{N} \ \forall j \leq k \ g_j(k) + 1 \leq g_m(k)$, and then setting $k = j = m$, that $g_m(m) + 1 \leq g_m(m)$, which is a contradiction. ◀

² We have no computability or other uniformity requirements to satisfy, and can thus just invoke the axiom of choice. Otherwise, as discussed e.g. in [23, Section 3.1] this approach would fail.

B Proofs omitted from Subsection 4.1

Proof of Lemma 23.

\implies Suppose that for every history h compatible with σ_1 and σ_2 such that $\text{last}(h) \in V_i$ and $\sigma_1(h) \neq \sigma_2(h)$, we have that $\text{cVal}(h, \sigma_1) \leq \text{aVal}(h, \sigma_2)$. We show that $\sigma_1 \preceq \sigma_2$. Let τ be a strategy of Player $-i$. Consider $\rho_1 = \mathbf{Out}(\sigma_1, \tau)$ and $\rho_2 = \mathbf{Out}(\sigma_2, \tau)$. If for all prefixes $h' \subseteq_{\text{pref}} \rho_1$ such that $\text{last}(h') \in V_i$, it holds that $\sigma_1(h') = \sigma_2(h')$, then in fact $\rho_1 = \rho_2$ and $p_i(\sigma_1, \tau) = p_i(\sigma_2, \tau)$. Otherwise, let h be the least common prefix of ρ_1 and ρ_2 such that $\text{last}(h) \in V_i$ and $\sigma_1(h) \neq \sigma_2(h)$. We know that $p_i(\rho_1) \leq \text{cVal}(h, \sigma_1)$ and $p_i(\rho_2) \geq \text{aVal}(h, \sigma_2)$ since $h \subseteq_{\text{pref}} \rho_1$ and $h \subseteq_{\text{pref}} \rho_2$. As $\text{cVal}(h, \sigma_1) \leq \text{aVal}(h, \sigma_2)$, we have that $p_i(\sigma_1, \tau) \leq p_i(\sigma_2, \tau)$. Thus, for every $\tau \in \Sigma_{-i}$, it holds that $p_i(\sigma_1, \tau) \leq p_i(\sigma_2, \tau)$, that is, $\sigma_1 \preceq \sigma_2$.

\Leftarrow Let h be a history compatible with σ_1 and σ_2 such that $\text{last}(h) \in V_i$, $\sigma_1(h) \neq \sigma_2(h)$ and $\text{cVal}(h, \sigma_1) > \text{aVal}(h, \sigma_2)$. Then, there exists two strategies τ_1 and τ_2 of player $-i$ such that $p_i(h, \sigma_1, \tau_1) = \text{cVal}(h, \sigma_1)$ and $p_i(h, \sigma_2, \tau_2) = \text{aVal}(h, \sigma_2)$. Let τ be a strategy

$$\text{of player } -i \text{ compatible with } h, \text{ and define } \tau'(h') = \begin{cases} \tau_1(h') & \text{if } h\sigma_1(h) \subseteq_{\text{pref}} h', \\ \tau_2(h') & \text{if } h\sigma_2(h) \subseteq_{\text{pref}} h', \\ \tau(h') & \text{otherwise.} \end{cases}$$

The strategy τ' is well defined, as $\sigma_1(h) \neq \sigma_2(h)$. Furthermore, we have that $p_i(\sigma_1, \tau') = p_i(h, \sigma_1, \tau_1) = \text{cVal}(h, \sigma_1) > \text{aVal}(h, \sigma_2) = p_i(h, \sigma_2, \tau_2) = p_i(\sigma_2, \tau')$, since generalised safety/reachability games are prefix-independent. Thus, $\sigma_1 \not\preceq \sigma_2$. \blacktriangleleft

Proof of Lemma 27. For each vertex v in the game, we fix a finite memory strategy τ^v that realizes $\text{aVal}(v)$ and $\text{acVal}(v)$. Note that since generalised safety/reachability games are prefix independent, values depend only on the current vertex, but not on the entire history.

We start with a finite memory strategy σ . If it is not already preadmissible, then it has witnesses of non-admissibility violating the desired property. Whether a history h is a witness of non-admissibility for a finite memory strategy σ depends only on the last vertex of h and the current state of σ . We now modify σ such that whenever σ is in a combination of vertex v and state s corresponding to a problematic witness of non-admissibility, the new strategy σ' moves to playing τ^v instead. The choices of v , s and τ^k ensure that σ' dominates σ .

The new strategy σ' may fail to be preadmissible, again, and we repeat the construction. Now any problematic history in σ' needs to enter the automaton for some τ^v at some point. By choice of τ^v , the history where τ^v has just been entered cannot be a witness of non-admissibility. It follows that a problematic history entering τ^v cannot end in v . Repeating the updating process for at most as many times as there are vertices in the game graph will yield a preadmissible finite memory strategy dominating σ . \blacktriangleleft

C Proofs omitted from Subsection 4.2

To complete the proof of Theorem 31, we need the following intermediary results:

► **Lemma 38.** *If h is not a witness of non-admissibility of σ , and not a witness of non-dominance of σ by τ , then h is not a witness of non-dominance of τ by σ .*

► **Lemma 39.** *Given an initialized game with initial vertex v_0 , the following holds: If for two strategies σ and τ it holds that for any maximal history h compatible with both, there is a prefix h' with $\text{aVal}(h', \sigma) = \text{aVal}(h', \tau)$ and $\text{cVal}(h', \sigma) = \text{cVal}(h', \tau)$, then $\text{aVal}(v_0, \sigma) = \text{aVal}(v_0, \tau)$ and $\text{cVal}(v_0, \sigma) = \text{cVal}(v_0, \tau)$.*

► **Lemma 40.** *Given an initialized game with initial vertex v_0 , the following holds: If σ is preadmissible and $\sigma \preceq \tau$, then $\text{aVal}(v_0, \sigma) = \text{aVal}(v_0, \tau)$ and $\text{cVal}(v_0, \sigma) = \text{cVal}(v_0, \tau)$.*

Proof. We show that the conditions of Lemma 39 are satisfied, which will imply our desired conclusion. Consider a maximal history h compatible with both σ and τ . First, assume that h is not a witness of non-admissibility of σ . Since $\sigma \preceq \tau$, by Lemma 23 h cannot be a witness of non-dominance of σ by τ , i.e. $\text{cVal}(h, \sigma) \leq \text{aVal}(h, \tau)$. By Lemma 38, it follows that h is not a witness of non-dominance of τ by σ either, i.e. $\text{cVal}(h, \tau) \leq \text{aVal}(h, \sigma)$. Put together, we have $\text{aVal}(h, \sigma) = \text{cVal}(h, \sigma) = \text{aVal}(h, \tau) = \text{cVal}(h, \tau)$.

It remains the case where h is a witness of non-admissibility of σ . Then by preadmissibility of σ , h has some prefix h' with $\text{aVal}(h', \sigma) = \text{aVal}(h')$ and $\text{cVal}(h', \sigma) = \text{acVal}(h')$. Since $\sigma \preceq \tau$, we must have $\text{aVal}(h', \sigma) \leq \text{aVal}(h', \tau)$, so it follows that $\text{aVal}(h', \sigma) = \text{aVal}(h', \tau)$, and then that $\text{cVal}(h', \tau) \leq \text{acVal}(h') = \text{cVal}(h', \sigma) \leq \text{cVal}(h', \tau)$, i.e. $\text{cVal}(h', \sigma) = \text{cVal}(h', \tau)$. ◀

Proof of Theorem 31. By Lemma 27 it suffices to prove the claim for preadmissible strategies (Definition 26). We thus start with a preadmissible finite memory strategy σ .

Preliminaries. Since we are working with prefix-independent outcomes and strategies realized by automata, we see that any of the values of σ at some history h depends only on the final vertex v of h and the state s the strategy σ is in after reading h . We can thus overload our notation to write $\text{aVal}(v, s)$ for $\text{aVal}(h, \sigma)$ and $\text{aVal}(v)$ for $\text{aVal}(h)$, and so on. In particular, whether some history h is a witness of non-admissibility of σ or not depends only on the final vertex v of h and the state s that σ is in after reading h . Let WNA be the set of such pairs (v, s) corresponding to non-admissibility witnesses. By the definition of preadmissibility, we cannot reach any $(v, s) \in \text{WNA}$ without first passing through some (v, s_v) with $\text{aVal}(v, s_v) = \text{aVal}(v)$ and $\text{cVal}(v, s_v) = \text{acVal}(v, s_v)$. By expanding the automaton if necessary (to remember where we were when first encountering some vertex), we can assume that for any $(v, s) \in \text{WNA}$ there is canonic choice of prior (v, s_v) .

► **Lemma 41.** *For any $(v, s) \in \text{WNA}$ and corresponding (v, s_v) we find that $\text{aVal}(v, s_v) = \text{aVal}(v, s) = \text{cVal}(v, s) < \text{cVal}(v, s_v)$.*

The construction. We now construct a parameterized automaton \mathcal{M} from σ that either realizes a single maximal strategy, or a maximal uniform chain. The parameterized automaton is identical to the one realizing σ everywhere except at the $(v, s) \in \text{WNA}$. In particular, if $\text{WNA} = \emptyset$, we are done. Otherwise, for each $(v, s) \in \text{WNA}$ we make the following modifications: If $\text{aVal}(v, s_v) \leq 0$, we modify the automaton to act in (v, s) as it does in (v, s_v) . If $\text{aVal}(v, s_v) > 0$, then we add green edges to let the automaton act in (v, s) as in (v, s_v) , and red edges to act as it would do originally.

Correctness. The comparison of the values lets us conclude via Lemma 23 that the parameterized automaton \mathcal{M} either realizes a single strategy dominating σ , or a uniform chain dominating σ .

It remains to argue that the strategy/uniform chain realized by \mathcal{M} is maximal. Let σ_n be the strategy where \mathcal{M} is initialized with $n \in \mathbb{N}$. Assume that $\tau \succ \sigma_n$, and let h be a witness of $\tau \not\preceq \sigma_n$ according to Lemma 23, i.e. satisfying $\text{cVal}(h, \tau) > \text{aVal}(h, \sigma_n)$. Since $\sigma_n \preceq \tau$, we have $\text{cVal}(h, \sigma_n) \leq \text{aVal}(h, \tau)$, so $\text{aVal}(h, \sigma_n) \leq \text{cVal}(h, \sigma_n) \leq \text{aVal}(h, \tau) \leq \text{cVal}(h, \tau)$ with one inequality being strict. In particular, h is a witness of non-admissibility of σ_n . By construction of \mathcal{M} the next move after h must be given by a red edge. This already implies that if \mathcal{M} realizes a single strategy, then that strategy is maximal.

Let m be the size of the parameterized automaton \mathcal{M} , let t be the size of the automaton realizing τ , and $N = mt + 1$.

► **Lemma 42.** *At any maximal history compatible with σ_N and τ , σ_N will follow a green or black edge next.*

Proof. Assume there were such a history hv compatible with both σ_N and τ where σ_N is about to apply a red edge, being in state s . If the combination (v, s) has been reached more than t times during hv , then it has to hold that on histories extending hv , τ always acts at v as \mathcal{M} does following the green edge at (v, s) , for τ cannot count up to $t + 1$ (in particular, h is maximal for being compatible with τ and σ_N). It follows that $\text{aVal}(hv, \tau) \leq 0$. Let $h'v$ be a prefix of this form of hv compatible with σ_n not ending in a red edge (this exists, since $n > m$). Then $\text{aVal}(h'v, \tau) \leq 0$, and since $\tau \succeq \sigma_n$, $\text{aVal}(h'v, \sigma_n) = \text{aVal}(h'v, \tau) \leq 0$. But then when constructing \mathcal{M} , we would not have placed red and green edges at (v, s_v) , leading to a contradiction. Thus, at any maximal history compatible with σ_N and τ , σ_N will follow a green or black edge next.

If the combination (v, s) has been visited at most t times during hv , then there has to be some other pair of counter access state s' and vertex v' which was reached more often than t times during hv by the pigeon hole principle (for since σ_N is about to follow a red edge, it has reached a counter access state at least $N = mt + 1$ many times), with σ_N taking the green edge there. Again, by the same reasoning as above, τ always follows the green edge at the corresponding histories, leading to the conclusion that the antagonistic value obtained by τ there is 0, and ultimately a contradiction to s' being created as a counter access state when constructing \mathcal{M} . ◀

If τ is part of a chain $(\tau_i)_{i \in \mathbb{N}}$ with $(\sigma_i)_{i \in \mathbb{N}} \sqsubseteq (\tau_i)_{i \in \mathbb{N}}$, then τ and σ_N have a common upper bound τ' . We proceed to show that this suffices to conclude $\tau \preceq \sigma_N$. This completes our argument, since by induction it follows that if $(\sigma_n)_{n \in \mathbb{N}} \sqsubseteq (\tau_n)_{n \in \mathbb{N}}$, then also $(\tau_n)_{n \in \mathbb{N}} \sqsubseteq (\sigma_n)_{n \in \mathbb{N}}$.

► **Lemma 43.** *If τ and σ_N have common upper bound τ' , then $\tau \preceq \sigma_N$.*

Proof. We proceed by ruling out all candidates for witnesses of non-dominance of τ by σ_N , and conclude our claim by Lemma 23. Any candidate is a maximal history h compatible with both σ_N and τ .

Case 1. Either h is not compatible with τ' , or $\tau'(h) \neq \sigma_N(h)$.

If h is not compatible with τ' , then h has a longest prefix h' compatible with τ' . If h is compatible with τ' , but $\tau'(h) \neq \sigma_N(h)$, we set $h' = h$. By Lemma 42, h' cannot be a witness of non-admissibility of σ_N , and by Lemma 23 it cannot be a witness of non-dominance of σ_N by τ' , since $\sigma_N \preceq \tau'$. Lemma 38 then gives us that h' is not a witness of non-dominance of τ' by σ_N , i.e. $\text{cVal}(h', \tau') \leq \text{aVal}(h', \sigma_N)$. Together with $\sigma_N \preceq \tau'$ we get that $\text{aVal}(h', \sigma_N) = \text{cVal}(h', \sigma_N)$. Since h is compatible with σ_N and extends h' , it follows that $\text{aVal}(h', \sigma_N) = \text{aVal}(h, \sigma_N) = \text{cVal}(h, \sigma_N)$. Since $\tau \preceq \tau'$, it follows that $\text{cVal}(h', \tau) \leq \text{cVal}(h', \tau') = \text{aVal}(h', \sigma_N)$. Since h is compatible with τ and extends h' , it follows that $\text{cVal}(h, \tau) \leq \text{cVal}(h', \tau) \leq \text{aVal}(h', \sigma_N) = \text{aVal}(h, \sigma_N)$, i.e. that h is not a witness of non-dominance of τ by σ_N .

Case 2. h is compatible with τ' and $\tau'(h) = \sigma_N(h)$.

Consider the subgame starting after that move. Since we have chosen N sufficiently big, in this subgame it is impossible for σ_N to pass through a red edge without previously passing through a green edge at the same vertex. By construction, this ensures that σ_N is still preadmissible in this subgame. Since reaching the subgame is compatible

10:22 Dominance Between Chains of Strategies

with τ' and σ_N , restricting to this subgame, we still have that $\sigma_N \preceq \tau'$. Thus, we can apply Lemma 40 to the subgame, and conclude that $\text{aVal}(h, \tau') = \text{aVal}(h, \sigma_N)$ and $\text{cVal}(h, \tau') = \text{cVal}(h, \sigma_N)$. Since h cannot be a witness of non-dominance of τ by τ' , it holds that $\text{cVal}(h, \tau) \leq \text{aVal}(h, \tau') = \text{aVal}(h, \sigma_N)$. Thus, h is not a witness of non-dominance of τ by σ_N either. \blacktriangleleft

D Proofs omitted from Subsection 4.3

The proof of Proposition 34.1 is based on the following auxiliary Lemma, whose demonstration relies on the study of the loops that appear in witnesses of non dominance.

► **Lemma 44.** *Let \mathcal{G} be a generalised safety/reachability game, let \mathcal{M} be a parametrized automaton over the game graph of \mathcal{G} , and let $(T_n)_{n \in \mathbb{N}}$ be the sequence of finite-memory strategies realized by \mathcal{M} . Then for every pair of integers $n_1, n_2 > |G||\mathcal{M}|$ satisfying $T_{n_1} \not\preceq T_{n_2}$, there exists $0 < k \leq |G||\mathcal{M}|$ such that for every $i \in \mathbb{N}$, $T_{n_1+(i-1)k} \not\preceq T_{n_2+(i-1)k}$.*

Proof of Proposition 34.1. Let \mathcal{G} be a generalised safety/reachability game, and let \mathcal{S} be a parametrized automaton over the game graph of \mathcal{G} . We denote by $(S_n)_{n \in \mathbb{N}}$ the sequence of finite-memory strategies realized by \mathcal{S} . Let $N_{\preceq} = |G||\mathcal{S}|$.

Let $U_{\mathcal{S}}$ denote the set composed of the integers n satisfying $S_n \not\preceq S_{n+1}$. It is clear that if $U_{\mathcal{S}}$ is not empty, then $(S_n)_{n \in \mathbb{N}}$ is not a chain. Conversely, if $U_{\mathcal{S}}$ is empty, then $(S_n)_{n \in \mathbb{N}}$ is a chain, since for every pair of integers $n_1 < n_2$, we have $S_{n_1} \preceq S_{n_1+1} \preceq \dots \preceq S_{n_2}$. Let us suppose, towards building a contradiction, that the minimal element m of $U_{\mathcal{S}}$ is strictly greater than N_{\preceq} . Then, we obtain from Lemma 44 that there exists an integer $k > 0$ such that $S_{m-k} \not\preceq S_{m-k+1}$ by setting $i = 0$. This contradicts the minimality of m . As a consequence, $m \leq N_{\preceq}$. This proves that $(S_n)_{n \in \mathbb{N}}$ is a chain if and only if $S_i \preceq S_{i+1}$ for every $1 \leq i \leq N_{\preceq}$. \blacktriangleleft

Rule Algebras for Adhesive Categories

Nicolas Behr¹

IRIF, Université Paris-Diderot (Paris 07), France

Paweł Sobociński

ECS, University of Southampton, UK

Abstract

We show that every adhesive category gives rise to an associative algebra of rewriting rules induced by the notion of double-pushout (DPO) rewriting and the associated notion of concurrent production. In contrast to the original formulation of rule algebras in terms of relations between (a concrete notion of) graphs, here we work in an abstract categorical setting. Doing this, we extend the classical concurrency theorem of DPO rewriting and show that the composition of DPO rules along abstract dependency relations is, in a natural sense, an associative operation. If in addition the adhesive category possesses a strict initial object, the resulting rule algebra is also unital. We demonstrate that in this setting the canonical representation of the rule algebras is obtainable, which opens the possibility of applying the concept to define and compute the evolution of statistical moments of observables in stochastic DPO rewriting systems.

2012 ACM Subject Classification Theory of computation → Concurrency, Mathematics of computing → Markov processes

Keywords and phrases Adhesive categories, rule algebras, Double Pushout (DPO) rewriting

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.11

1 Introduction

Double pushout graph (DPO) rewriting [9] is the most well-known approach to algebraic graph transformation. The underlying rewriting mechanics are specified in terms of the universal properties of pushouts – for this reason, the approach is domain-independent and instantiates across a number of concrete notions of graphs and graph-like structures. Moreover, the introduction of adhesive and quasi-adhesive categories [11, 10] (which, roughly speaking, ensure that the pushouts involved are “well-behaved”, i.e. they satisfy similar exactness properties as pushouts in the category of sets and functions) entailed that a standard corpus of theorems [14] that ensures the “good behavior” of DPO rewriting holds if the underlying ambient category is (quasi-)adhesive.

An important classical theorem of DPO rewriting is the *concurrency theorem*, which involves an analysis of *two* DPO productions applied in series. Given a *dependency relation* (which, intuitively, determines how the right-hand side of the first rule overlaps with the left-hand side of the second), a purely category-theoretic construction results in a *composite rule* which applies the two rules simultaneously. The concurrency theorem then states that in any graph, the two rules can be applied in series in a way consistent with the relevant dependency relation if and only if the composite rule can be applied, yielding the same result.

¹ Corresponding author email: nicolas.behr@irif.fr; supported by a *Marie Skłodowska-Curie Individual Fellowship* (Grant Agreement No. 753750 – RaSiR).



The operation that takes two rules together with a dependency relation and produces a composite rule can be considered as an algebraic operation on the set of DPO productions for a given category. From this viewpoint, it is natural to ask whether this operation is associative. It is remarkable that this appears to have been open until now. Our main contribution is an elementary proof of associativity of this type of composition.

Associativity is advantageous for a number of reasons. In [2, 4], the first author and his team developed the *rule algebra* framework for a concrete notion of multigraphs. Inspired by a standard construction in mathematical physics, the operation of rule composition along a common interface yields an associative algebra: given a free vector space with basis the set of DPO rules, the product of the associative algebra takes two basis elements to a formal sum, over all possible dependency relations, of their compositions. This associative algebra is useful in applications, being the formal carrier of combinatorial information that underlies *stochastic* interpretations of rewriting. The most famous example in mathematical physics is the Heisenberg-Weyl algebra [6, 7], which served as the starting point for [2]. Indeed, [2, 4] generalized the Heisenberg-Weyl construction from mere set rewriting to multigraph rewriting. Our work, since it is expressed abstractly in terms of adhesive categories, entails that the Heisenberg-Weyl and the DPO graph rewriting rule algebra can *both* be seen as two instances of the same construction, expressed in abstract categorical terms.

Structure of the paper. Following the preliminaries in Section 2, we prove our main result in Section 3. Next, in Section 4 we give the abstract definition of rule algebra, and demonstrate that it captures the well-known Heisenberg-Weyl algebra in Section 5. We conclude with applications to combinatorics and stochastic mechanics in Sections 6 and 7.

2 Adhesive categories and Double-Pushout rewriting

We briefly review standard material, following mostly [11] (see [8, 14] for further references).

► **Definition 2.1** ([11], Def. 3.1). A category \mathbf{C} is said to be *adhesive* if

- (i) \mathbf{C} has pushouts along *monomorphisms*,
- (ii) \mathbf{C} has pullbacks, and if
- (iii) pushouts along monomorphisms are *van Kampen (VK)squares*.

Examples include **Set** (the category of sets and set functions), **Graph** (the category of directed multigraphs and graph homomorphisms), any presheaf topos, and any elementary topos [12]. One might further generalize by considering *quasi-adhesive categories* (see [11, 10]). We now recall *Double-Pushout (DPO) rewriting* in an adhesive category.

► **Definition 2.2** ([11], Def. 7.1). A span p of morphisms

$$L \xleftarrow{l} K \xrightarrow{r} R \tag{1}$$

is called a *production*. p is said to be *left linear* if l is a monomorphism, and *linear* if both l and r are monomorphisms. We denote the set of linear productions by $\text{Lin}(\mathbf{C})$. We will also frequently make use of the alternative notation $L \xrightarrow{p} R$ where $p = (L \xleftarrow{l} K \xrightarrow{r} R) \in \text{Lin}(\mathbf{C})$.

A homomorphism of productions $p \rightarrow p'$ consists of arrows, $L \rightarrow L'$, $K \rightarrow K'$ and $R \rightarrow R'$, such that the obvious diagram commutes. A homomorphism is an isomorphism when all of its components are isomorphisms. We do not distinguish between isomorphic productions.

► **Definition 2.3** ([11], Def. 7.2). Given a production p as in (1), a *match* of p in an object $C \in \text{ob}(\mathbf{C})$ is a morphism $m : L \rightarrow C$. A match is said to satisfy the *gluing condition* if there exists an object E and morphisms $g : K \rightarrow E$ and $v : E \rightarrow C$ such that (2) is a pushout.

$$\begin{array}{ccc}
 L & \xleftarrow{l} & K \\
 \downarrow m & & \downarrow g \\
 C & \xleftarrow{v} & E
 \end{array} \tag{2}$$

More concisely, the *gluing condition* holds if there is a *pushout complement* of $C \xleftarrow{m} L \xleftarrow{l} K$.

To proceed, we need to recall a number of properties of pushouts and pushout complements in adhesive categories. We start with some basic pasting properties that hold in any category.

► **Lemma 2.4.** *Given a commutative diagram as below,*

$$\begin{array}{ccccc}
 A & \longrightarrow & B & \longrightarrow & E \\
 \downarrow & & \downarrow & & \downarrow \\
 C & \longrightarrow & D & \longrightarrow & F
 \end{array}$$

- (pullback version) *if the right square is a pullback then the left square is a pullback if and only if the entire exterior rectangle is a pullback.*
- (pushout version) *If the left square is a pushout then the right square is a pushout if and only if the entire exterior rectangle is a pushout.*

► **Lemma 2.5** ([11], Lemmas 4.2, 4.3 and 4.5). *In any adhesive category:*

- (i) *Monomorphisms are stable under pushout.*
- (ii) *Pushouts along monomorphisms are also pullbacks.*
- (iii) *Pushout complements of monomorphisms (if they exist) are unique up to isomorphism.*

From here on, we will focus solely on *linear productions*, which entails due to the above statements a number of practical simplifications.

► **Definition 2.6** (compare [11], Def. 7.3). Let \mathbf{C} be an adhesive category, and denote by $\text{Lin}(\mathbf{C})$ the set of linear productions on \mathbf{C} . Given an object $C \in \mathbf{C}$ and a linear production $p \in \text{Lin}(\mathbf{C})$, we denote the *set of admissible matches* $\mathcal{M}_p(C)$ as the set of monomorphisms $m : L \hookrightarrow C$ for which m satisfies the *gluing condition*. As a consequence, there exists objects and morphisms such that in the diagram below both squares are pushouts:

$$\begin{array}{ccccc}
 L & \xleftarrow{l} & K & \xleftarrow{r} & R \\
 m \downarrow & & \downarrow k & & \downarrow m' \\
 C & \xleftarrow{l'} & K' & \xleftarrow{r'} & D
 \end{array} \tag{3}$$

We write $p_m(C) := D$ for the object “produced” by the above diagram. The process is called *derivation* of C along production p and admissible match m , and denoted $C \xRightarrow[p,m]{} p_m(C)$.

Note that by virtue of Lemma 2.5, the object $p_m(C)$ produced via a given derivation of an object C along a linear production p and an admissible match m is *unique up to isomorphism*. From here on, we will refer to linear productions as *linear (rewriting) rules*. Next, we recall the concept of (*concurrent*) *composition* of linear rules.

3 Concurrent composition and associativity

Convention. unless mentioned otherwise, all arrows are assumed to be monomorphisms.

For rules $p_1, p_2 \in \text{Lin}(\mathbf{C})$, a *dependency relation* consists of an object X_{12} and a span of monomorphisms $\mathbf{m} : R_1 \xleftarrow{x_1} X_{12} \xrightarrow{x_2} L_2$, s.t. K_{12}, K_{21} and morphisms illustrated below exist, where the cospan $R_1 \rightarrow Y_{12} \leftarrow L_2$ is the pushout of \mathbf{m} , and the two indicated regions are also pushouts; i.e. there exist pushouts complements of $K_1 \xrightarrow{r_1} R_1 \rightarrow Y_{12}$ and $K_2 \xrightarrow{l_2} L_2 \rightarrow Y_{12}$.

$$\begin{array}{ccccc}
 & & r'_1 & & l'_2 \\
 & & \swarrow & & \searrow \\
 K_{21} & \cdots & & Y_{12} & \cdots & K_{12} \\
 & & \swarrow & \downarrow & \searrow & \\
 & & & & & \\
 K_1 & \xrightarrow{r_1} & R_1 & \xleftarrow{x_1} & X_{12} & \xrightarrow{x_2} & L_2 & \xleftarrow{l_2} & K_2
 \end{array} \quad (4)$$

Intuitively, the existence of the left and right pushout diagrams amounts to the two rules agreeing on the overlap specified by X_{12} , and amenable to being executed concurrently. We refer to such \mathbf{m} as an *admissible match* of p_2 in p_1 and denote the set of these by $p_2 \Vdash p_1$.

Algebraically speaking, given p_1, p_2 and $\mathbf{m} \in p_2 \Vdash p_1$, we can consider “concurrent execution” to be an operation that composes p_1 and p_2 “along” \mathbf{m} to obtain a rule $p_2 \blacktriangleleft^{\mathbf{m}} p_1$. To obtain $p_2 \blacktriangleleft^{\mathbf{m}} p_1$, we extend (4) by taking two further pushouts (marked with dotted arrows) and take a pullback (marked with dashed arrows):

$$\begin{array}{ccccccc}
 & & & & Z_{12} & & \\
 & & & & \downarrow & & \\
 & & y_1 & & & & y_2 \\
 & & \swarrow & & \searrow & & \\
 L_{12} & \xleftarrow{l'_1} & K_{21} & \xrightarrow{r'_1} & Y_{12} & \xleftarrow{l'_2} & K_{12} & \xrightarrow{r'_2} & R_{12} \\
 & \lrcorner & \uparrow & \swarrow & \downarrow & \searrow & \uparrow & \lrcorner & \\
 & & & & & & & & \\
 L_1 & \xleftarrow{l_1} & K_1 & \xrightarrow{r_1} & R_1 & \xleftarrow{x_1} & X_{12} & \xrightarrow{x_2} & L_2 & \xleftarrow{l_2} & K_2 & \xrightarrow{r_2} & R_2
 \end{array} \quad (5)$$

Now we define the *composite of p_1 with p_2 along m* as

$$p_2 \blacktriangleleft^{\mathbf{m}} p_1 := (L_{12} \xleftarrow{z_1} Z_{12} \xrightarrow{z_2} R_{12}), \quad z_1 := l'_1 \circ y_1, \quad z_2 := r'_2 \circ y_2. \quad (6)$$

The following well-known result shows that composition is compatible with application.

► **Theorem 3.1** (Concurrency Theorem; [11], Thm. 7.11). *Let $p, q \in \text{Lin}(\mathbf{C})$ be two linear rules and $C \in \text{ob}(\mathbf{C})$ an object.*

- *Given a two-step sequence of derivations $C \xRightarrow{p, m} p_m(C) \xRightarrow{q, n} q_n(p_m(C))$, there exists a composite rule $r = p_2 \blacktriangleleft^{\mathbf{d}} p_1$ for unique $\mathbf{d} \in q \Vdash p$, and a unique admissible match $e \in \mathcal{M}_r(C)$, such that $C \xRightarrow{r, e} r_e(C)$ and $r_e(C) \cong q_n(p_m(C))$.*
- *Given a dependency relation $\mathbf{d} \in q \Vdash p$, $r = p_2 \blacktriangleleft^{\mathbf{d}} p_1$ and an admissible match $e \in \mathcal{M}_r(C)$, there exists a unique pair of admissible matches $m \in \mathcal{M}_p(C)$ and $n \in \mathcal{M}_q(p_m(C))$ such that $C \xRightarrow{p, m} p_m(C) \xRightarrow{q, n} q_n(p_m(C))$ with $q_n(p_m(C)) \cong r_e(C)$.*

The following technical lemma will be of use when proving our main result.

► **Lemma 3.2** (Admissibility is compatible with composition). *Suppose that $p_1, p_2 \in \text{Lin}(\mathbf{C})$ and suppose that $m_{(12)3} \in p_3 \Vdash (p_2 \blacktriangleleft^{m_{12}} p_1)$. Let $p_2 \blacktriangleleft^{m_{12}} p_1$ be as shown in (6), computed as in (5). Let $p'_2 = Y_{12} \xleftarrow{l'_2} K_{12} \xrightarrow{r'_2} R_{12}$. Then $m_{(12)3} \in p'_2 \Vdash p_3$.*

Proof. By the assumption $m_{(12)3} \in p_3 \Vdash (p_2 \overset{m_{12}}{\blacktriangleleft} p_1)$, there exists the pushout below left.

$$\begin{array}{ccc}
 Z'_{12} \dashrightarrow Y_{(12)3} & Z'_{12} \longrightarrow K'_{12} \longrightarrow Y_{(12)3} \\
 \uparrow \dashrightarrow \uparrow & \uparrow \longrightarrow \uparrow \longrightarrow \uparrow \\
 Z_{12} \longrightarrow R_{12} & Z_{12} \longrightarrow K_{12} \longrightarrow R_{12}
 \end{array}$$

By construction (see (5)), the arrow $Z_{12} \rightarrow R_{12}$ factors through K_{12} . Taking the pushout of the span $Z'_{12} \leftarrow Z_{12} \rightarrow K_{12}$ results in the diagram drawn above right. Since the whole region and the left square are pushouts, the right square is a pushout (Lemma 2.4). ◀

We now show that *concurrent composition of linear rules* is, in a natural sense, associative.

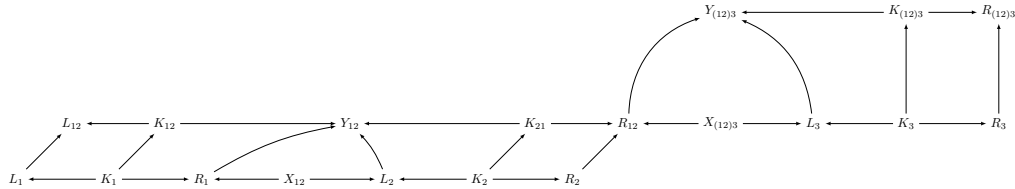
► **Theorem 3.3** (Associativity Theorem). *The composition operation $\cdot \blacktriangleleft \cdot$ is associative in the following sense: given linear rules $p_1, p_2, p_3 \in \text{Lin}(\mathbf{C})$, there exists a bijective correspondence between pairs of admissible matches $m_{21} \in p_2 \Vdash p_1$ and $m_{3(21)} \in p_3 \Vdash (p_2 \overset{m_{12}}{\blacktriangleleft} p_1)$, and pairs of admissible matches $m_{32} \in p_3 \Vdash p_2$ and $m_{(32)1} \in (p_3 \overset{m_{23}}{\blacktriangleleft} p_2) \Vdash p_1$ such that*

$$p_3 \overset{m_{3(21)}}{\blacktriangleleft} (p_2 \overset{m_{21}}{\blacktriangleleft} p_1) = (p_3 \overset{m_{32}}{\blacktriangleleft} p_2) \overset{m_{(32)1}}{\blacktriangleleft} p_1. \tag{7}$$

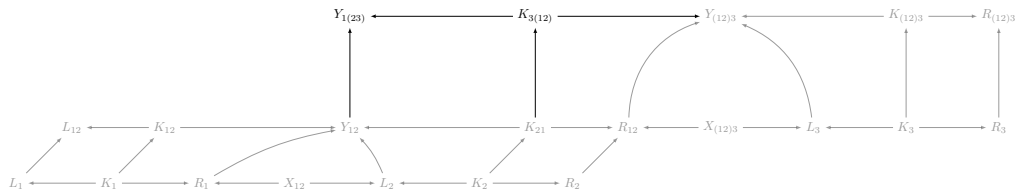
Proof. Since DPO derivations are symmetric, it suffices to show one side of the correspondence. Our proof is constructive, demonstrating how, given a pair of admissible matches

$$(m_{21} \in p_2 \Vdash p_1 \text{ and } m_{3(21)} \in p_3 \Vdash (p_2 \overset{m_{12}}{\blacktriangleleft} p_1)),$$

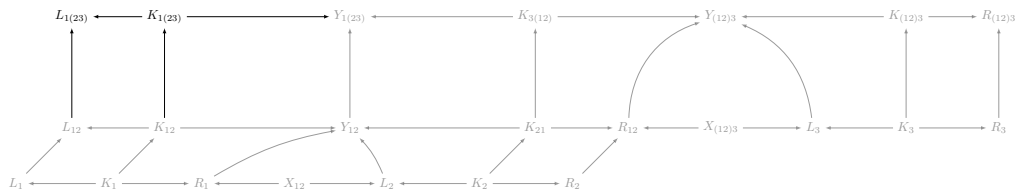
one obtains $m_{32} \in p_3 \Vdash p_2$ and $m_{(32)1} \in (p_3 \overset{m_{32}}{\blacktriangleleft} p_2) \Vdash p_1$ satisfying (7). We begin with $p_2 \overset{m_{21}}{\blacktriangleleft} p_1, p_3$ and the dependency relation $m_{3(21)}$, illustrated below.



By Lemma 3.2, since the match $m_{3(21)}$ is by assumption admissible, we can find a pushout complement and pushout to extend the above diagram as follows,

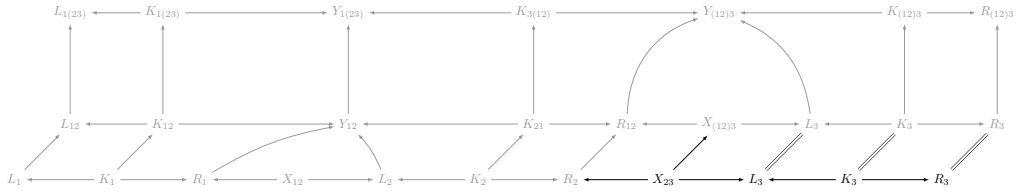


and again as below.

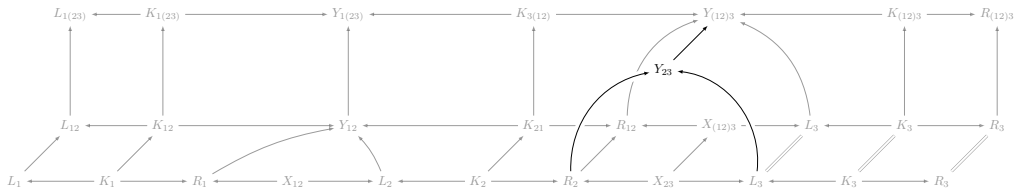


11:6 Rule Algebras for Adhesive Categories

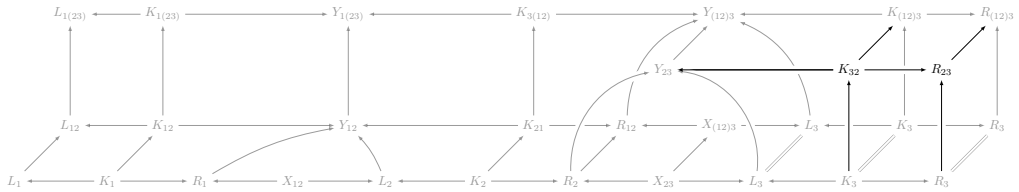
In the next step, we compute X_{23} as the evident pullback. Then we further extend the diagram via repeating the components of rule p_3 .



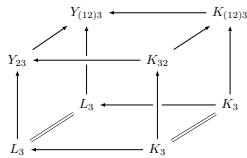
Now we push out R_2 and L_3 along X_{23} , obtaining $Y_{23} \rightarrow Y_{(12)3}$ from the universal property.



Next, we compute K_{32} by pulling back Y_{23} and $K_{1(23)}$ along $Y_{(12)3}$. We obtain $K_3 \rightarrow K_{32}$ from the universal property. To obtain the other morphisms, push out K_{32} and R_3 along K_3 .

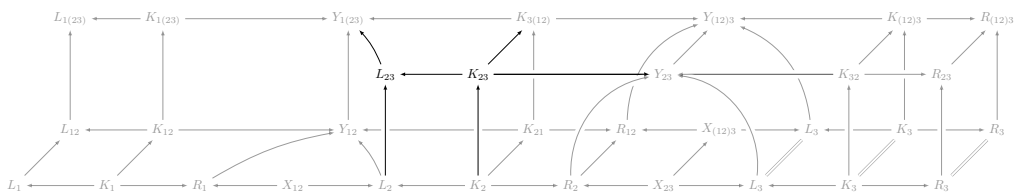


We need to establish that the newly constructed front face on the left is a pushout. To do so, let us consider the cube on the left in isolation.

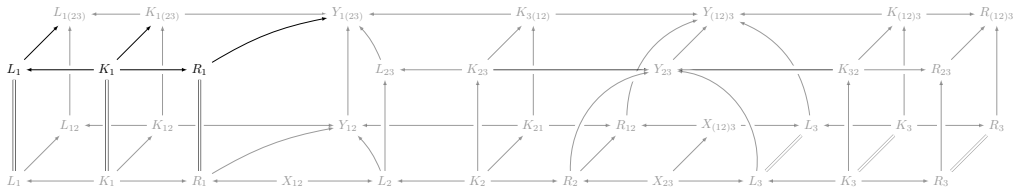


The rear face is a pushout, and therefore also a pullback. The bottom face is trivially both a pushout and a pullback. Pasting these two pushouts together yields a pushout, and since the top face – by construction – is a pullback, the front face is a pushout by Lemma 2.4: hence all faces of the cube, apart from the left and the right, are both pushouts and pullbacks.

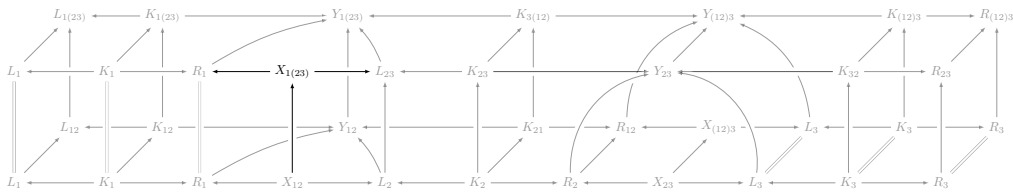
We take advantage of the symmetry involved, and obtain two further pushouts as front faces in the following. Moreover, the two new upper faces are pushouts also.



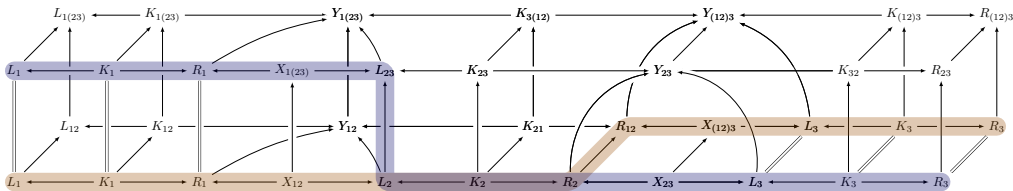
The next step is a trivial repetition of rule p_1 : the new upper faces are both pushouts since they both arise as two pushouts pasted together.



We now obtain $X_{(12)3}$ by pulling back R_1 and L_{23} along $Y_{1(23)}$, the remaining monomorphism $X_{12} \rightarrow X_{(12)3}$ follows from the universal property.



The final step consists in proving that the cospan $R_1 \rightarrow Y_{1(23)} \leftarrow L_{23}$ is the pushout of the span $R_1 \leftarrow X_{1(23)} \rightarrow L_{23}$. Since the proof requires a somewhat lengthy diagram chase, we relegate this part of the proof to Appendix A.1. To conclude, the associativity property manifests itself in the following form, whereby the data provided along the path highlighted in orange below permits to uniquely compute the data provided along the path highlighted in blue (with both sets of overlaps computing the same “triple composite” production):



4 From associativity of concurrent derivations to rule algebras

In DPO rewriting, each linear rewriting rule has a non-deterministic effect when acting on a given object, in the sense that there generically exist multiple possible choices of admissible match of the rule into the object. One interesting way of incorporating this non-determinism into a mathematical rewriting framework is motivated by the physics literature:

- Each linear rule is lifted to an element of an abstract *vector space*.
- Concurrent composition of linear rules is lifted to a *bilinear multiplication operation* on this abstract vector space, endowing it with the structure of an *algebra*.
- The action of rules on objects is implemented by mapping each linear rule (seen as an element of the abstract algebra) to an endomorphism on an abstract vector space whose basis vectors are in bijection with the objects of the adhesive category.

While this recipe might seem somewhat ad hoc, we will demonstrate in Section 5 that it recovers in fact one of the key constructions of quantum physics and enumerative combinatorics, namely we recover the well-known Heisenberg-Weyl algebra and its canonical representation.

11:8 Rule Algebras for Adhesive Categories

► **Definition 4.1.** Let $\delta : Lin(\mathbf{C}) \rightarrow \mathcal{R}_{\mathbf{C}}$ be defined as a morphism which maps each linear rule $p = (I \xrightarrow{r} O) \in Lin(\mathbf{C})$ to a basis vector $\delta(p)$ of a free \mathbb{R} -vector space $\mathcal{R}_{\mathbf{C}} \equiv (\mathcal{R}_{\mathbf{C}}, +, \cdot)$. In order to distinguish between elements of $Lin(\mathbf{C})$ and $\mathcal{R}_{\mathbf{C}}$, we introduce the notation

$$(O \xleftarrow{r} I) := \delta \left(I \xrightarrow{r} O \right). \quad (8)$$

We will later refer to $\mathcal{R}_{\mathbf{C}}$ as the \mathbb{R} -vector space of *rule algebra elements*.

► **Definition 4.2.** Define the *rule algebra product* $*_{\mathcal{R}_{\mathbf{C}}}$ as the binary operation

$$*_{\mathcal{R}_{\mathbf{C}}} : \mathcal{R}_{\mathbf{C}} \times \mathcal{R}_{\mathbf{C}} \rightarrow \mathcal{R}_{\mathbf{C}} : (R_1, R_2) \mapsto R_1 *_{\mathcal{R}_{\mathbf{C}}} R_2, \quad (9)$$

where for two basis vectors $R_i = \delta(p_i)$ encoding the linear rules $p_i \in Lin(\mathbf{C})$ ($i = 1, 2$),

$$R_1 *_{\mathcal{R}_{\mathbf{C}}} R_2 := \sum_{\mathbf{m}_{12} \in p_1 \uparrow p_2} \delta \left(p_1 \xleftarrow{\mathbf{m}_{12}} p_2 \right). \quad (10)$$

The definition is extended to arbitrary (finite) linear combinations of basis vectors by bilinearity, whence for $p_i, p_j \in Lin(\mathbf{C})$ and $\alpha_i, \beta_j \in \mathbb{R}$,

$$\left(\sum_i \alpha_i \cdot \delta(p_i) \right) *_{\mathcal{R}_{\mathbf{C}}} \left(\sum_j \beta_j \cdot \delta(p_j) \right) := \sum_{i,j} (\alpha_i \cdot \beta_j) \cdot (\delta(p_i) *_{\mathcal{R}_{\mathbf{C}}} \delta(p_j)). \quad (11)$$

We refer to $\mathcal{R}_{\mathbf{C}} \equiv (\mathcal{R}_{\mathbf{C}}, *_{\mathcal{R}_{\mathbf{C}}})$ as the *rule algebra* (of linear DPO-type rewriting rules over the adhesive category \mathbf{C}).

► **Theorem 4.3.** *For every adhesive category \mathbf{C} , the associated rule algebra $\mathcal{R}_{\mathbf{C}} \equiv (\mathcal{R}_{\mathbf{C}}, *_{\mathcal{R}_{\mathbf{C}}})$ is an associative algebra. If \mathbf{C} in addition possesses a strict initial object $c_{\emptyset} \in ob(\mathbf{C})$, $\mathcal{R}_{\mathbf{C}}$ is in addition a unital algebra, with unit element $R_{\emptyset} := (c_{\emptyset} \xleftarrow{\emptyset} c_{\emptyset})$.*

Proof. Associativity follows immediately from the associativity of the operation $\cdot \blacktriangleleft$, proved in Theorem 3.3. The claim that R_{\emptyset} is the unit element of the rule algebra $\mathcal{R}_{\mathbf{C}}$ of an adhesive category \mathbf{C} with strict initial object follows directly from the definition of the rule algebra product for $R_{\emptyset} *_{\mathcal{R}_{\mathbf{C}}} R$ and $R *_{\mathcal{R}_{\mathbf{C}}} R_{\emptyset}$ for $R \in \mathcal{R}_{\mathbf{C}}$. For clarity, we present below the category-theoretic composition calculation that underlies the equation $R_{\emptyset} *_{\mathcal{R}_{\mathbf{C}}} R = R$:

The property of a rule algebra being unital and associative has the important consequence that one can provide *representations* for it. The following definition, given at the level of adhesive categories with strict initial objects, captures several of the concrete notions of canonical representations in the physics literature; in particular, it generalizes the concept of canonical representation of the Heisenberg-Weyl algebra as explained in Section 5.

► **Definition 4.4.** Let \mathbf{C} be an adhesive category with a strict initial object $c_\emptyset \in \text{ob}(\mathbf{C})$, and let $\mathcal{R}_{\mathbf{C}}$ be its associated rule algebra of DPO type. Denote by $\hat{\mathbf{C}}$ the \mathbb{R} -vector space of objects of \mathbf{C} , whence (with $|C\rangle$ denoting the basis vector of $\hat{\mathbf{C}}$ associated to an element $C \in \text{ob}(\mathbf{C})$)

$$\hat{\mathbf{C}} := \text{span}_{\mathbb{R}}(\{|C\rangle \mid C \in \text{ob}(\mathbf{C})\}) \equiv (\hat{\mathbf{C}}, +, \cdot). \quad (13)$$

Then the *canonical representation* $\rho_{\mathbf{C}}$ of $\mathcal{R}_{\mathbf{C}}$ is defined as the algebra homomorphism $\rho_{\mathbf{C}} : \mathcal{R}_{\mathbf{C}} \rightarrow \text{End}(\hat{\mathbf{C}})$, with

$$\rho_{\mathbf{C}}(p) |C\rangle := \begin{cases} \sum_{m \in \mathcal{M}_p(C)} |p_m(C)\rangle & \text{if } \mathcal{M}_p(C) \neq \emptyset \\ 0_{\hat{\mathbf{C}}} & \text{otherwise,} \end{cases} \quad (14)$$

extended to arbitrary elements of $\mathcal{R}_{\mathbf{C}}$ and of $\hat{\mathbf{C}}$ by linearity.

The fact that $\rho_{\mathbf{C}}$ as given in Definition 4.4 is a homomorphism is shown below.

► **Theorem 4.5** (Canonical Representation). *For \mathbf{C} adhesive with strict initial object, $\rho_{\mathbf{C}} : \mathcal{R}_{\mathbf{C}} \rightarrow \text{End}(\hat{\mathbf{C}})$ of Definition 4.4 is a homomorphism of unital associative algebras.*

Proof. See Appendix A.2. ◀

5 Recovering the blueprint: the Heisenberg-Weyl algebra

As a first consistency check and interesting special (and arguably simplest) case of rule algebras, consider the adhesive category \mathbb{F} of equivalence classes of finite sets, and functions. This category might alternatively be interpreted as the category of isomorphism classes of *discrete graphs*, whose monomorphisms are precisely the injective partial morphisms of discrete graphs. Specializing to a subclass of morphisms, namely to *trivial* monomorphisms,

$$I \xrightarrow{\emptyset} O \equiv (I \leftarrow \emptyset \rightarrow O),$$

we recover the famous Heisenberg-Weyl algebra and its canonical representation:

► **Definition 5.1.** Let \mathcal{R}_0 denote the rule algebra of DPO type rewriting for discrete graphs. Then the subalgebra \mathcal{H} of \mathcal{R}_0 is defined as the algebra whose elementary generators are

$$x^\dagger := (\bullet \xleftarrow{\emptyset} \emptyset), \quad x := (\emptyset \xleftarrow{\emptyset} \bullet), \quad (15)$$

and whose elements are (finite) linear combinations of words in x^\dagger and x (with concatenation given by the rule algebra multiplication $*_{\mathcal{R}_0}$) and of the unit element $R_\emptyset = (\emptyset \xleftarrow{\emptyset} \emptyset)$. The canonical representation of \mathcal{H} is the restriction of the canonical representation of \mathcal{R}_0 to \mathcal{H} .

The following theorem demonstrates how well-known properties of the Heisenberg-Weyl algebra (see e.g. [7, 4, 5] and references therein) follow directly from the previously introduced constructions of the rule algebra and its canonical representation. This justifies our claim that the Heisenberg-Weyl construction is a special case of our general framework.

► **Theorem 5.2** (Heisenberg-Weyl algebra from discrete graph rewriting rule algebra).

(i) For integers $m, n > 0$,

$$\underbrace{x^\dagger *_{\mathcal{R}_0} \dots *_{\mathcal{R}_0} x^\dagger}_{m \text{ times}} = \underbrace{x^\dagger \uplus \dots \uplus x^\dagger}_{m \text{ times}}, \quad \underbrace{x *_{\mathcal{R}_0} \dots *_{\mathcal{R}_0} x}_{n \text{ times}} = \underbrace{x \uplus \dots \uplus x}_{n \text{ times}}, \quad (16)$$

where we define for linear rules $p_1, p_2 \in \text{Lin}(\mathbf{C})$

$$\delta(p_1) \uplus \delta(p_2) := \delta(p_1 \xleftarrow{\emptyset} p_2). \quad (17)$$

11:10 Rule Algebras for Adhesive Categories

(ii) The generators $x, x^\dagger \in \mathcal{H}$ fulfill the canonical commutation relation

$$[x, x^\dagger] \equiv x *_{\mathcal{R}_0} x^\dagger - x^\dagger *_{\mathcal{R}_0} x = R_\emptyset, \quad R_\emptyset = (\emptyset \xleftarrow{\emptyset} \emptyset). \quad (18)$$

(iii) Every element of \mathcal{H} may be expressed as a (finite) linear combination of so-called normal-ordered expressions $x^\dagger *_{\mathcal{R}_0} x *_{\mathcal{R}_0} x^\dagger *_{\mathcal{R}_0} x$ (with $r, s \in \mathbb{Z}_{\geq 0}$).

(iv) Denoting by $|n\rangle \equiv |\bullet^{\uplus n}\rangle$ ($n \in \mathbb{Z}_{\geq 0}$) the basis vector associated to the discrete graph with n vertices in the vector space \hat{G}_0 of isomorphism classes discrete graphs, the canonical representation of \mathcal{H} according to Definition 4.4 reads explicitly

$$a^\dagger |n\rangle = |n+1\rangle, \quad a |n\rangle = \begin{cases} n \cdot |n-1\rangle & \text{if } n > 0 \\ 0_{\hat{G}_0} & \text{else} \end{cases}, \quad (19)$$

with $a^\dagger := \rho_{\mathcal{R}_0}(x^\dagger)$ (the creation operator) and $a := \rho_{\mathcal{R}_0}(x)$ (the annihilation operator).

Proof. See Appendix A.3. ◀

6 Applications of rule algebras to combinatorics

In this section we consider an example application, working with undirected multigraphs.

Given a set X , let $\mathcal{P}_2 X$ be the set of subsets of X of cardinality 2. Note that, unlike the ordinary powerset construction, \mathcal{P}_2 fails to be a covariant functor on the category of sets, since it is undefined on non-injective functions. An *undirected multigraph* is a triple $\mathcal{U} = (V, E, t : E \rightarrow \mathcal{P}_2 V)$ where V is a set of vertices, E a set of edges, and t assigns two distinct vertices to each edge. A homomorphism $f : \mathcal{U} \rightarrow \mathcal{U}'$ of undirected multigraphs consists of two functions, $f_E : E \rightarrow E'$ and $f_V : V \rightarrow V'$, such that f_V is

- *non-edge collapsing*, i.e. for all $e \in E$ with $t(e) = \{v, v'\}$, we have $f_V(v) \neq f_V(v')$, and
- *edge preserving*, i.e. for all $e \in E$ with $t(e) = \{v, v'\}$, we have $t' f_E(e) = \{f_V(v), f_V(v')\}$.

Let **uGraph** the the category of undirected multigraphs and their morphisms. It is easy to see that the empty multigraph ($V = E = \emptyset$) is a strict initial object. Moreover, it is not difficult to show that pullbacks and pushouts exist and are calculated point-wise for vertices and edges in the category of sets. It follows that **uGraph** is adhesive for similar reasons to why the usual category of directed multigraphs – which is a presheaf category – is adhesive.

For convenience, we adopt a notation in which we consider a rule algebra basis element $(O \xleftarrow{f} I) \in \mathcal{R}_{\mathbf{uGraph}}$ as the graph of its induced injective partial morphism $(I \xrightarrow{f} O) \in \text{Inj}(I, O)$ of graphs I and O , with the input graph I drawn at the bottom, O at the top, where the structure of the morphism f is indicated with dotted lines. See the example below:

► **Definition 6.1.** We define the algebra \mathcal{A} as the one generated² by the rule algebra elements

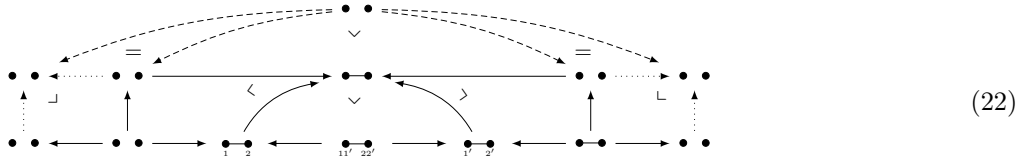
$$e_+ := \frac{1}{2} \cdot \left(\begin{array}{c} \bullet \text{---} \bullet \\ \vdots \quad \vdots \\ \bullet \text{---} \bullet \end{array} \right), \quad e_- := \frac{1}{2} \cdot \left(\begin{array}{c} \bullet \quad \bullet \\ \vdots \quad \vdots \\ \bullet \text{---} \bullet \end{array} \right), \quad d := \frac{1}{2} \cdot \left(\begin{array}{c} \bullet \quad \bullet \\ \vdots \quad \vdots \\ \bullet \quad \bullet \end{array} \right). \quad (20)$$

The algebra thus defined may be characterized via its *commutation relations*, which read (with $[x, y] := x *_{\mathcal{R}} y - y *_{\mathcal{R}} x$ for $\mathcal{R} \equiv \mathcal{R}_{\mathbf{uGraph}}$)

$$[e_-, e_+] = d, \quad [e_+, d] = [e_-, d] = 0. \quad (21)$$

² As in the case of the Heisenberg-Weyl algebra, by “generated” we understand that a generic element of \mathcal{A} is a finite linear combination of (finite) words in the generators and of the identity element R_\emptyset , with concatenation given by the rule algebra composition.

Here, the only nontrivial contribution (i.e. the one that renders the first commutator non-zero) may be computed from the DPO-type composition diagram³ below (compare (5) and (6)) and its variant for the admissible match $\bullet_1 \bullet_2 \leftarrow \bullet_{12'} \bullet_{21'} \rightarrow \bullet_{1'} \bullet_{2'}$:



We find an interesting structure for the representation of \mathcal{A} :

► **Lemma 6.2.** *Let $E_{\pm} := \rho(e_{\pm})$ and $D := \rho(d)$, and for an arbitrary basis vector $|G\rangle \in \hat{\mathbb{G}}$ (with \mathbb{G} denoting the set of isomorphism classes of finite undirected multigraphs), we find that the linear endomorphisms $\rho(X)$ for $X \in \{E_+, E_-, D\}$ admit a decomposition into invariant subspaces $\hat{\mathbb{G}}_n$, with $n \in \mathbb{Z}_{\geq 0}$ denoting the number of vertices of the graphs in a given subspace:*

$$\rho(X) = \bigoplus_{n \geq 0} (\rho(X))|_{\hat{\mathbb{G}}_n}. \tag{23}$$

Proof. The three rules that define the algebra \mathcal{A} do not modify the number of vertices when applied to a given graph (via the canonical representation). ◀

One may easily verify that the operator $D = \rho(d)$ may be equivalently expressed as

$$D = \frac{1}{2} \cdot \rho \left(\begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \right) = \frac{1}{2} (O_{\bullet} O_{\bullet} - O_{\bullet}), \quad O_{\bullet} := \rho \left(\begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \right). \tag{24}$$

Since the diagonal operator O_{\bullet} when applied to an arbitrary graph state $|G\rangle$ for $G \in \mathbb{G}$ effectively counts the number $n_V(G)$ of vertices of G ,

$$O_{\bullet} |G\rangle = n_V(G) |G\rangle, \tag{25}$$

one finds that

$$D |G\rangle = \frac{1}{2} O_{\bullet} (O_{\bullet} - 1) |G\rangle = \frac{1}{2} n_V(G) (n_V(G) - 1) |G\rangle. \tag{26}$$

One may thus alternatively analyze the canonical representation of \mathcal{A} split into invariant subspaces of D . The lowest non-trivial such subspace is the space $\hat{\mathbb{G}}_2$ of undirected multigraphs on two vertices. It in fact furnishes a representation of the Heisenberg-Weyl algebra, with E_+ and E_- taking the roles of the creation and of the annihilation operator, respectively, and with the number vectors $|n\rangle \equiv |\bullet^{\uplus n}\rangle$ implemented as follows (with $(m)_n := \Theta(m-n)m!/(m-n)!$):

$$E_+^n |\bullet \bullet\rangle = \left| \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \right\rangle, \quad E_- \left| \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \right\rangle = (n)_1 \left| \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} \right\rangle. \tag{27}$$

³ Note that the number indices are used solely to specify the precise structure of the match, and are not to be understood as actual vertex labels or types.

11:12 Rule Algebras for Adhesive Categories

But already the invariant subspace based on the initial vector $|\bullet \bullet \bullet\rangle \in \hat{\mathbb{C}}_3$ has a very interesting combinatorial structure:

$$\begin{aligned}
 E_+ |\bullet \bullet \bullet\rangle &= 3 |\bullet \dashrightarrow \bullet\rangle \equiv 3 |\{1, 0, 0\}\rangle \\
 E_+^2 |\bullet \bullet \bullet\rangle &= 3 (|\bullet \leftrightarrow \bullet\rangle + 2 |\bullet \dashrightarrow \bullet\rangle) \equiv 3 (|\{2, 0, 0\}\rangle + 2 |\{1, 1, 0\}\rangle) \\
 E_+^3 |\bullet \bullet \bullet\rangle &= 3 (|\bullet \leftrightarrow \bullet\rangle + 6 |\bullet \dashrightarrow \bullet\rangle + 2 |\bullet \dashrightarrow \bullet\rangle) \\
 &\equiv 3 (|\{3, 0, 0\}\rangle + 6 |\{2, 1, 0\}\rangle + 2 |\{1, 1, 1\}\rangle) \\
 &\vdots \\
 E_+^n |\bullet \bullet \bullet\rangle &\equiv E_+^n |\{0, 0, 0\}\rangle = 3 \sum_{k=0}^n T(n, k) |S(n, k)\rangle
 \end{aligned} \tag{28}$$

Here, the state $|\{f, g, h\}\rangle$ with $f \geq g \geq h \geq 0$ and $f + g + h = n$ is the graph state on three vertices with (in one of the possible presentations of the isomorphism class) f edges between the first two, g edges between the second two and h edges between the third and the first vertex. Furthermore, $T(n, k)$ and $S(n, k)$ are given by the entry *A286030* of the OEIS database [1]. The interpretation of $S(n, k)$ and $T(n, k)$ is that each triple $S(n, k)$ encodes the outcome of a game of three players, counting (without regarding the order of players) the number of wins per player for a total of n games. Then $T(n, k)/3^{(n-1)}$ gives the probability that a particular pattern $S(n, k)$ occurs in a random sample.

It thus appears to be an interesting avenue of future research to investigate the apparently quite intricate interrelations between representation theory and combinatorics.

7 Applications of rule algebras to stochastic mechanics

One of the main motivations that underpinned the development of the rule algebra framework prior to this paper [2, 4] has been the link between associative unital algebras of transitions and continuous-time Markov chains (CTMCs). Famous examples of such particular types of CTMCs include chemical reaction systems (see e.g. [5] for a recent review) and stochastic graph rewriting systems (see [2] for a rule-algebraic implementation). With our novel formulation of unital associative rule algebras and their canonical representation for generic strict initial adhesive categories, it is possible to specify a *general stochastic mechanics framework*. While we postpone a detailed presentation of this result to future work, suffice it here to define the basic framework and to indicate the potential of the idea with a short worked example. We begin by specializing the general definition of continuous-time Markov chains (see e.g. [13]) to the setting of rewriting systems (compare [2, 5]):

► **Definition 7.1.** Consider an adhesive category \mathbf{C} with strict initial object $o_\emptyset \in \text{ob}(\mathbf{C})$, and let $\hat{\mathbf{C}}$ denote the free \mathbb{R} -vector space of objects of \mathbf{C} according to Definition 4.4. Then we define the space $\text{Prob}(\mathbf{C})$ as the *space of sub-probability distributions* in the following sense:

$$\text{Prob}(\mathbf{C}) := \left\{ |\Psi\rangle = \sum_{o \in \text{ob}(\mathbf{C})} \psi_o |o\rangle \mid \forall o \in \text{ob}(\mathbf{C}) : \psi_o \in \mathbb{R}_{\geq 0} \wedge \sum_{o \in \text{ob}(\mathbf{C})} \psi_o \leq 1 \right\}. \tag{29}$$

In particular, this identifies the sequences $\{\psi_o\}_{o \in \text{ob}(\mathbf{C})} \in \ell_{\mathbb{R}}^1(\text{ob}(\mathbf{C}))$ as special types of $\ell_{\mathbb{R}}^1$ -summable sequences indexed by objects of \mathbf{C} . Let $\text{Stoch}(\mathbf{C}) := \text{End}(\text{Prob}(\mathbf{C}))$ be the space of *sub-stochastic operators*. Then a **continuous-time Markov chain (CTMC)** is specified in terms of a tuple of data $(|\Psi(0)\rangle, H)$, where $|\Psi(0)\rangle \in \text{Prob}(\mathbf{C})$ is the *initial state*, and

where $H \in \text{End}_{\mathbb{R}}(\mathcal{S}_{\mathbf{C}})$ is the *infinitesimal generator* or *Hamiltonian* of the CTMC (with $\mathcal{S}_{\mathbf{C}}$ the Fréchet space of real-valued sequences $f \equiv (f_o)_{o \in \text{ob}(\mathbf{C})}$ with semi-norms $\|f\|_o := |f_o|$). H is required to be an infinitesimal (sub-)stochastic operator, whence to fulfill the constraints

$$\begin{aligned} H &\equiv (h_{o,o'})_{o,o' \in \text{ob}(\mathbf{C})} \quad \forall o, o' \in \text{ob}(\mathbf{C}) : \\ &\quad (i) \ h_{o,o} \leq 0, \quad (ii) \ \forall o \neq o' : h_{o,o'} \geq 0, \quad (iii) \ \sum_{o'} h_{o,o'} = 0. \end{aligned} \quad (30)$$

Then this data encodes the *evolution semi-group* $\mathcal{E} : \mathbb{R}_{\geq 0} \rightarrow \text{Stoch}(\mathbf{C})$ as the (point-wise minimal non-negative) solution of the *Kolmogorov backwards* or *master equation*:

$$\frac{d}{dt} \mathcal{E}(t) = H \mathcal{E}(t), \quad \mathcal{E}(0) = \mathbb{1}_{\text{End}_{\mathbb{R}}(\mathcal{S}_{\mathbf{C}})} \Rightarrow \quad \forall t, t' \in \mathbb{R}_{\geq 0} : \mathcal{E}(t) \mathcal{E}(t') = \mathcal{E}(t + t'). \quad (31)$$

Consequently, the *time-dependent state* $|\Psi(t)\rangle$ of the system is given by

$$\forall t \in \mathbb{R}_{\geq 0} : \quad |\Psi(t)\rangle = \mathcal{E}(t) |\Psi(0)\rangle. \quad (32)$$

Typically, our interest in analyzing a given CTMC will consist in studying the dynamical statistical behavior of so-called observables:

► **Definition 7.2.** Let $\mathcal{O}_{\mathbf{C}} \subset \text{End}_{\mathbb{R}}(\mathcal{S}_{\mathbf{C}})$ denote the space of *observables*, defined as the space of *diagonal operators*,

$$\mathcal{O}_{\mathbf{C}} := \{O \in \text{End}_{\mathbb{R}}(\mathcal{S}_{\mathbf{C}}) \mid \forall o \in \text{ob}(\mathbf{C}) : O|o\rangle = \omega_O(o)|o\rangle, \ \omega_O(o) \in \mathbb{R}\}. \quad (33)$$

We furthermore define the so-called *projection operation* $\langle | : \mathcal{S}_{\mathbf{C}} \rightarrow \mathbb{R}$ via extending by linearity the definition of $\langle |$ acting on basis vectors of $\hat{\mathbf{C}}$,

$$\forall o \in \text{ob}(\mathbf{C}) : \quad \langle |o\rangle := 1_{\mathbb{R}}. \quad (34)$$

These definitions induce a notion of *correlators* of observables, defined for $O_1, \dots, O_n \in \mathcal{O}_{\mathbf{C}}$ and $|\Psi\rangle \in \text{Prob}(\mathbf{C})$ as

$$\langle O_1, \dots, O_n \rangle_{|\Psi\rangle} := \langle |O_1, \dots, O_n | \Psi\rangle = \sum_{o \in \text{ob}(\mathbf{C})} \psi_o \cdot \omega_{O_1}(o) \cdots \omega_{O_n}(o). \quad (35)$$

The precise relationship between the notions of CTMCs and DPO rewriting rules as encoded in the rule algebra formalism is established in the form of the following theorem (compare [2]):

► **Theorem 7.3 (Stochastic mechanics framework).** *Let \mathbf{C} be an adhesive category with strict initial object, let $\{(O_j \xrightarrow{r_j} I_j) \in \mathcal{R}_{\mathbf{C}}\}_{j \in \mathcal{J}}$ be a (finite) set of rule algebra elements and $\{\kappa_j \in \mathbb{R}_{\geq 0}\}_{j \in \mathcal{J}}$ a collection of non-zero parameters (called base rates). Then one may construct a Hamiltonian H from this data according to*

$$H := \hat{H} + \bar{H}, \quad \hat{H} := \sum_{j \in \mathcal{J}} \kappa_j \cdot \rho \left(O_j \xrightarrow{r_j} I_j \right), \quad \bar{H} := - \sum_{j \in \mathcal{J}} \kappa_j \cdot \rho \left(I_j \xrightarrow{\text{id}_{\text{dom}(r_j)}} I_j \right). \quad (36)$$

Here, for arbitrary $(I \xrightarrow{r} O) \equiv (I \xleftarrow{i} K \xrightarrow{o} O) \in \text{Lin}(\mathbf{C})$, we define

$$(I \xrightarrow{\text{id}_{\text{dom}(r)}} I) := (I \xleftarrow{i} K \xrightarrow{i} I). \quad (37)$$

The observables for the resulting CTMC are operators of the form

$$O_M^t = \rho \left(M \xleftarrow{t} M \right). \quad (38)$$

We furthermore have the jump-closure property, whereby for all $(O \xleftarrow{r} I) \in \mathcal{R}_{\mathbf{C}}$

$$\langle | \rho(O \xleftarrow{r} I) = \langle | O_I^{\text{id}_{\text{dom}(r)}}. \quad (39)$$

11:14 Rule Algebras for Adhesive Categories

Proof. See Appendix A.4. ◀

We illustrate the framework with an example for $\mathbf{C} = \mathbf{uGraph}$ (the category of (isomorphism classes of) undirected multigraphs and morphisms thereof), where we pick the two rule algebra elements e_+ and e_- specified in (20) to define the transitions of the system. Together with two non-negative real parameters $\kappa_+, \kappa_- \in \mathbb{R}_{\geq 0}$, the resulting Hamiltonian $H = \hat{H} + \bar{H}$ reads (with $E_{\pm} := \rho(e_{\pm})$ and O_{\bullet} as in (24))

$$\hat{H} = \kappa_+ E_+ + \kappa_- E_-, \quad \bar{H} = -\frac{1}{2}\kappa_+ O_{\bullet}(O_{\bullet} - 1) - \kappa_- O_E, \quad O_E := \frac{1}{2}\rho \left(\begin{array}{c} \bullet \quad \bullet \\ \vdots \\ \bullet \quad \bullet \end{array} \right). \quad (40)$$

Using the general fact that a Hamiltonian as constructed according to Theorem 7.3 verifies

$$\langle | H = 0, \quad (41)$$

we may for example compute the time evolution of the expectation values of observables for this CTMC. Intuitively, the CTMC describes a stochastic system where edges are added and removed at random. Since these transitions do not modify the number of vertices, we immediately conclude that if the initial state $|\Psi(0)\rangle \in \text{Prob}(\mathbf{uGraph})$ is a *pure state*, i.e. if $|\Psi(0)\rangle = |G_0\rangle$ for some $G_0 \in \text{ob}(\mathbf{uGraph})$, one finds⁴

$$\forall t \geq 0: \quad \langle | O_{\bullet} |\Psi(t)\rangle = \langle | O_{\bullet} |G_0\rangle = N_V, \quad (42)$$

with N_V the number of vertices of G_0 . Let us analogously denote by N_E the number of edges of G_0 , determined according to

$$N_E = \langle | O_E |G_0\rangle. \quad (43)$$

The time evolution of the moments of the edge-counting observable O_E may be computed by means of algebraic methods. Referring to [2, 5] for more extensive computations, suffice it here to demonstrate the derivation of the evolution of the average edge-count for $|\Psi(0)\rangle = |G_0\rangle$:

$$\begin{aligned} \frac{d}{dt} \langle | O_E |\Psi(t)\rangle &= \langle | O_E H |\Psi(t)\rangle = \langle | (H O_E + [O_E, H]) |\Psi(t)\rangle \\ &\stackrel{(41)}{=} \kappa_+ \langle | E_+ |\Psi(t)\rangle - \kappa_- \langle | E_- |\Psi(t)\rangle \\ &\stackrel{(39)}{=} \frac{1}{2}\kappa_+ \langle | O_{\bullet}(O_{\bullet} - 1) |\Psi(t)\rangle - \kappa_- \langle | O_E |\Psi(t)\rangle \\ &\stackrel{(42)}{=} \frac{1}{2}\kappa_+ N_V(N_V - 1) - \kappa_- \langle | O_E |\Psi(t)\rangle. \end{aligned} \quad (44)$$

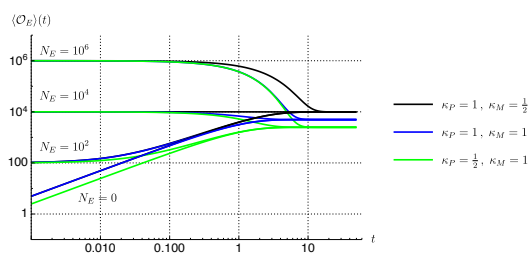
Together with the initial value $\langle | O_E |\Psi(0)\rangle = N_E$, this ODE is solved (for $\kappa_M \neq 0$ and with the convention $\binom{x}{y} := 0$ for $x < y$) by

$$\langle O_E \rangle(t) \equiv \langle | O_E |\Psi(t)\rangle = e^{-t\kappa_M} \left(N_E - \frac{\kappa_P}{\kappa_M} \binom{N_V}{2} \right) + \frac{\kappa_P}{\kappa_M} \binom{N_V}{2} \xrightarrow[t \rightarrow \infty]{} \frac{\kappa_P}{\kappa_M} \binom{N_V}{2}. \quad (45)$$

Interestingly, the coefficient $\binom{N_V}{2}$ is precisely the number of edges of a complete graph on N_V vertices. Moreover, if $\kappa_P = \kappa_M$ and $N_{E^*} = \binom{N_V}{2}$, $\langle O_E \rangle(t) = N_{E^*} = \text{const}$ for all $t \geq 0$.

⁴ More precisely, one may verify that $[O_{\bullet}, H] = 0$, whence the claim follows from $\langle | O_{\bullet} |\Psi(0)\rangle = N_V$ and

$$\frac{d}{dt} \langle | O_{\bullet} |\Psi(t)\rangle = \langle | O_{\bullet} H |\Psi(t)\rangle = \langle | (H O_{\bullet} + [O_{\bullet}, H]) |\Psi(t)\rangle = 0.$$



■ **Figure 1** Time-evolution of $\langle O_E \rangle(t)$ for $|\Psi(0)\rangle = |G_0\rangle$ with $N_V = 100$.

We present in Figure 1 the time-evolution of $\langle O_E \rangle(t)$ for three different choices of parameters κ_+ and κ_- , and for four different choices each of initial number of edges N_E .

As an outlook and reference to ongoing and future work, techniques such as the ones developed in [2] and [3] in favorable cases even permit to derive the full time-dependent probability distribution of observables – in fact, in the present example, one may demonstrate that the distribution of the edge-counting observable O_E stabilizes for $t \rightarrow \infty$ onto a Poisson distribution of parameter $\frac{\kappa_P}{\kappa_M} \binom{N_V}{2}$. This result might be somewhat anticipated, in that for the special case $N_V = 2$ we found in the previous section that E_+ and E_- acting on the states with two vertices effectively yield a representation of the Heisenberg-Weyl algebra, whence in this case the process reduces to a birth-death process on edges with rates κ_+ and κ_- (see [5] for further details on chemical reaction systems).

8 Conclusion and Outlook

Based on our novel theorem on the associativity of the operation of forming DPO-type concurrent compositions of linear rewriting rules, we introduced the concept of *rule algebras*: each linear rule is mapped to an element of an abstract vector space of linear rules, on which the concurrent composition operation is implemented as a binary, bilinear multiplication operation. For every adhesive category \mathbf{C} , the associated rule algebra is associative, and if the category possesses a strict initial object (i.e. if \mathbf{C} is an extensive category), this algebra is in addition unital. We hinted at the potential of our approach in the realm of combinatorics, and, as a first major application of our framework, we presented a *universal construction of continuous-time Markov chains* based on linear rules of extensive categories \mathbf{C} . It appears reasonable in light of the deep insights gained into such CTMC theories for the special cases of discrete rewriting rules [5] and multigraph rewriting rules [3, 2] to expect that our approach will lead to progress in the understanding and analysis of stochastic rewriting systems in both theory and practice.

References

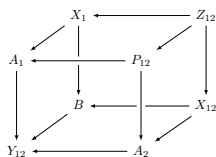
- 1 OEIS Foundation Inc. (2018), The On-Line Encyclopedia of Integer Sequences, <https://oeis.org/A286030>.
- 2 Nicolas Behr, Vincent Danos, and Ilias Garnier. Stochastic mechanics of graph rewriting. *Proceedings of the 31st Annual ACM-IEEE Symposium on Logic in Computer Science (LICS 2016)*, pages 46–55, 2016.
- 3 Nicolas Behr, Vincent Danos, and Ilias Garnier. Combinatorial Conversion and Moment Bisimulation for Stochastic Rewriting Systems (in preparation), 2018.

- 4 Nicolas Behr, Vincent Danos, Ilias Garnier, and Tobias Heindel. The algebras of graph rewriting. *arXiv:1612.06240*, 2016.
- 5 Nicolas Behr, Gerard HE Duchamp, and Karol A Penson. Combinatorics of Chemical Reaction Systems. *arXiv:1712.06575*, 2017.
- 6 Pawel Blasiak, Gerard HE Duchamp, Allan I Solomon, Andrzej Horzela, Karol A Penson, et al. Combinatorial Algebra for second-quantized Quantum Theory. *Advances in Theoretical and Mathematical Physics*, 14(4):1209–1243, 2010.
- 7 Pawel Blasiak and Philippe Flajolet. Combinatorial Models of Creation-Annihilation. *Séminaire Lotharingien de Combinatoire*, 65(B65c):1–78, 2011.
- 8 Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. Algebraic Approaches to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach. In *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 163–246, 1997.
- 9 Hartmut Ehrig and Hans-Jörg Kreowski. Parallelism of manipulations in multidimensional information structures. In *Mathematical Foundations of Computer Science*, number 45 in LNCS, pages 284–293. Springer, 1976.
- 10 Richard Garner and Stephen Lack. On the axioms for adhesive and quasiadhesive categories. *Theor. App. Categories*, 27(3):27–46, 2012.
- 11 Stephen Lack and Paweł Sobociński. Adhesive and quasiadhesive categories. *RAIRO-Theoretical Informatics and Applications*, 39(3):511–545, 2005.
- 12 Stephen Lack and Paweł Sobociński. Toposes are adhesive. In *Graph Transformations, Third International Conference, (ICGT 2006)*, volume 4178 of LNCS, pages 184–198. Springer, 2006.
- 13 James R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998.
- 14 Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

A Proofs

A.1 Proof of associativity of rule compositions

► **Lemma A.1.** *Let \mathbf{C} be an adhesive category, and consider the following commutative diagram, in which all arrows are monomorphisms, and where*



- the bottom and left faces are pushout squares, and
- the front and back faces are pullback squares.

Then the right and top faces are pushout squares.

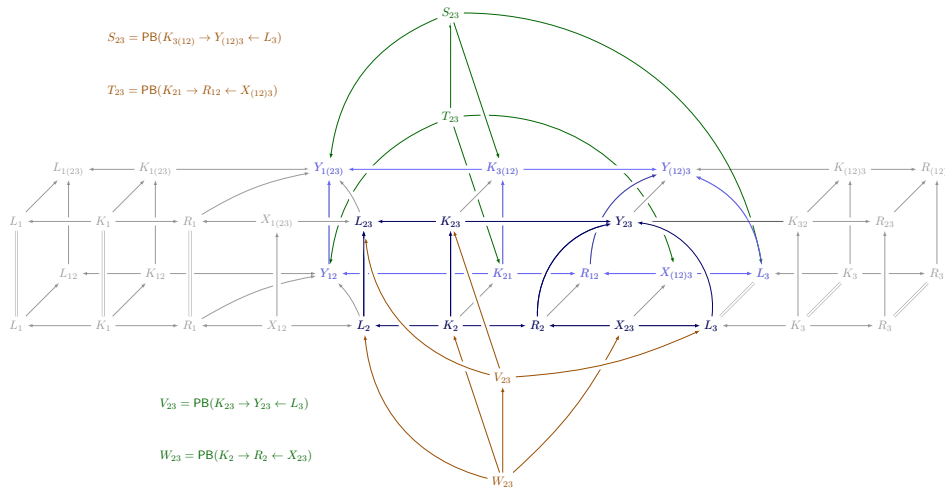
Proof. Composition of the back square and the bottom square yields a pullback square, whence according to Lemma 2.4 the top face is also a pullback square. Since thus all faces but the right one are pullbacks and the left face is a pushout square due to the VK property of \mathbf{C} . Analogously, since the bottom square is a pushout square and all vertical faces are pullback squares, the top face is a pushout square. ◀

► **Theorem 3.3 (Associativity Theorem).** *The composition operation \cdot is associative in the following sense: given linear rules $p_1, p_2, p_3 \in \text{Lin}(\mathbf{C})$, there exists a bijective correspondence*

between pairs of admissible matches $m_{21} \in p_2 \Vdash p_1$ and $m_{3(21)} \in p_3 \Vdash (p_2 \overset{m_{12}}{\triangleleft} p_1)$, and pairs of admissible matches $m_{32} \in p_3 \Vdash p_2$ and $m_{(32)1} \in (p_3 \overset{m_{23}}{\triangleleft} p_2) \Vdash p_1$ such that

$$p_3 \overset{m_{3(21)}}{\triangleleft} (p_2 \overset{m_{21}}{\triangleleft} p_1) = (p_3 \overset{m_{32}}{\triangleleft} p_2) \overset{m_{(32)1}}{\triangleleft} p_1. \quad (7)$$

Proof. We refer the readers to the main text for the first part of the proof. To prove the final part, whence that the $Y_{1(23)}$ is the pushout of $R_1 \leftarrow X_{1(23)} \rightarrow L_{23}$, we construct the following extended diagram (with S_{23} , T_{23} , V_{23} and W_{23} obtained by taking the indicated pullbacks $\text{PB}(\dots)$, and where the remaining new morphisms are formed as those that make the respective triangles involving the aforementioned objects commute):



Invoking Lemma A.1 twice, we may conclude that the squares $\square_{W_{23}, V_{23}, K_{23}, K_2}$, $\square_{W_{23}, V_{23}, L_3, X_{23}}$, $\square_{T_{23}, S_{23}, K_{3(12)}, K_{21}}$ and $\square_{T_{23}, S_{23}, L_3, X_{(12)3}}$ are pushout squares. In addition, since the squares $\square_{W_{23}, V_{23}, L_{23}, L_2}$ and $\square_{T_{23}, S_{23}, Y_{1(23)}, Y_{12}}$ are compositions of pushout squares, according to Lemma 2.4 they are pushout squares themselves. In order to prove the claim, we have to demonstrate that the cospan $R_1 \rightarrow Y_{1(23)} \leftarrow L_{23}$ are *jointly epimorphic*. Since Y_{12} is the pushout of $R_1 \leftarrow X_{12} \rightarrow L_2$, and since Y_{12} is included in $Y_{1(23)}$ (as encoded in the arrow $Y_{12} \rightarrow Y_{1(23)}$), the proof reduces to proving that the monomorphism $L_{23} \rightarrow Y_{1(23)}$ covers $Y_{1(23)} \setminus Y_{12}$. The proof is facilitated by taking advantage of the notion of *algebra of subobjects* available in every adhesive category (see [11] for the details). Note first that according to the structure of the auxiliary diagram constructed above, $Y_{1(23)} = Y_{12} \cup_{T_{23}} S_{23}$, while S_{23} in turn is the pushout complement of $T_{23} \rightarrow X_{(12)3} \rightarrow L_3$, whence $S_{23} = L_3 \setminus (X_{(12)3} \setminus T_{23})$. Analogously, $L_{23} = L_2 \cup_{W_{23}} V_{23}$, where V_{23} is the pushout complement of $W_{23} \rightarrow X_{23} \rightarrow L_3$, whence $V_{23} = L_3 \setminus (X_{23} \setminus W_{23})$. In addition, since $L_{23} \rightarrow Y_{1(23)}$, $L_2 \rightarrow L_{23}$ and $W_{23} \rightarrow L_2$, we conclude that $W_{23} \rightarrow T_{23}$. But since the monomorphism $X_{23} \rightarrow X_{(12)3}$ encodes that X_{23} is a subobject of $X_{(12)3}$, combining all arguments reveals that the portion of L_3 in $Y_{1(23)}$ not already covered by Y_{12} is always strictly smaller than the portion of L_3 in L_{23} not already covered by L_2 , whence the claim that $R_1 \rightarrow Y_{1(23)} \leftarrow L_{23}$ is jointly epimorphic follows. In summary, we have proved that each triple of linear rules and choice of admissible overlaps $(X_{12}, X_{(12)3})$ induces an overlap pair $(X_{23}, X_{1(23)})$ as given in the construction, which concludes the proof of associativity. \blacktriangleleft

A.2 Proof of the homomorphism property of the canonical representations

► **Theorem 4.5** (Canonical Representation). *For \mathbf{C} adhesive with strict initial object, $\rho_{\mathbf{C}} : \mathcal{R}_{\mathbf{C}} \rightarrow \text{End}(\hat{\mathbf{C}})$ of Definition 4.4 is a homomorphism of unital associative algebras.*

Proof. In order for $\rho_{\mathbf{C}}$ to qualify as an algebra homomorphism (of unital associative algebras $\mathcal{R}_{\mathbf{C}}$ and $\text{End}(\hat{\mathbf{C}})$), we must have (with $R_{\emptyset} = \delta(r_{\emptyset})$, $r_{\emptyset} = c_{\emptyset} \xrightarrow{\emptyset} c_{\emptyset}$)

$$(i) \rho_{\mathbf{C}}(R_{\emptyset}) = \mathbb{1}_{\text{End}(\hat{\mathbf{C}})} \quad \text{and} \quad (ii) \forall R_1, R_2 \in \mathcal{R}_{\mathbf{C}} : \rho_{\mathbf{C}}(R_1 *_{\mathcal{R}_{\mathbf{C}}} R_2) = \rho_{\mathbf{C}}(R_1) \rho_{\mathbf{C}}(R_2).$$

Due to linearity, it suffices to prove the two properties on basis elements $\delta(p), \delta(q)$ of $\mathcal{R}_{\mathbf{C}}$ and on basis elements $|C\rangle$ of $\hat{\mathbf{C}}$. Property (i) follows directly from the definition,

$$\forall C \in \text{ob}(\mathbf{C}) : \rho_{\mathbf{C}}(R_{\emptyset}) |C\rangle \stackrel{(14)}{=} \sum_{m \in \mathcal{M}_{r_{\emptyset}}(C)} |(r_{\emptyset})_m(C)\rangle = |C\rangle.$$

Property (ii) follows from Theorem 3.1 (the concurrency theorem): for all basis elements $\delta(p), \delta(q) \in \mathcal{R}_{\mathbf{C}}$ (with $p, q \in \text{Lin}(\mathbf{C})$) and for all $C \in \text{ob}(\mathbf{C})$,

$$\begin{aligned} \rho_{\mathbf{C}}(\delta(q) *_{\mathbf{C}} \delta(p)) |C\rangle &\stackrel{(10)}{=} \sum_{\mathbf{d} \in q \uparrow p} \rho_{\mathbf{C}} \left(\delta \left(q \xleftarrow{\mathbf{d}} p \right) \right) |C\rangle \\ &\stackrel{(14)}{=} \sum_{\mathbf{d} \in q \uparrow p} \sum_{e \in \mathcal{M}_{r_{\mathbf{d}}}(C)} |(r_{\mathbf{d}})_e(C)\rangle \quad (r_{\mathbf{d}} = q \xleftarrow{\mathbf{d}} p) \\ &= \sum_{m \in \mathcal{M}_p(C)} \sum_{n \in \mathcal{M}_q(p_m(C))} |q_n(p_m(C))\rangle \quad (\text{via Thm. 3.1}) \\ &\stackrel{(14)}{=} \sum_{m \in \mathcal{M}_p(C)} \rho_{\mathbf{C}}(\delta(q)) |p_m(C)\rangle \\ &\stackrel{(14)}{=} \rho_{\mathbf{C}}(\delta(q)) \rho_{\mathbf{C}}(\delta(p)) |C\rangle. \quad \blacktriangleleft \end{aligned}$$

A.3 Proof of the relationship between discrete graph rewriting and the Heisenberg-Weyl algebra

Proof.

- (i) Since there is no partial injection possible between the input of one copy and the output of another copy of x^{\dagger} other than the trivial match, and similarly for two copies of x , the claim follows.
- (ii) Computing the commutator $[x, x^{\dagger}] = x * x^{\dagger} - x^{\dagger} * x$ (with $* \equiv *_{\mathcal{R}_0}$) explicitly, we find that

$$x * x^{\dagger} = x \uplus x^{\dagger} + id_{\mathcal{R}_0}, \quad x^{\dagger} * x = x^{\dagger} \uplus x, \quad (46)$$

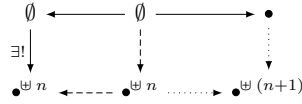
from which the claim follows due to commutativity of the operation \uplus on \mathcal{R}_0 , $x \uplus x^{\dagger} = x^{\dagger} \uplus x$.

- (iii) It suffices to prove the statement for basis elements of \mathcal{H} . Consider thus an arbitrary composition of a finite number of copies of the generators x and x^{\dagger} . Then by repeated application of the commutation relation $[x, x^{\dagger}] = id_{\mathcal{R}_0}$, and since $id_{\mathcal{R}_0}$ is the unit element for $*$ on \mathcal{R}_0 , we can convert the arbitrary basis element of \mathcal{H} into a linear combination of normal-ordered elements.

(iv) Note first that by definition $|0\rangle = |\emptyset\rangle$. To prove the claim that for all $n \geq 0$

$$a^\dagger |n\rangle = |n+1\rangle,$$

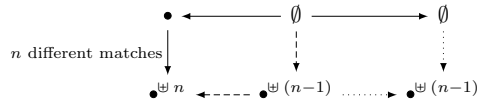
we apply Definitions 2.6 and 4.4 by computing the following diagram (compare (3)): there exists precisely one admissible match of the empty graph $\emptyset \in G_0$ into the n -vertex discrete graph $\bullet^{\uplus n}$, whence constructing the pushout complement marked with dashed arrows and the pushout marked with dotted arrows we verify the claim:



Proceeding analogously in order to prove the formula for the representation $a = \rho_{\mathcal{R}_0}(x)$,

$$a |n\rangle := \begin{cases} n \cdot |n-1\rangle & \text{if } n > 0 \\ 0_{\hat{G}_0} & \text{else,} \end{cases}$$

we find that for $n > 0$ there exist n admissible matches of the 1-vertex graph \bullet into the n -vertex graph $\bullet^{\uplus n}$, for each of which the application of the rule $\bullet \rightarrow \emptyset$ along the match results in the graph $\bullet^{\uplus (n-1)}$:



$$\Rightarrow \forall n > 0 : a |\bullet^{\uplus n}\rangle = n \cdot |\bullet^{\uplus (n-1)}\rangle$$

Finally, for $n = 0$, since by definition there exists no admissible match from the 1-vertex graph \bullet into the empty graph \emptyset , whence indeed

$$a |\emptyset\rangle = \rho_{\mathcal{R}_0} \left(\emptyset \xleftarrow{\bullet} \bullet \right) |\emptyset\rangle = 0_{\hat{G}_0}.$$

A.4 Proof of the stochastic mechanics framework theorem

► **Theorem 7.3** (Stochastic mechanics framework). *Let \mathbf{C} be an adhesive category with strict initial object, let $\{(O_j \xrightarrow{r_j} I_j) \in \mathcal{R}_{\mathbf{C}}\}_{j \in \mathcal{J}}$ be a (finite) set of rule algebra elements and $\{\kappa_j \in \mathbb{R}_{\geq 0}\}_{j \in \mathcal{J}}$ a collection of non-zero parameters (called base rates). Then one may construct a Hamiltonian H from this data according to*

$$H := \hat{H} + \bar{H}, \quad \hat{H} := \sum_{j \in \mathcal{J}} \kappa_j \cdot \rho \left(O_j \xrightarrow{r_j} I_j \right), \quad \bar{H} := - \sum_{j \in \mathcal{J}} \kappa_j \cdot \rho \left(I_j \xrightarrow{id_{dom(r_j)}} I_j \right). \quad (36)$$

Here, for arbitrary $(I \xrightarrow{r} O) \equiv (I \xleftarrow{i} K \xrightarrow{o} O) \in Lin(\mathbf{C})$, we define

$$(I \xrightarrow{id_{dom(r)}} I) := (I \xleftarrow{i} K \xrightarrow{i} I). \quad (37)$$

The observables for the resulting CTMC are operators of the form

$$O_M^t = \rho \left(M \xleftarrow{t} M \right). \quad (38)$$

We furthermore have the jump-closure property, whereby for all $(O \xrightarrow{r} I) \in \mathcal{R}_{\mathbf{C}}$

$$\langle | \rho(O \xleftarrow{r} I) = \langle | O_I^{id_{dom(r)}}. \quad (39)$$

Proof. By definition, the canonical representation of a generic rule algebra element $(O \xleftarrow{r} I) \in \mathcal{R}_{\mathbf{C}}$ is both a row- and a column-finite object, since for every object $C \in ob(\mathbf{C})$ the set of admissible matches $\mathcal{M}_p(C)$ of the associated linear rule $p \equiv (I \xrightarrow{r} O)$ is finite, and since for every object $C \in ob(\mathbf{C})$ there exists only finitely many objects $C' \in ob(\mathbf{C})$ such that $C = p_m(C')$ for some match $m \in \mathcal{M}_p(C')$. Consequently, $\rho_{\mathbf{C}}(O \xleftarrow{r} I)$ lifts consistently from a linear operator in $End(\hat{\mathbf{C}})$ to a linear operator in $End(\mathcal{S}_{\mathbf{C}})$. Let us prove next the claim on the precise structure of observables. Recall that according to Definition 7.2, an observable $O \in \mathcal{O}_{\mathbf{C}}$ must be a linear operator in $End(\mathcal{S}_{\mathbf{C}})$ that acts diagonally on basis states $|C\rangle$ (for $C \in ob(\mathbf{C})$), whence that satisfies for all $C \in ob(\mathbf{C})$

$$O|C\rangle = \omega_O(C)|C\rangle \quad (\omega_O(C) \in \mathbb{R}).$$

Comparing this equation to the definition of the canonical representation (Definition 4.4) of a generic rule algebra basis element $\delta(p) \in \mathcal{R}_{\mathbf{C}}$ (for $p \equiv (I \xleftarrow{i} K \xrightarrow{o} O) \in Lin(\mathbf{C})$),

$$\rho_{\mathbf{C}}(\delta(p))|C\rangle := \begin{cases} \sum_{m \in \mathcal{M}_p(C)} |p_m(C)\rangle & \text{if } \mathcal{M}_p(C) \neq \emptyset \\ 0_{\hat{\mathbf{C}}} & \text{else,} \end{cases}$$

we find that in order for $\rho_{\mathbf{C}}(\delta(p))$ to be diagonal we must have

$$\forall C \in ob(\mathbf{C}) : \forall m \in \mathcal{M}_p(C) : p_m(C) = C.$$

But by definition of derivations of objects along admissible matches (Definition 2.6), the only linear rules $p \in Lin(\mathbf{C})$ that have this special property are precisely the rules of the form

$$p_M^r = (M \xleftarrow{r} K \xrightarrow{r} M).$$

In particular, defining $O_M^r := \rho_{\mathbf{C}}(\delta(p_M^r))$, we find that the eigenvalue $\omega_{O_M^r}(C)$ coincides with the cardinality of the set $\mathcal{M}_{p_M^r}(C)$ of admissible matches,

$$\forall C \in ob(\mathbf{C}) : O_M^r|C\rangle = |\mathcal{M}_{p_M^r}(C)| \cdot |C\rangle.$$

This proves that the operators O_M^r form a basis of diagonal operators on $End(\mathbf{C})$ (and thus on $End(\mathcal{S}_{\mathbf{C}})$).

To prove the jump-closure property, note that it follows from Definition 2.6 that for an arbitrary linear rule $p \equiv (I \xleftarrow{i} K \xrightarrow{o} O) \in Lin(\mathbf{C})$, a generic object $C \in \mathbf{C}$ and a monomorphism $m : I \rightarrow C$, the admissibility of m as a match is determined by whether or not the match fulfills the gluing condition (Definition 2.3), i.e. whether or not the following pushout complement exists,

$$\begin{array}{ccc} I & \xleftarrow{i} & K \\ \downarrow m & \lrcorner & \downarrow g \\ C & \xleftarrow{v} & E \end{array}.$$

Thus we find that with $p' = (I \xleftarrow{i} K \xrightarrow{i} I) \in Lin(\mathbf{C})$, the set $\mathcal{M}_p(C)$ of admissible matches of p in C and $\mathcal{M}_{p'}(C)$ of p' in C have the same cardinality. Combining this with the definition of the projection operator $\langle |$ (Definition 7.2),

$$\forall C \in ob(\mathbf{C}) : \langle |C\rangle := 1_{\mathbb{R}},$$

we may prove the claim of the jump-closure property via verifying it on arbitrary basis elements (with notations as above):

$$\langle | \rho_{\mathbf{C}}(\delta(p)) |C\rangle = |\mathcal{M}_p(C)| = |\mathcal{M}_{p'}(C)| = \langle | \rho_{\mathbf{C}}(\delta(p')) |C\rangle.$$

Since $C \in \text{ob}(\mathbf{C})$ was chosen arbitrarily, we thus have indeed that

$$\langle | \rho_{\mathbf{C}}(\delta(p)) = \langle | \rho_{\mathbf{C}}(\delta(p')) .$$

Finally, combining all of these findings, one may verify that H as stated in the theorem fulfills all required properties in order to qualify as an infinitesimal generator of a continuous-time Markov chain. ◀

Submodular Functions and Valued Constraint Satisfaction Problems over Infinite Domains

Manuel Bodirsky¹

Institut für Algebra, Technische Universität Dresden
Germany
manuel.bodirsky@tu-dresden.de

Marcello Mamino

Dipartimento di Matematica, Università di Pisa
Italy
marcello.mamino@dm.unipi.it

Caterina Viola²

Institut für Algebra, Technische Universität Dresden
Germany
caterina.viola@tu-dresden.de

Abstract

Valued constraint satisfaction problems (VCSPs) are a large class of combinatorial optimisation problems. It is desirable to classify the computational complexity of VCSPs depending on a fixed set of allowed cost functions in the input. Recently, the computational complexity of all VCSPs for finite sets of cost functions over finite domains has been classified in this sense. Many natural optimisation problems, however, cannot be formulated as VCSPs over a finite domain. We initiate the systematic investigation of infinite-domain VCSPs by studying the complexity of VCSPs for piecewise linear homogeneous cost functions. We remark that in this paper the infinite domain will always be the set of rational numbers. We show that such VCSPs can be solved in polynomial time when the cost functions are additionally submodular, and that this is indeed a maximally tractable class: adding any cost function that is not submodular leads to an NP-hard VCSP.

2012 ACM Subject Classification Mathematics of computing → Mathematical optimization, Theory of computation → Complexity theory and logic

Keywords and phrases Valued constraint satisfaction problems, Piecewise linear functions, Submodular functions, Semilinear, Constraint satisfaction, Optimisation, Model Theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.12

1 Introduction

In a *valued constraint satisfaction problem (VCSP)* we are given a finite set of variables, a finite set of cost functions that depend on these variables, and a cost u ; the task is to find values for the variables such that the sum of the cost functions is less than u . By restricting the set of possible cost functions in the input, a great variety of computational optimisation

¹ The first and second author have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681988, CSP-Infinity).

The first author has also received funding from the DFG (Project number 622397)

² This author is supported by DFG Graduiertenkolleg 1763 (QuantLA).



problems can be modelled as a valued constraint satisfaction problem. By allowing the cost functions to evaluate to $+\infty$, we can even model ‘crisp’ constraints, given by relations that have to be satisfied by the variable assignments. Hence the class of (classical) constraint satisfaction problems (CSPs) is a subclass of the class of all VCSPs.

If the domain is finite, the computational complexity of the VCSP has recently been classified for all sets of cost functions, assuming the Feder-Vardi conjecture for classical CSPs [14, 13, 15]. Even more recently, two solutions to the Feder-Vardi conjecture have been announced [18, 6]. These fascinating achievements settle the complexity of the VCSP over finite domains.

Several outstanding combinatorial optimisation problems cannot be formulated as VCSPs over a finite domain, but they can be formulated as VCSPs over the domain \mathbb{Q} , the set of rational numbers. One example is the famous linear programming problem, where the task is to optimise a linear function subject to linear inequalities. This can be modelled as a VCSP by allowing unary linear cost functions and cost functions of higher arity to express the crisp linear inequalities. Another example is the minimisation problem for sums of piecewise linear convex cost functions (see, e.g., [5]). Both of these problems can be solved in polynomial time, e.g. by the ellipsoid method (see, e.g., [10]).

Despite the great interest in such concrete VCSPs over the rational numbers in the literature, VCSPs over infinite domains have not yet been studied systematically. In order to obtain general results we need to restrict the class of cost functions that we investigate, because without any restriction it is already hopeless to classify the complexity of infinite-domain CSPs (any language over a finite alphabet is polynomial-time Turing equivalent to an infinite domain CSP [2]). One restriction that captures a variety of optimisation problems of theoretical and practical interest is the class of all *piecewise linear homogeneous* cost functions over \mathbb{Q} , defined below. We first illustrate by an example the type of cost functions that we want to capture in our framework.

► **Example 1.1.** An internet provider charges the clients depending on the amount of data x downloaded and the amount of data y that is uploaded. The cost function of the provider could be the partial function $f: \mathbb{Q}^2 \rightarrow \mathbb{Q}$ given by

$$f(x, y) := \begin{cases} 3x & \text{if } 0 \leq y < 2x \\ \frac{3}{2}y & \text{if } 0 \leq 2x \leq y \\ \text{undefined} & \text{otherwise.} \end{cases}$$

A partial function $f: \mathbb{Q}^n \rightarrow \mathbb{Q}$ is called *piecewise linear homogeneous (PLH)* if it is first-order definable over the structure $\mathfrak{L} := (\mathbb{Q}; <, 1, (c \cdot)_{c \in \mathbb{Q}})$; being undefined at $(x_1, \dots, x_n) \in \mathbb{Q}^n$ is interpreted as $f(x_1, \dots, x_n) = +\infty$. The structure \mathfrak{L} has quantifier elimination (see Section 3.2) and hence there are finitely many regions such that f is a homogeneous linear polynomial in each region; this is the motivation for the name *piecewise linear homogeneous*. The cost function from Example 1.1 is PLH.

The cost function in Example 1.1 satisfies an additional important property: it is *submodular* (defined in Section 3.3). Submodular cost functions naturally appear in several scientific fields such as, for example, economics, game theory, machine learning, and computer vision, and play a key role in operational research and combinatorial optimisation (see, e.g., [9]). Submodularity also plays an important role for the computational complexity of VCSPs over finite domains, and guided the research on VCSPs for some time (see, e.g., [7, 12]), even though this might no longer be visible in the final classification obtained in [14, 13, 15].

In this paper we show that VCSPs for submodular PLH cost functions can be solved in polynomial time (Theorem 5.1 in Section 5). To solve this problem, we first describe how to solve the feasibility problem (does there exist a solution satisfying all crisp constraints) and then how to find the optimal solution. The first step follows from a new, more general polynomial-time tractability result, namely for *max-closed* PLH constraints (Section 4). To then solve the optimisation problem for PLH constraints, we introduce a technique to reduce the task to a problem over a finite domain that can be solved by a fully combinatorial polynomial-time algorithm for submodular set-function optimisation by Iwata and Orlin [11]. Moreover, we show that submodularity defines a *maximal tractable class*: adding any cost function that is submodular leads to an NP-hard VCSP (Section 6). Section 7 closes with some problems and challenges.

2 Valued Constraint Satisfaction Problems

A *valued constraint language* Γ (over D) (or simply *language*) consists of

- a signature τ consisting of function symbols f , each equipped with an arity $\text{ar}(f)$,
- a set $D = \text{dom}(\Gamma)$ (the *domain*),
- for each $f \in \tau$ a *cost function*, i.e., a function $f^\Gamma : D^{\text{ar}(f)} \rightarrow \mathbb{Q} \cup \{+\infty\}$.

Here, $+\infty$ is an extra element with the expected properties that for all $c \in \mathbb{Q} \cup \{+\infty\}$

$$\begin{aligned} (+\infty) + c &= c + (+\infty) = +\infty \\ \text{and } c < +\infty &\text{ iff } c \in \mathbb{Q}. \end{aligned}$$

Let Γ be a valued constraint language with a finite signature τ . The *valued constraint satisfaction problem* for Γ , denoted by $\text{VCSP}(\Gamma)$, is the following computational problem.

► **Definition 2.1.** An *instance* I of $\text{VCSP}(\Gamma)$ consists of

- a finite set of variables V_I ,
- an expression ϕ_I of the form

$$\sum_{i=1}^m f_i(x_1^i, \dots, x_{\text{ar}(f_i)}^i)$$

where $f_1, \dots, f_m \in \tau$ and all the x_j^i are variables from V_I , and

- a value $u_I \in \mathbb{Q} \cup \{+\infty\}$.

The task is to decide whether there exists a map $\alpha : V_I \rightarrow \text{dom}(\Gamma)$ whose *cost*, defined as

$$\sum_{i=1}^m f_i^\Gamma(\alpha(x_1^i), \dots, \alpha(x_{\text{ar}(f_i)}^i))$$

is finite, and if so, whether there is one whose cost is smaller or equal to u_I .

A *solution* of an instance of $\text{VCSP}(\Gamma)$ is a tuple $x \in D^{|V_I|}$ such that $x \in \text{dom}(f)$ for all valued constraint f in the instance.

Note that since the signature τ of Γ is finite, it is inessential for the computational complexity of $\text{VCSP}(\Gamma)$ how the function symbols in ϕ_I are represented. The function described by the expression ϕ_I is also called the *objective function*. When $u_I = +\infty$ then this problem is called the *feasibility* problem, which can also be modelled as a (classical) constraint satisfaction problem. The choice of defining the VCSP as a decision problem and not as an optimisation problem is motivated by two major issues that do not occur in the finite domain case: in the infinite domain setting one needs to decide whether the infimum is attained, and to model the case in which the infimum is $-\infty$.

12:4 Submodular Functions and VCSPs over Infinite Domains

Many well-known optimisation problems can only be formulated when we allow infinite domains D .

► **Example 2.2.** Let Γ be the valued constraint language with signature $\tau = \{g_1, g_2, g_3\}$ and the cost functions

- $g_1^\Gamma: \mathbb{Q} \rightarrow \mathbb{Q}$ defined by $g_1(x) = -x$,
- $g_2^\Gamma: \mathbb{Q}^2 \rightarrow \mathbb{Q}$ defined by $g_2(x, y) := \min(x, -y)$, and
- $g_3^\Gamma: \mathbb{Q}^3 \rightarrow \mathbb{Q}$ defined by $g_3(x, y, z) := \max(x, y, z)$.

Two examples of instances of VCSP(Γ) are

$$g_1(x) + g_1(y) + g_1(z) + g_2(x, y) + g_3(x, y, z) + g_3(x, x, x) + g_3(x, x, x) \quad (1)$$

and $g_1(x) + g_1(y) + g_1(z) + g_3(x, y, z) + g_3(x, x, y) + g_3(y, z, z) \quad (2)$

We can make the cost function described by the expression in (1) arbitrarily small by fixing x to 0 and choosing y and z sufficiently large. On the other hand, the minimum for the cost function in (2) is 0, obtained by setting x, y, z to 0. Note that g_1 and g_3 are convex functions, but g_2 is not, nevertheless, as we will see later, VCSP(Γ) can be solved in polynomial time.

3 Cost functions over the rationals

In this section we describe natural and large classes of cost functions over the domain $D = \mathbb{Q}$, the rational numbers. These classes are most naturally introduced using first-order definability.

We give two examples of structures that play an important role in this article.

- **Example 3.1.** Let \mathfrak{S} be the structure with domain \mathbb{Q} and the signature $\{+, 1, \leq\}$ where
- $+$ is a binary function symbol that denotes the usual addition over \mathbb{Q} ,
 - 1 is a constant symbol that denotes $1 \in \mathbb{Q}$, and
 - \leq is a binary relation symbol that denotes the usual linear order of the rationals.

- **Example 3.2.** Let \mathfrak{L} be the structure with the (countably infinite) signature $\tau_0 := \{<, 1\} \cup \{c \cdot\}_{c \in \mathbb{Q}}$ where
- $<$ is a relation symbol of arity 2 and $<^\mathfrak{L}$ is the strict linear order of \mathbb{Q} ,
 - 1 is a constant symbol and $1^\mathfrak{L} := 1 \in \mathbb{Q}$, and
 - $c \cdot$ is a unary function symbol for every $c \in \mathbb{Q}$ such that $(c \cdot)^\mathfrak{L}$ is the function $x \mapsto cx$ (multiplication by c).

3.1 Quantifier Elimination

Let τ be a signature. We adopt the usual definition of first-order logic.

We say that a τ -structure \mathfrak{A} has *quantifier elimination* if every first-order τ -formula is equivalent to a quantifier-free τ -formula over \mathfrak{A} .

► **Theorem 3.3** ([8]). *The structure \mathfrak{S} from Example 3.1 has quantifier elimination.*

► **Theorem 3.4.** *The structure \mathfrak{L} from Example 3.2 has quantifier elimination.*

Proof. See the appendix. ◀

Observe that every atomic τ_0 -formula has at most two variables:

- if it has no variables, then it is equivalent to \top or \perp ,
- if it has only one variable, say x , then it is equivalent to $c \cdot x \sigma d \cdot 1$ or to $d \cdot 1 \sigma c \cdot x$ for $\sigma \in \{<, =\}$ and $c, d \in \mathbb{Q}$. Moreover, if $c = 0$ then it is equivalent to a formula without variables, and otherwise it is equivalent to $x \sigma \frac{d}{c} \cdot 1$ or to $\frac{d}{c} \cdot 1 \sigma x$ for $\sigma \in \{<, =\}$, which we abbreviate by the more common $x < \frac{d}{c}$, $x = \frac{d}{c}$, and $\frac{d}{c} < x$, respectively.
- if it has two variables, say x and y , then it is equivalent to $c \cdot x \sigma d \cdot y$ or $c \cdot x \sigma d \cdot y$ for $\sigma \in \{<, =\}$. Moreover, if $c = 0$ or $d = 0$ then the formula is equivalent to a formula with at most one variable, and otherwise it is equivalent to $x \sigma \frac{d}{c} \cdot y$ or to $\frac{d}{c} \cdot y \sigma x$.

3.2 Piecewise Linear Homogeneous Functions

A *partial function* of arity $n \in \mathbb{N}$ over a set A is a function

$$f: \text{dom}(f) \rightarrow A \text{ for some } \text{dom}(f) \subseteq A^n.$$

Let \mathfrak{A} be a τ -structure with domain A . A partial function over A is called *first-order definable over \mathfrak{A}* if there exists a first-order τ -formula $\phi(x_0, x_1, \dots, x_n)$ such that for all $a_1, \dots, a_n \in A$

- if $(a_1, \dots, a_n) \in \text{dom}(f)$ then $\mathfrak{A} \models \phi(a_0, a_1, \dots, a_n)$ if and only if $a_0 = f(a_1, \dots, a_n)$, and
- if $f(a_1, \dots, a_n) \notin \text{dom}(f)$ then there is no $a_0 \in A$ such that $\mathfrak{A} \models \phi(a_0, a_1, \dots, a_n)$.

In the following, we consider *cost functions over \mathbb{Q}* , which will be functions from $\mathbb{Q}^n \rightarrow \mathbb{Q} \cup \{+\infty\}$. It is sometimes convenient to view a cost function as a partial function over \mathbb{Q} . If $t \in \mathbb{Q}^{\text{ar}(f)} \setminus \text{dom}(f)$ we interpret this as $f(t) = +\infty$.

- **Definition 3.5.** A cost function $f: \mathbb{Q}^n \rightarrow \mathbb{Q} \cup \{+\infty\}$ (viewed as a partial function) is called
- *piecewise linear (PL)* if it is first-order definable over \mathfrak{S} , piecewise linear functions are sometimes called *semilinear* functions;
 - *piecewise linear homogeneous (PLH)* if it is first-order definable over \mathfrak{L} (viewed as a partial function).

A valued constraint language Γ is called *piecewise linear (piecewise linear homogeneous)* if every cost function in Γ is PL (or PLH, respectively).

Every piecewise linear homogeneous cost function is also piecewise linear, since all functions of the structure \mathfrak{L} are clearly first-order definable in \mathfrak{S} . The cost functions in the valued constraint language from Example 2.2 are PLH.

We would like to point out that already the class of PLH cost functions is very large. In particular, one can view it as a generalisation of the class of all cost functions over a finite domain D . Indeed, every VCSP for a valued constraint language over a finite domain is also a VCSP for a language that is PLH. To see this, suppose that $f: D^d \rightarrow \mathbb{Q} \cup \{+\infty\}$ is such a cost function, identifying D with a subset of \mathbb{Q} in an arbitrary way. Then the function $f': \mathbb{Q}^d \rightarrow \mathbb{Q} \cup \{+\infty\}$ defined by $f'(x_1, \dots, x_n) := f(x_1, \dots, x_n)$ if $x_1, \dots, x_n \in D$, and $f'(x_1, \dots, x_n) = +\infty$ otherwise, is PLH.

3.3 Submodularity

Let D be a set. When $x^1, \dots, x^k \in D^n$ and $g: D^k \rightarrow D$ is a function, then $g(x^1, \dots, x^k)$ denotes the n -tuple obtained by applying g *component-wise*, i.e.,

$$g(x^1, \dots, x^k) := (g(x_1^1, \dots, x_1^k), \dots, g(x_n^1, \dots, x_n^k)).$$

► **Definition 3.6.** Let D be a totally ordered set and let G be a totally ordered Abelian group. A partial function $f: D^n \rightarrow G$ is called *submodular* if for all $x, y \in D^n$

$$f(\max(x, y)) + f(\min(x, y)) \leq f(x) + f(y).$$

Note that in particular if $x, y \in \text{dom}(f)$, then $\min(x, y) \in \text{dom}(f)$ and $\max(x, y) \in \text{dom}(f)$. All cost functions in Example 2.2 are submodular.

Other examples of submodular PLH functions are those that can be written as the maximum of two increasing linear homogeneous functions or as the minimum of two linear homogeneous functions with different monotonicity.

4 Tractability of Max-Closed PLH Constraints

The question whether an instance of $\text{VCSP}(\Gamma)$ is *feasible*, namely has a solution, can be viewed as a (classical) constraint satisfaction problem. Formally, the constraint satisfaction problem for a structure \mathfrak{A} with a finite relational signature τ is the following computational problem, denoted by $\text{CSP}(\mathfrak{A})$:

- the input is a finite conjunction ψ of atomic τ -formulas, and
- the question is whether ψ is satisfiable in \mathfrak{A} .

We can associate to Γ the following relational structure $\text{Feas}(\Gamma)$: for every cost function f of arity n from Γ the signature of $\text{Feas}(\Gamma)$ contains a relation symbol R_f of arity n such that $R_f^{\text{Feas}(\Gamma)} = \text{dom}(f)$.

Every polynomial-time algorithm for $\text{VCSP}(\Gamma)$, in particular, has to solve $\text{CSP}(\text{Feas}(\Gamma))$. In fact, an instance ϕ of $\text{VCSP}(\Gamma)$ can be translated into an instance ψ of $\text{CSP}(\text{Feas}(\Gamma))$ by replacing subexpressions of the form $f(x_1, \dots, x_n)$ in ϕ by $R_f(x_1, \dots, x_n)$ and by replacing $+$ by \wedge . It is easy to see that ϕ is a feasible instance of $\text{VCSP}(\Gamma)$ if and only if ψ is satisfiable in $\text{Feas}(\Gamma)$.

► **Definition 4.1.** Let \mathfrak{A} be a structure with relational signature τ and domain A . Then a function $g: A^k \rightarrow A$ is called a *polymorphism* of \mathfrak{A} if for all $R \in \tau$ we have that $R^{\mathfrak{A}}$ is *preserved* by g , namely $g(x^1, \dots, x^k) \in R^{\mathfrak{A}}$ for all $x^1, \dots, x^k \in R^{\mathfrak{A}}$ (where g is applied component-wise).

► **Definition 4.2.** A relation $R \subseteq \mathbb{Q}^n$ is called *piecewise linear homogeneous (PLH)* if it is first-order definable over \mathfrak{L} (see Example 3.2).

In general, a valued constraint language can have infinitely many cost functions. If we consider Γ to be a finite submodular PLH valued constraint language, then $\text{Feas}(\Gamma)$ is a relational structure all of whose relations are

- PLH, and
- preserved by the polymorphisms \max and \min .

We observed that for the polynomial-time tractability of $\text{VCSP}(\Gamma)$ we need, in particular, that $\text{CSP}(\text{Feas}(\Gamma))$ be tractable. In this section we prove a more general result:

► **Theorem 4.3.** *Let \mathfrak{A} be a structure having domain \mathbb{Q} and finite relational signature τ . Assume that for all $R \in \tau$, the interpretation $R^{\mathfrak{A}}$ is PLH and preserved by \max . Then $\text{CSP}(\mathfrak{A})$ is polynomial-time solvable.*

This result is incomparable to known results about max-closed semilinear relations [4]. In particular, there, the weaker bound $\text{NP} \cap \text{co-NP}$ has been shown for a larger class, and

polynomial tractability only for a smaller class (which does not contain many max-closed PLH relations, for instance $x \geq \max(y, z)$).

We use a technique introduced in [3] which relies on the following concept.

► **Definition 4.4.** Let \mathfrak{A} be a structure with a finite relational signature τ . A *sampling algorithm* for \mathfrak{A} takes as input a positive integer d and computes a finite τ -structure \mathfrak{B} such that every finite conjunction of atomic τ -formulas having at most d distinct free variables is satisfiable in \mathfrak{A} if, and only if, it is satisfiable in \mathfrak{B} . A sampling algorithm is called *efficient* if its running time is bounded by a polynomial in d .

The definition above is a slight re-formulation of Definition 2.2 in [3], and it is easily seen to give the same results using the same proofs. We decided to bound the number of variables instead of the size of the conjunction of atomic τ -formulas because this is more natural in our context. These two quantities are polynomially related by the assumption that the signature τ is finite.

► **Definition 4.5.** A k -ary function $g: D^k \rightarrow D$ is called *totally symmetric* if $g(x_1, \dots, x_k) = g(y_1, \dots, y_k)$ for all $x_1, \dots, x_k, y_1, \dots, y_k \in D$ such that $\{x_1, \dots, x_k\} = \{y_1, \dots, y_k\}$.

► **Theorem 4.6** (Bodirsky-Macpherson-Thapper, [3], Theorem 2.5). *Let \mathfrak{A} be a structure over a finite relational signature with totally symmetric polymorphisms of all arities. If there exists an efficient sampling algorithm for \mathfrak{A} then $\text{CSP}(\mathfrak{A})$ is in P .*

In this section, we study $\text{CSP}(\mathfrak{A})$, where \mathfrak{A} is a τ -structure satisfying the hypothesis of Theorem 4.3. We give a formal definition of the *numerical data* in \mathfrak{A} , we will need it later on. By quantifier elimination (Theorem 3.4), we can write each of the finitely many relations $R^{\mathfrak{A}}$ for $R \in \tau$ as a quantifier-free τ_0 -formula ϕ_R . We can assume (as in the proof of Theorem 3.4) that all formulas ϕ_R are positive (namely contain no negations). From now on, we will fix one such representation. Let $\text{At}(\phi_R)$ denote the set of atomic subformulas of ϕ_R . Each atomic τ_0 -formula is of the form $t_1 \leq t_2$, where t_1 and t_2 are terms. We call the atomic formula non-trivial if it is not equivalent to \perp or \top , from now on we make the following assumptions on the atomic formulas (cf. again the proof of Theorem 3.4)

- that atomic formulas except \top, \perp are non-trivial
- that the functions $k \cdot$ are never composed, because $k \cdot h \cdot x$ can be replaced by $(kh) \cdot x$
- that, in any atomic formula $k \cdot x_i \leq h \cdot x_j$, the constants k and h are not both negative.

Given a set of non-trivial atomic formulas Φ , we define

$$H(\Phi) = \left\{ \frac{c_1}{c_2} \mid t_1 = c_1 \cdot x_i, t_2 = c_2 \cdot x_j, \text{ for some } t_1 \leq t_2 \text{ in } \Phi \right\}$$

$$K(\Phi) = \left\{ \frac{c_2}{c_1} \mid t_1 = c_1 \cdot x_i, t_2 = c_2 \cdot 1, \text{ for some } t_1 \leq t_2 \text{ in } \Phi \right\} \\ \cup \left\{ \frac{c_1}{c_2} \mid t_1 = c_1 \cdot 1, t_2 = c_2 \cdot x_j, \text{ for some } t_1 \leq t_2 \text{ in } \Phi \right\}$$

We describe now the numeric domain \mathbb{Q}^* in which our algorithm operates.

► **Definition 4.7.** We call \mathbb{Q}^* the ordered \mathbb{Q} -vector space

$$\mathbb{Q}^* = \{x + y\epsilon \mid x, y \in \mathbb{Q}\}$$

12:8 Submodular Functions and VCSPs over Infinite Domains

where ϵ is merely a formal device, namely $x + y\epsilon$ represents the pair (x, y) . We define addition and multiplication by a scalar component-wise

$$\begin{aligned}(x_1 + y_1\epsilon) + (x_2 + y_2\epsilon) &= (x_1 + x_2) + (y_1 + y_2)\epsilon \\ c \cdot (x + y\epsilon) &= (cx) + (cy)\epsilon.\end{aligned}$$

The order is induced by \mathbb{Q} extended with $0 < \epsilon \ll 1$, namely the lexicographical order of the components x and y

$$(x_1 + y_1\epsilon) < (x_2 + y_2\epsilon) \quad \text{iff} \quad \begin{cases} x_1 < x_2 & \text{or} \\ x_1 = x_2 \wedge y_1 < y_2. \end{cases}$$

\mathbb{Q} is clearly embedded in \mathbb{Q}^* (the embedding is given by the map $k \mapsto k + 0\epsilon$).

Any τ_0 -formula has an obvious interpretation in any ordered \mathbb{Q} -vector space Q extending \mathbb{Q} , and, in particular, in \mathbb{Q}^* .

► **Proposition 4.8.** *Let $\phi(x_1 \dots x_d)$ and $\psi(x_1 \dots x_d)$ be τ_0 -formulas. Then ϕ and ψ are equivalent in \mathbb{Q} if, and only if, they are equivalent in any ordered \mathbb{Q} -vector space Q extending \mathbb{Q} (for instance $Q = \mathbb{Q}^*$).*

Proof. It follows from [17, Chapter 1, Remark 7.9] that the first-order theory of ordered \mathbb{Q} -vector spaces in the signature $\tau_0 \cup \{+, -\}$ is complete. As a consequence the formula $\forall x_1 \dots x_d \phi(x_1 \dots x_d) \leftrightarrow \psi(x_1 \dots x_d)$ holds in \mathbb{Q} if and only if it does in Q . ◀

The proposition gives us a natural extension \mathfrak{A}^* of \mathfrak{A} to the domain \mathbb{Q}^* . Namely the τ -structure obtained by interpreting each relation symbol $R \in \tau$ by the relation $R^{\mathfrak{A}^*}$ defined on \mathbb{Q}^* by the same (quantifier-free) τ_0 -formula ϕ_R that defines $R^{\mathfrak{A}}$ over \mathbb{Q} (by the proposition, the choice of equivalent τ_0 -formulas is immaterial). Similarly, we will see that, as long as satisfiability is concerned, there is no difference between \mathfrak{A} and \mathfrak{A}^* .

► **Corollary 4.9.** *Let ϕ be an instance of $\text{CSP}(\mathfrak{A})$, and let ϕ^* be the corresponding instance of $\text{CSP}(\mathfrak{A}^*)$. Then ϕ is satisfiable if and only if ϕ^* is.*

Proof. From Proposition 4.8 observing that ϕ (resp. ϕ^*) is unsatisfiable if and only if it is equivalent to \perp . ◀

As a consequence, we can work in the extended structure \mathfrak{A}^* . Our goal is to prove the following theorem.

► **Theorem 4.10.** *There is an efficient sampling algorithm for \mathfrak{A}^* .*

Assuming, for a moment, Theorem 4.10, it is easy to prove Theorem 4.3.

Proof of Theorem 4.3. By Proposition 4.7, for all $k \geq 1$ the function

$$(x_1, \dots, x_k) \mapsto \max(x_1, \dots, x_k)$$

is a k -ary totally symmetric polymorphism of $\text{CSP}(\mathfrak{A}^*)$. Therefore, $\text{CSP}(\mathfrak{A}^*)$ is in P by Theorem 4.10 and Theorem 4.6. Finally, by Corollary 4.9, $\text{CSP}(\mathfrak{A}^*)$ and $\text{CSP}(\mathfrak{A})$ are equivalent. ◀

Let ϕ be an atomic τ_0 -formula. We write $\bar{\phi}$ for the formula $t_1 \leq t_2$ if ϕ is of the form $t_1 < t_2$, and for the formula $t_1 = t_2$ if ϕ is of the form $t_1 = t_2$.

► **Lemma 4.11.** *Let Φ be a finite set of atomic τ_0 -formulas having free variables in $\{x_1 \dots x_d\}$. Assume that $\bar{\Phi} := \bigcup_{\phi \in \Phi} \bar{\phi}$ has a simultaneous solution $(x_1 \dots x_d) \in \mathbb{Q}^{>0}$ in positive numbers. Then $\bar{\Phi}$ has a solution taking values in the set $C_{\Phi,d} \subset \mathbb{Q}$ defined as follows*

$$C_{\Phi,d} = \left\{ |k| \prod_{i=1}^s |h_i|^{e_i} \mid k \in K(\Phi), e_1 \dots e_s \in \mathbb{Z}, \sum_{r=1}^s |e_r| < d \right\}$$

where $h_1 \dots h_s$ is an enumeration of the (finitely many) elements of $H(\Phi)$.

Proof. See the appendix. ◀

► **Lemma 4.12.** *Let Φ be a finite set of atomic τ_0 -formulas having free variables in $\{x_1 \dots x_d\}$. Assume that the formulas in Φ are simultaneously satisfiable in \mathbb{Q} . Then they are simultaneously satisfiable in $D_{\Phi,d} := -C_{\Phi,d}^* \cup \{0\} \cup C_{\Phi,d}^*$ where*

$$C_{\Phi,d}^* = \{x + nx\epsilon \mid x \in C_{\Phi,d}, n \in \mathbb{Z}, -d \leq n \leq d\}$$

$C_{\Phi,d}$ is defined as in Lemma 4.11, and $-C_{\Phi,d}^*$ denotes the set $\{-x \mid x \in C_{\Phi,d}^*\}$.

Proof. See the appendix. ◀

Proof of Theorem 4.10. The sampling algorithm produces the finite substructure $\mathfrak{A}_{\text{At}(\tau),d}^*$ of \mathfrak{A}^* having domain $D_{\text{At}(\tau),d}$ where $\text{At}(\tau) := \bigcup_{R \in \tau} \text{At}(\phi_R)$, namely the τ -structure with domain $D_{\text{At}(\tau),d}$ in which each relation symbol $R \in \tau$ denotes the restriction of $R^{\mathfrak{A}^*}$ to $D_{\text{At}(\tau),d}$. It is immediate to observe that this structure has size polynomial in d .

Since $\mathfrak{A}_{\text{At}(\tau),d}^*$ is a substructure of \mathfrak{A}^* , it is clear that if an instance is satisfiable in $\mathfrak{A}_{\text{At}(\tau),d}^*$, then it is a fortiori satisfiable in \mathfrak{A}^* .

The vice versa follows from Lemma 4.12. In fact, consider a set Ψ of atomic τ -formulas having free variables $x_1 \dots x_d$. Assume that Ψ is satisfied in \mathfrak{A}^* by one assignment $x_i = a_i$ for $i \in \{1 \dots d\}$. For each $\phi_R \in \Psi$ let $\Phi_R \subset \text{At}(\phi_R)$ be the set of atomic subformulas of ϕ_R which are satisfied by our assignment a_i . Clearly the atomic τ_0 -formulas $\Phi := \bigcup_{\phi_R \in \Psi} \Phi_R$ are simultaneously satisfiable. Remembering that the formulas ϕ_R have no negations by construction, it is obvious that any simultaneous solution of Φ must also satisfy Ψ . By Lemma 4.12, Φ has a solution in the set $D_{\Phi,d}$ defined therein. We can observe that $C_{\Phi,d} \subset C_{\text{At}(\tau),d}$, hence $D_{\Phi,d} \subset D_{\text{At}(\tau),d}$ and the claim follows. ◀

5 Tractability of Submodular PLH Valued Constraints

Here we extend the method developed in Section 4 to the treatment of VCSPs. To better highlight the parallel with Section 4, so that the reader already familiar with it may quickly get an intuition of the arguments here, we will use identical notations to represent corresponding objects. This choice has the drawback that some symbols, notably \mathbb{Q}^* , need to be re-defined (the new \mathbb{Q}^* , for instance, will contain the old one). In this section, we will sometimes skip details that can be borrowed unchanged from Section 4.

Our goal is to prove the following result

► **Theorem 5.1.** *Let Γ be a PLH valued finite constraint language. Assume that all cost functions in Γ are submodular. Then $\text{VCSP}(\Gamma)$ is polynomial-time solvable.*

Let us begin with the new definition of \mathbb{Q}^* .

12:10 Submodular Functions and VCSPs over Infinite Domains

► **Definition 5.2.** We let \mathbb{Q}^* denote the ring $\mathbb{Q}((\epsilon))$ of formal Laurent power series in the indeterminate ϵ . Namely \mathbb{Q}^* is the set of formal expressions

$$\sum_{i=-\infty}^{+\infty} a_i \epsilon^i$$

where $a_i \neq 0$ for only finitely many *negative* values of i . Clearly \mathbb{Q} is embedded in \mathbb{Q}^* . The ring operations on \mathbb{Q}^* are defined as usual

$$\begin{aligned} \sum_{i=-\infty}^{+\infty} a_i \epsilon^i + \sum_{i=-\infty}^{+\infty} b_i \epsilon^i &= \sum_{i=-\infty}^{+\infty} (a_i + b_i) \epsilon^i \\ \sum_{i=-\infty}^{+\infty} a_i \epsilon^i \cdot \sum_{i=-\infty}^{+\infty} b_i \epsilon^i &= \sum_{i=-\infty}^{+\infty} \left(\sum_{j=-\infty}^{+\infty} a_j b_{i-j} \right) \epsilon^i \end{aligned}$$

where the sum in the product definition is always finite by the hypothesis on a_i, b_i with negative index i . The order is the lexicographical order induced by $0 < \epsilon \ll 1$, namely

$$\sum_{i=-\infty}^{+\infty} a_i \epsilon^i < \sum_{i=-\infty}^{+\infty} b_i \epsilon^i \quad \text{iff} \quad \exists i \ a_i < b_i \wedge \forall j < i \ a_j = b_j.$$

It is well known that \mathbb{Q}^* is an ordered field, namely all non-zero elements have a multiplicative inverse and the order is compatible with the field operations. We define the following subsets of \mathbb{Q}^* for $m \leq n$

$$\mathbb{Q}_{m,n}^* := \left\{ \sum_{i=m}^n \epsilon^i a_i \mid a_i \in \mathbb{Q} \right\} \subset \mathbb{Q}^*$$

► **Definition 5.3.** We define a new structure \mathfrak{L}^* , that is both an extension and an expansion of \mathfrak{L} (see Example 3.2), namely it has \mathbb{Q}^* as domain and $\tau_1 := \tau_0 \cup \{k\}_{k \in \mathbb{Q}_{-1,1}^*}$ as signature, where the interpretation of symbols in τ_0 is formally the same as for \mathfrak{L} and the symbols $k \in \mathbb{Q}_{-1,1}^*$ denote constants (zero-ary functions).

Notice that, for technical reasons, we allow only constants in $\mathbb{Q}_{-1,1}^*$. During the rest of this section, τ_1 -formulas will be interpreted in the structure \mathfrak{L}^* . We make on τ_1 -formulas the same assumptions of Section 4 (that atomic subformulas are non-trivial and not negated), also $H(\Phi)$ and $K(\Phi)$ where Φ is a set of atomic τ_1 -formulas are defined similarly to Section 4. Observe that the reduct of \mathfrak{L}^* obtained by restricting the language to τ_0 is *elementarily equivalent* to \mathfrak{L} , namely it satisfies the same first-order sentences.

The following lemmas 5.4, 5.5, and 5.6 are analogues of Lemma 4.11 and Lemma 4.12.

► **Lemma 5.4.** *Let Φ be a finite set of atomic τ_1 -formulas having free variables in $\{x_1 \dots x_d\}$. Call $\bar{\Phi}$ the set $\bar{\phi} \mid \phi \in \Phi$. Suppose that there is $0 < r \in \mathbb{Q}^*$ such that all satisfying assignments of $\bar{\Phi}$ in the domain \mathbb{Q}^* also satisfy $0 < x_i \leq r$ for all i . Let $u, \alpha_1 \dots \alpha_d$ be elements of \mathbb{Q}^* . Assume that the formulas in Φ are simultaneously satisfiable by a point $(x_1 \dots x_d) \in \mathbb{Q}^*$ such that $\sum_i \alpha_i x_i < u$. Let us define the set*

$$C_{\Phi,d} = \left\{ |k| \prod_{i=1}^s |h_i|^{e_i} \mid k \in K(\Phi), e_1 \dots e_s \in \mathbb{Z}, \sum_{r=1}^s |e_r| < d \right\} \subseteq \mathbb{Q}_{-1,1}^*$$

where $h_1 \dots h_s$ is an enumeration of the (finitely many) elements of $H(\Phi)$. Then there is a point in $(x'_1 \dots x'_d) \in C_{\Phi,d}^d \subseteq \mathbb{Q}^*$ with $\sum_i \alpha_i x'_i < u$ that satisfies simultaneously all $\bar{\phi}$, for $\phi \in \Phi$.

Proof. See the appendix. ◀

► **Lemma 5.5.** *Let Φ be a finite set of atomic τ_1 -formulas having free variables in $\{x_1 \dots x_d\}$. Suppose that there are $0 < l < r \in \mathbb{Q}^*$ such that all satisfying assignments of Φ in the domain \mathbb{Q}^* also satisfy $l < x_i < r$ for all i . Let $\alpha_1 \dots \alpha_d$ be rational numbers and $u \in \mathbb{Q}_{-1,1}^*$. Assume that the formulas in Φ are simultaneously satisfiable by a point $(x_1 \dots x_d) \in \mathbb{Q}^*$ such that $\sum_i \alpha_i x_i \leq u$. Then the same formulas are simultaneously satisfiable by a point $(x'_1 \dots x'_d) \in (C_{\Phi,d}^*)^d \subseteq (\mathbb{Q}^*)^d$ such that $\sum_i \alpha_i x'_i \leq u$ where*

$$C_{\Phi,d}^* = \{x + nx\epsilon^3 \mid x \in C_{\Phi,d}, n \in \mathbb{Z}, -d \leq n \leq d\} \subseteq \mathbb{Q}_{-1,4}^*.$$

Proof. See the appendix. ◀

► **Lemma 5.6.** *Let Φ be a finite set of atomic τ_0 -formulas having free variables in $\{x_1 \dots x_d\}$. Let $u, \alpha_1 \dots \alpha_d$ be rational numbers. Then the following are equivalent*

1. *The formulas in Φ are simultaneously satisfiable in \mathbb{Q} , by a point $(x_1 \dots x_d) \in \mathbb{Q}^d$ such that $\sum_i \alpha_i x_i \leq u$.*
2. *The formulas in Φ are simultaneously satisfiable in $D_{\Phi,d} \subseteq \mathbb{Q}^*$, by a point $(x'_1 \dots x'_d) \in D_{\Phi,d}^d$ such that $\sum_i \alpha_i x'_i \leq u$, where the set $D_{\Phi,d}$ is defined as follows*

$$\begin{aligned} D_{\Phi,d} &:= -C_{\Phi',d}^* \cup \{0\} \cup C_{\Phi',d}^* \subseteq \mathbb{Q}_{-1,4}^* \\ \Phi' &:= \Phi \cup \{x > \epsilon, x < -\epsilon, x > -\epsilon^{-1}, x < \epsilon^{-1}\} \end{aligned}$$

Proof. The implication $2 \rightarrow 1$ is immediate observing that the conditions Φ and $\sum_i \alpha_i x_i \leq u$ are first-order definable in \mathfrak{S} . In fact, any assignment with values in $D_{\Phi,d}$ satisfying the conditions is, in particular, an assignment in \mathbb{Q}^* , and, by completeness of the first-order theory of ordered \mathbb{Q} -vector spaces, we have an assignment taking values in \mathbb{Q} .

For the vice versa, fix any assignment $x_i = a_i$ with $a_i \in \mathbb{Q}$ for $i \in \{1 \dots d\}$. We pre-process the formulas in Φ producing a new set of atomic formulas Φ' as follows. We replace all variables x_i such that $a_i = 0$ with the constant $0 = 0 \cdot 1$. Then we replace each of the remaining variables x_i with either y_i or $-y_i$ according to the sign of a_i . Finally, we add the constraints $\epsilon < y_i$ and $y_i < \epsilon^{-1}$ for each of these variables. Similarly we produce new coefficients $\alpha'_i = \text{sign}(a_i)\alpha_i$. It is clear that the new set of formulas Φ' has a satisfying assignment in positive rational numbers with $\sum_i \alpha'_i y_i \leq u$. Observing that a positive rational x always satisfies $\epsilon < x < \epsilon^{-1}$, we see that Φ' satisfies the hypothesis of Lemma 5.5 with $l = \epsilon$ and $r = \epsilon^{-1}$. Hence the statement. ◀

Two roads diverge now. Clearly the formulas Φ in Lemma 5.6 are going to define a piece of the domain of a piecewise linear homogeneous function, while the coefficients α_i define the function on that piece. We could decide to interpret our PLH functions in the domain \mathbb{Q}^* or we could decide to substitute a suitably small rational value of ϵ in the formal expression of $D_{\Phi,d}$ and map the problem to \mathbb{Q} . In the first case we have to *transfer* the known approaches for \mathbb{Q} to the new domain, in the second case we can *use* them (after having computed a suitable ϵ). It is not clear which road is the less traveled by. For reasons that will be discussed in Subsection 5.1 we take the one of transferring.

It is obvious that one can extend Definition 2.1 considering VCSPs whose cost functions take values in any totally ordered ring containing \mathbb{Q} , and in particular in \mathbb{Q}^* . We will need to establish the basics of such extended VCSPs. More precisely, we will need to prove Corollary 5.12 hereafter, that builds on a *fully combinatorial* algorithm (Theorem 5.11) due to Iwata and Orlin [11].

► **Definition 5.7.** Let R be a totally ordered commutative ring with unit. A problem over R can be solved in *fully combinatorial polynomial time* if there exists a polynomial-time (uniform) machine on R (see [1], Chapters 3-4) solving it by performing only additions and comparisons of elements in R as fundamental operations. We recall that a uniform machine on a totally ordered commutative ring with unity operates on strings of symbols that represent elements of an ordered commutative ring, rather than bits as in classical Turing machines. (Notice that in such a machine there are no machine-constants except 1.)

► **Definition 5.8.** A set function is a function ψ defined on the set 2^V , of subsets of a given set V .

► **Definition 5.9.** A set function $\psi: 2^V \rightarrow Q$ with values in a totally ordered Abelian group Q is submodular if for all $U, W \in 2^V$

$$\psi(U) + \psi(W) \geq \psi(U \cap W) + \psi(U \cup W).$$

► **Definition 5.10.** A collection \mathcal{C} of subsets of a given set Q is said to be a *ring family* if it is closed under union and intersection.

Equivalently, a ring family is a distributive sublattice of $\mathcal{P}(Q)$ with respect to union and intersection, notably every distributive lattice can be represented in this form (Birkhoff's representation theorem). Computationally, we represent a ring family following [16, Section 6]. Namely, fixed a representation for the elements of Q , the ring family \mathcal{C} is represented by the smallest set $M \subseteq Q$ in \mathcal{C} , and an oracle that given an element of $v \in Q$ returns the smallest $M_v \subset Q$ in \mathcal{C} such that $v \in M_v$. The construction of Section 6 in [16] proves that any algorithm capable of minimising submodular set functions can be used to minimise submodular set functions defined on a ring family represented in this way. Observe that this construction is fully combinatorial.

► **Theorem 5.11** (Iwata-Orlin [11] + Schrijver [16]). *There exists a fully combinatorial polynomial-time algorithm over \mathbb{Q} that*

- *taking as input a finite set $Q = \{1, \dots, n\}$ and a ring family, $\mathcal{C} \subseteq 2^Q$, represented as in [16, Section 6] (namely as above),*
- *having access to an oracle computing a submodular set-function $\psi: \mathcal{C} \rightarrow \mathbb{Q}$, computes an element $S \in \mathcal{C}$ such that $\psi(S) = \min_{A \in \mathcal{C}} \psi(A)$ in time bounded by a polynomial $p(n)$ in the size n of the domain.*

► **Corollary 5.12.** *Let R be a totally ordered commutative ring with unit (for instance \mathbb{Q}^*), there exists a fully combinatorial polynomial-time algorithm over R that*

- *taking as input a finite set $Q = \{1, \dots, n\}$ and a ring family, $\mathcal{C} \subseteq 2^Q$, represented as in Theorem 5.11,*
- *having access to an oracle computing a submodular set-function $\psi: \mathcal{C} \rightarrow R$, computes an element $S \in \mathcal{C}$ such that $\psi(S) = \min_{A \in \mathcal{C}} \psi(A)$ in time bounded by a polynomial $p(n)$ in the size n of the domain.*

Proof. Theorem 5.11 provides a fully combinatorial algorithm to minimise submodular functions that, over \mathbb{Q} , runs in polynomial time and computes a correct result. We claim that any such algorithm must be correct and run in polynomial time over R as well. To show this, we prove the following:

1. The algorithm terminates in time $p(n)$, where $p(n)$ is as in Theorem 5.11.
2. The output of the algorithm coincides with the minimum of ψ .

Let R_ψ denote the subgroup of the additive group $(R, +)$ generated by $\psi(\mathcal{C})$, and let $E_\psi := \{g_1, \dots, g_m\}$ be a set of free generators of R_ψ . For any tuple $r = (r_1, \dots, r_m) \in \mathbb{Q}^m$, we define a group homomorphism $h_r: R_\psi \rightarrow \mathbb{Q}$, by $h_r(g_i) = r_i$. Let $R_N := N \cdot (E_\psi \cup \{0\} \cup -E_\psi)$ be the subset of R consisting of the elements of the form $\pm x_1 \pm x_2 \dots \pm x_k$, with $k \leq N$, $x_1, x_2, \dots, x_k \in E_\psi$.

In general, the group homomorphisms h_r are not order preserving. We claim that for all N , there exists $r \in \mathbb{Q}^m$ such that $h_r|_{R_N}$ is order preserving. To see this, assume that no such tuple r exists. The inequalities denoting that $h_r|_{R_N}$ is order preserving are expressed by a finite linear program P in the variables r_1, \dots, r_m . By the assumption and Farkas' lemma there is a linear combination (with coefficients in \mathbb{Z}) of the inequalities of P which is contradictory. Therefore P is contradictory in any ordered ring, and, in particular, in R . However $r_i = g_i$, for all $i \in \{1, \dots, m\}$, is a valid solution of P in R .

Fix $N := \hat{N} \cdot 2^{p(n)}$, where \hat{N} is such that $\psi(S) \in R_{\hat{N}}$ for all $S \in \mathcal{C}$. For this N , let r be a tuple satisfying the claim. We run two parallel instances of the algorithm, one over R with input ψ , and the other in \mathbb{Q} with input $h_r \circ \psi$. We can prove that the two runs are *exactly parallel* for at least $p(n)$ steps, therefore, since the second run stops within these $p(n)$ steps, also the first one must do so. Formally, we prove, in a register machine model, that, at each step $i \leq p(n)$, if a register contains the value g in the first run, it must contain the value $h_r(g)$ in the second. This is easily established proving by induction on i that a value computed at step i must be in $R_{\hat{N} \cdot 2^i}$. Point 1 is thus established.

For point 2, let \min_R and $\min_{\mathbb{Q}}$ be the output of the algorithm over (ψ, R) and $(h_r \circ \psi, \mathbb{Q})$, respectively. The induction above shows, in particular, that $\min_{\mathbb{Q}} = h_r(\min_R)$. We know that $h_r(\min_R) = \min_{\mathbb{Q}} = h_r \circ \psi(S_0)$ for some S_0 and $h_r \circ \psi(S) \geq \min_{\mathbb{Q}} = h_r(\min_R)$ for each element S of \mathcal{C} . By our choice of N , the corresponding relations, $\min_R = \psi(S_0)$ and $\psi(S) \geq \min_R$ for each element S of \mathcal{C} , must hold in R . ◀

The following lemma is essentially contained in [7, Theorem 6.7], except that we replace the set of values by an arbitrary totally ordered commutative ring with unit R . To state the lemma properly, we need to observe that, given a submodular function f defined on Q^d , where $Q = \{1, \dots, n\}$, we can associate to it the following ring family $\mathcal{C}_f \subseteq \mathcal{P}(Q \times \{1, \dots, d\})$. For every $x = (x_1, \dots, x_d) \in Q^d$ define

$$C_x := \{(q, i) \mid q \in Q, q \leq x_i\} \subseteq Q \times \{1, \dots, d\}$$

then we let \mathcal{C}_f be the union of C_x for all x such that $f(x) < +\infty$.

► **Lemma 5.13.** *Let $R \supseteq \mathbb{Q}$ be a totally ordered ring. There exists a fully combinatorial polynomial-time algorithm over R that*

- taking as input a finite set $Q = \{1, \dots, n\}$ and an integer d ,
 - having access to an oracle computing a partial submodular $f: Q^d \rightarrow R$,
 - given the representation of \mathcal{C}_f as in Theorem 5.11,
- computes an $x \in Q^d$ such that $f(x)$ is minimal, in time polynomial in n and d .

Proof. The problem reduces to minimising a submodular set-function on the ring family \mathcal{C}_f , for the details see the proof of Theorem 6.7 in [7]. ◀

Proof of Theorem 5.1. Similarly to the proof of Theorem 4.3, we will use a sampling technique. Namely, given an instance I of VCSP(Γ), we will employ Lemma 5.6 to fix a finite structure Γ_I , of size (and also representation size) polynomial in $|V_I|$, having a subset \mathbb{Q}_I^* of $\mathbb{Q}_{-1,4}^*$ as domain, such that the variables V_I of I have an assignment in \mathbb{Q} having cost $\leq u_I$ if and only if they have one in \mathbb{Q}_I^* . Once we have Γ_I , we will conclude by Lemma 5.13.

12:14 Submodular Functions and VCSPs over Infinite Domains

The structure Γ_I obviously needs to have the same signature τ as Γ . For each function symbol $f \in \tau$ we consider a τ_0 -formula ϕ_f defining f^Γ and we let f^{Γ_I} be the function defined in \mathbb{Q}^* by the same formula. By Proposition 4.8 the choice of ϕ_f is immaterial. Remains to define the domain $\mathbb{Q}_I^* \subset \mathbb{Q}^*$.

By quantifier-elimination (Theorem 3.4), any piecewise linear homogeneous cost function $f: \mathbb{Q}^n \rightarrow \mathbb{Q} \cup \{+\infty\}$ can be written as

$$f(x_1, \dots, x_n) = \begin{cases} t_{f,1} & \text{if } \chi_{f,1} \\ \dots & \\ t_{f,m_f} & \text{if } \chi_{f,m_f} \\ +\infty & \text{otherwise} \end{cases}$$

where $t_{f,1}, \dots, t_{f,m_f}$ are τ_0 -terms, $\chi_{f,1}, \dots, \chi_{f,m_f}$ are conjunctions of atomic τ_0 -formulas with variables from $\{x_1, \dots, x_n\}$, and $\chi_{f,1}, \dots, \chi_{f,m_f}$ define disjoint subsets of \mathbb{Q}^n . We fix such a representation for each of the cost functions in Γ , and we collect all the atomic formulas appearing in every one of the conjunctions $\chi_{f,i}$, for $f \in \Gamma$ and $1 \leq i \leq m_f$, into the set Φ . Clearly Φ is finite and depends only on the fixed language Γ . Finally, $\mathbb{Q}_I^* := D_{\Phi, |V_I|}$ as defined in Lemma 5.6.

The size of \mathbb{Q}_I^* is clearly polynomial by simple inspection of the definition. Its representation has also polynomial size if the numbers are represented in binary, and, with this representation, the evaluation of f^{Γ_I} for $f \in \tau$ takes polynomial time.

Given an assignment $\alpha: V_I \rightarrow \mathbb{Q}_I^*$ of value $\leq u_I$ we have, a fortiori, an assignment $V_I \rightarrow \mathbb{Q}^*$ of value $\leq u_I$, hence, by the usual completeness of the first-order theory of ordered \mathbb{Q} -vector spaces, there is an assignment $V_I \rightarrow \mathbb{Q}$ with the same property.

Finally let $\beta: V_I \rightarrow \mathbb{Q}$ be an assignment having value $\leq u_I$. We need to find an assignment $\beta': V_I \rightarrow \mathbb{Q}_I^*$ with value $\leq u_I$. Let

$$\phi_I = \sum_{i=1}^m f_i(x_1^i, \dots, x_{\text{ar}(f_i)}^i)$$

(cf. Definition 2.1). For each $i \in \{1, \dots, m\}$ select the formula χ_i among $\chi_{f_i,1}, \dots, \chi_{f_i,m_{f_i}}$ that is satisfied by the assignment β . Clearly, the conjunction of atomic τ_0 -formulas $\chi := \bigwedge_{i=1}^m \chi_i$ is satisfiable. Moreover, ϕ_I restricted to the subset of $(\mathbb{Q}^*)^{|V_I|}$ where χ holds is obviously linear. Then we can apply Lemma 5.6, and we get an assignment β' whose values are in $D_{\chi, |V_I|}$ (where, by a slight abuse of notation, we wrote χ for the set of conjuncts of χ). We conclude observing that $D_{\chi, |V_I|} \subseteq D_{\Phi, |V_I|} = \mathbb{Q}_I^*$.

It remains to check that Lemma 5.13 applies to our situation. Clearly $R = \mathbb{Q}^*$, the function f is the objective function described by ϕ_I , and we let $n = |\mathbb{Q}_I^*|$ so that we identify Q with an enumeration of \mathbb{Q}_I^* in increasing order (which can be computed in polynomial time without obstacle). The oracle computing f is straightforward to implement since sums and comparisons in \mathbb{Q}^* merely reduce to the corresponding component-wise operations on the coefficients. The representation of the ring family \mathcal{C}_f requires a moment of attention. To construct the oracle, as well as to find the minimal element M , we need an algorithm that, given a variable $x \in V_I$ and a value $q \in \mathbb{Q}_I^*$, finds the component-wise minimal feasible assignment $\alpha_x: V_I \rightarrow \mathbb{Q}_I^*$ that gives to x a value $\geq q$ (which exists observing that the set of feasible assignments is min-closed). This algorithm is easy to construct observing that the feasibility problem is a min-closed CSP. We describe how to find M , the procedure for M_v is essentially the same.

Suppose that for each variable $x \in V_I$ we can find the smallest element $\beta(x) \in \mathbb{Q}_I^*$ such that there is a feasible assignment $\gamma_x: V_I \rightarrow \mathbb{Q}_I^*$ such that $\gamma_x(x) = \beta(x)$, then, by the min-closure, $\beta = \min_{x \in V_I} \gamma_x$ is the minimal assignment. To find $\beta(x)$ it is sufficient to solve the feasibility problem, using Theorem 4.3, adding a constraint $x \geq k$ for increasing values of $k \in \mathbb{Q}_I^*$. ◀

5.1 Why \mathbb{Q}^* ?

It might appear that in more than one occasion we chose to work in mathematically overcomplicated structures. For example, the algorithm for Theorem 5.1 merely manipulates points in $\mathbb{Q}_{-1,4}^*$, which is just \mathbb{Q}^6 with the lexicographic order, yet we went to the trouble of introducing the field of formal Laurent power series. More radically, one might observe that assigning a rational value to the formal variable ϵ small enough, we could have mapped the entire algorithm to \mathbb{Q} , thus dispensing with non-Archimedean extensions entirely. As we believe to owe to our reader an explanation for this, we better give three.

First, the idea of limiting our horizon to $\mathbb{Q}_{-1,4}^* \simeq \mathbb{Q}^6$ might seem a simplification, but, in practice, it makes things more complicated. For example, in several places we used the fact that \mathbb{Q}^* has a field structure to make proofs more direct and intuitive. Second, going for the most elementary exposition, namely choosing an ϵ small enough explicitly, would have completely obfuscated any idea in the arguments, which would have been converted in some unsightly bureaucracy of inequalities. Even computationally, mapping everything to \mathbb{Q} is tantamount as converting arrays of small integers into bignums by concatenation, hardly an improvement. Finally, the existence of an efficiently computable rational value of ϵ that works is not necessary for our method, even though, in this case, a posteriori, such an ϵ exists.

Our third, and most important, justification, is that we desire to present the approach used in this paper, which is quite generic, as much as the results. To this aim, it is convenient to express the underlying ideas in their natural language. For example, Corollary 5.12 is a completely black-boxed way to transfer combinatorial algorithms between domains that share some algebraic structure. We do not claim great originality in that observation, yet we believe that the method is interesting, and worthy of being presented in the cleanest form that we could devise.

6 Maximal Tractability

A *sublanguage* of a valued constraint language Γ is a valued constraint language that can be obtained from Γ by dropping some of the cost functions.

► **Definition 6.1.** Let \mathcal{V} be a class of valued constraint languages over a fixed domain D and let Γ be a language of \mathcal{V} . We say that Γ is *maximally tractable within \mathcal{V}* if

- VCSP(Γ') is polynomial time solvable for every finite sublanguage Γ' of Γ ; and
- for every valued constraint language Δ in \mathcal{V} properly containing Γ , there exists a finite sublanguage Δ' of Δ such that VCSP(Δ') is NP-hard.

Using [7, Theorem 6.7], it is easy to show the following. (See the appendix for details.)

► **Theorem 6.2.** *The valued constraint language consisting of all submodular PLH cost functions is maximally tractable within the class of PLH valued constraint languages.*

7 Conclusion and Outlook

We have presented a polynomial-time algorithm for submodular PLH cost functions over the rationals. In fact, our algorithm not only decides the feasibility problem and whether there exists a solution of cost at most u_I , but can also be adapted to efficiently compute the infimum of the cost of all solutions (which might be $-\infty$), and decides whether the infimum is attained. The modification is straightforward observing that the sample computed does not depend on the threshold u_I .

We also showed that submodular PLH cost functions are *maximally tractable* within the class of PLH cost functions. Such maximal tractability results are of particular importance for the more ambitious goal to classify the complexity of the VCSP for all classes of PLH cost functions: to prove a complexity dichotomy it suffices to identify *all* maximally tractable classes.

Another challenge is to extend our tractability result to the class of all submodular *piecewise linear* VCSPs. We believe that submodular piecewise linear VCSPs are in P, too. But note that already the structure $(\mathbb{Q}; 0, S, D)$ where $S := \{(x, y) \mid y = x + 1\}$ and $D := \{(x, y) \mid y = 2x\}$ (which has both min and max as a polymorphism) does *not* admit an efficient sampling algorithm (it is easy to see that for every $d \in \mathbb{N}$ every d -sample must have exponentially many vertices in d), so a different approach than the approach in this paper is needed.

References

- 1 Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer-Verlag, New York, 1998. With a foreword by Richard M. Karp.
- 2 Manuel Bodirsky and Martin Grohe. Non-dichotomies in constraint satisfaction complexity. In Luca Aceto, Ivan Damgard, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, editors, *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, pages 184–196. Springer Verlag, July 2008.
- 3 Manuel Bodirsky, Dugald Macpherson, and Johan Thapper. Constraint satisfaction tractability from semi-lattice operations on infinite sets. *Transaction of Computational Logic (ACM-TOCL)*, 14(4):1–30, 2013.
- 4 Manuel Bodirsky and Marcello Mamino. Tropically convex constraint satisfaction. *Theory of Computing Systems*, pages 1–29, 2017. An extended abstract of the paper appeared under the title “Max-Closed Semilinear Constraints” in the proceedings of CSR’16; preprint available under ArXiv:1506.04184.
- 5 Stephen P. Boyd and Lieven Vandenbergh. *Convex Optimization*. Cambridge University Press, 2004.
- 6 Andrei A. Bulatov. A dichotomy theorem for nonuniform csp. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330, 2017.
- 7 David A Cohen, Martin C Cooper, Peter G Jeavons, and Andrei A Krokhin. The complexity of soft constraint satisfaction. *Artificial Intelligence*, 170(11):983–1016, 2006.
- 8 Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.
- 9 Satoru Fujishige. *Submodular Functions and Optimization*. volume 58 of Annals of Discrete Mathematics. North-Holland, Amsterdam, 2005. 2nd edition.
- 10 M. Grötschel, L. Lovász, and L. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, Heidelberg, 1994. 2nd edition.

- 11 Satoru Iwata and James B. Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '09*, pages 1230–1237, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=1496770.1496903>.
- 12 Peter Jonsson, Fredrik Kuivinen, and Johan Thapper. Min CSP on four elements: Moving beyond submodularity. In *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, pages 438–453, 2011.
- 13 Vladimir Kolmogorov, Andrei A. Krokhin, and Michal Rolinek. The complexity of general-valued csp. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1246–1258, 2015.
- 14 Vladimir Kolmogorov, Johan Thapper, and Stanislav Zivny. The power of linear programming for general-valued csp. *SIAM J. Comput.*, 44(1):1–36, 2015.
- 15 Marcin Kozik and Joanna Ochremiak. Algebraic properties of valued constraint satisfaction problem. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 846–858, 2015.
- 16 Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000.
- 17 Lou van den Dries. *Tame topology and o-minimal structures*, volume 248 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1998. doi:10.1017/CB09780511525919.
- 18 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342, 2017.

A Appendix

A.1 Quantifier Elimination: Proof of Theorem 3.4

To prove Theorem 3.4 it suffices to prove the following lemma.

► **Lemma A.1.** *For every quantifier-free τ_0 -formula φ there exists a quantifier-free τ_0 -formula ψ such that $\exists x.\varphi$ is equivalent to ψ over \mathcal{L} .*

Proof. We define ψ in seven steps.

1. Rewrite φ , using De Morgan’s laws, in such a way that all the negations are applied to atomic formulas.
2. Replace
 - $\neg(s = t)$ by $s < t \vee t < s$, and
 - $\neg(s < t)$ by $t < s \vee s = t$,
 where s and t are τ_0 -terms.
3. Write φ in disjunctive normal form in such a way that each of the clauses is a conjunction of non-negated atomic τ_0 -formulas (this can be done by distributivity).
4. Observe that $\exists x \bigvee_i \bigwedge_j \chi_{i,j}$, where the $\chi_{i,j}$ are atomic τ_0 -formulas, is equivalent to $\bigvee_i \exists x \bigwedge_j \chi_{i,j}$. Therefore, it is sufficient to prove the lemma for $\varphi = \bigwedge_j \chi_j$ where the χ_j are atomic τ_0 -formulas. As explained above, we can assume without loss of generality that the χ_j are of the form \top , \perp , $x \sigma c$, $c \sigma x$, or $x \sigma cy$, for $c \in \mathbb{Q}$ and $\sigma \in \{<, =\}$. If χ_j equals \perp , then φ is equivalent to \perp and there is nothing to be shown. If χ_j equals \top then it can simply be removed from φ . If χ_j equals $x = c$ or $x = cy$ then replace every occurrence of x by $c \cdot 1$ or by $c \cdot y$, respectively. Then φ does not contain the variable x anymore and thus $\exists x.\varphi$ is equivalent to φ .

12:18 Submodular Functions and VCSPs over Infinite Domains

5. We are left with the case that all atomic τ_0 -formulas involving x are (strict) inequalities, that is, $\varphi = \bigwedge_i \chi_i \wedge \bigwedge_i \chi'_i \wedge \bigwedge_i \chi''_i$, where
- the χ_i are atomic formulas not containing x ,
 - the χ'_i are atomic formulas of the form $x > u_i$,
 - the χ''_i are atomic formulas of the form $x < v_i$.
- Then $\exists x.\varphi$ is equivalent to $\bigwedge_i \chi_i \wedge \bigwedge_{i,j}(u_i < v_j)$.

Each step of this procedure preserves the satisfying assignments for φ and the resulting formula is in the required form; this is obvious for all but the last step, and for the last step follows from the correctness of Fourier-Motzkin elimination for systems of linear inequalities. Therefore the procedure is correct. ◀

Proof (of Theorem 3.4). Let φ be a τ_0 -formula. We prove that it is equivalent to a quantifier-free τ_0 -formula by induction on the number n of quantifiers of φ . For $n = 1$ we have two cases:

- If φ is of the form $\exists x.\varphi'$ (with φ' quantifier-free) then, by Lemma A.1, it is equivalent to a quantifier-free τ_0 -formula ψ .
- If φ is of the form $\forall x.\varphi'$ (with φ' quantifier-free), then it is equivalent to $\neg\exists x.\neg\varphi'$. By Lemma A.1, $\exists x.\neg\varphi'$ is equivalent to a quantifier-free τ_0 -formula ψ . Therefore, φ is equivalent to the quantifier-free τ_0 -formula $\neg\psi$.

Now suppose that φ is of the form $Q_1x_1Q_2x_2\cdots Q_nx_n.\varphi'$ for $n \geq 2$ and $Q_1, \dots, Q_n \in \{\forall, \exists\}$, and suppose that the statement is true for τ_0 -formulas with at most $n - 1$ quantifiers. In particular, $Q_2x_2\cdots Q_nx_n.\varphi'$ is equivalent to a quantifier-free τ_0 -formula ψ . Therefore, φ is equivalent to $Q_1x_1.\psi$, that is, a τ_0 -formula with one quantifier that is equivalent to a quantifier-free τ_0 -formula, again by inductive hypothesis. ◀

A.2 Proof of Lemma 4.11 and Lemma 4.12

Proof of Lemma 4.11. Let $\gamma \leq \beta$ be maximal such that there are Ψ_1, Ψ_2, Ψ_3 with

$$\begin{aligned}\bar{\Phi} &= \{s_1 = s'_1, \dots, s_\alpha = s'_\alpha\} \cup \{t_1 \leq t'_1, \dots, t_\beta \leq t'_\beta\} \\ \Psi_1 &= \{s_1 = s'_1, \dots, s_\alpha = s'_\alpha\} \\ \Psi_2 &= \{t_1 = t'_1, \dots, t_\gamma = t'_\gamma\} \\ \Psi_3 &= \{t_{\gamma+1} \leq t'_{\gamma+1}, \dots, t_\beta \leq t'_\beta\},\end{aligned}$$

where s_i, s'_i, t_j, t'_j are τ_0 -terms for all i, j , and $\Psi_1 \cup \Psi_2 \cup \Psi_3$ is satisfiable in positive numbers. Clearly the space of positive solutions of $\Psi_1 \cup \Psi_2$ must be contained in that of Ψ_3 . In fact, by construction, they intersect: consider any straight line segment connecting a solution of $\Psi_1 \cup \Psi_2 \cup \Psi_3$ and a solution of $\Psi_1 \cup \Psi_2$ not satisfying Ψ_3 , on this segment there must be a solution of $\Psi_1 \cup \Psi_2 \cup \Psi_3$ lying on the boundary of one of the inequalities of Ψ_3 , contradicting the maximality of γ . By the last observation it suffices to prove that there is a solution of $\Psi_1 \cup \Psi_2$ taking values in $C_{\Phi, d}$. Put an edge between two variables x_i and x_j when they appear in the same formula of $\Psi_1 \cup \Psi_2$. For each connected component of the graph thus defined, either it contains at least one variable x_i such that there is a constraint of the form $h \cdot x_i = k \cdot 1$, or all constraints are of the form $h \cdot x_i = h' \cdot x_j$. In the first case assign $x_i = \frac{k}{h}$, in the second assign one of the variables x_i arbitrarily to 1, then, in any case, since the diameter of the connected component is $< d$, all variables in it are forced to take values in $C_{\Phi, d}$ by simple propagation of x_i . ◀

Proof of Lemma 4.12. First we fix a solution $x_i = a_i$ for $i = 1 \dots d$ of Φ . In general, some of the values a_i will be positive, some 0, and some negative: we look for a new solution $z_1 \dots z_d \in D_{\Phi,d}$ such that z_i is positive, respectively 0 or negative, if and only if a_i is.

To this aim we rewrite the formulas in Φ replacing each variable x_i with either y_i , or 0 (formally $0 \cdot 1$), or $-y_i$ (formally $-1 \cdot y_i$). We call Φ^+ the new set of formulas, which, by construction, is satisfiable in positive numbers $y_i = b_i$. To establish the lemma, it suffices to find a solution of Φ^+ taking values in $C_{\Phi,d}^*$.

By Lemma 4.11, we have an assignment $y_i = c_i$ of values $c_1 \dots c_d$ in $C_{\Phi^+,d} \subseteq C_{\Phi,d}$ that satisfies simultaneously all formulas ϕ with $\phi \in \Phi^+$. Let $-d \leq n_1 \dots n_d \leq d$ be integers such that for all i, j

$$\begin{aligned} n_i < n_j & \text{ if and only if } \frac{b_i}{c_i} < \frac{b_j}{c_j} \\ 0 < n_i & \text{ if and only if } 1 < \frac{b_i}{c_i} \\ n_i < 0 & \text{ if and only if } \frac{b_i}{c_i} < 1 \end{aligned}$$

Such numbers exist: simply sort the set $\{1\} \cup \{\frac{b_i}{c_i} \mid i = 1 \dots d\}$ and consider the positions in the sorted sequence counting from that of 1. We claim that the assignment $y_i = c_i + n_i c_i \epsilon \in \mathbb{Q}^*$ satisfies all formulas of Φ^+ . To check this, we consider the different cases for atomic formulas

- $k \cdot y_i < h \cdot y_j$: if $kc_i < hc_j$ this is obviously satisfied. Otherwise $kc_i = hc_j$, in this case k and h are positive and the constraint

$$kc_i + kn_i c_i \epsilon < hc_j + hn_j c_j \epsilon$$

is equivalent to $n_i < n_j$. This, in turn, is equivalent by construction to $\frac{b_i}{c_i} < \frac{b_j}{c_j}$ which we get by observing that $b_i hc_j = b_i kc_i < b_j hc_i$.

- $k \cdot y_i = h \cdot y_j$: obviously $kb_i = hb_j$ and $kc_i = hc_j$, therefore $\frac{b_i}{c_i} = \frac{b_j}{c_j}$, and, as a consequence, also $n_i = n_j$ from which the statement.
- $k \cdot 1 < h \cdot y_j$: similarly to the first case, if $k < hc_j$ this is immediate. Otherwise $k = hc_j$, so k and h are positive, the constraint

$$k \cdot 1 < hc_j + hn_j c_j \epsilon$$

is equivalent to $0 < n_j$, in other words $1 < \frac{b_j}{c_j}$, which follows observing that $hc_j = k < hb_j$.

- $k \cdot y_i < h \cdot 1$: as the case above.
- $k \cdot 1 = h \cdot y_j$: obviously $k \cdot 1 = hb_j = hc_j$, therefore $\frac{b_j}{c_j} = 1$, so $n_j = 0$ and the case follows.
- $k \cdot y_i = h \cdot 1$: as the case above. ◀

A.3 Proof of Lemma 5.4 and Lemma 5.5

Proof of Lemma 5.4. As in the proof of Lemma 4.11 (to which we direct the reader for many details) we take a maximal $\gamma \leq \beta$ such that there are Ψ_1, Ψ_2, Ψ_3 with

$$\begin{aligned} \bar{\Phi} &= \{s_1 = s'_1, \dots, s_\alpha = s'_\alpha\} \cup \{t_1 \leq t'_1, \dots, t_\beta \leq t'_\beta\} \\ \Psi_1 &= \{s_1 = s'_1, \dots, s_\alpha = s'_\alpha\} \\ \Psi_2 &= \{t_1 = t'_1, \dots, t_\gamma = t'_\gamma\} \\ \Psi_3 &= \{t_{\gamma+1} \leq t'_{\gamma+1}, \dots, t_\beta \leq t'_\beta\} \end{aligned}$$

and $\Psi_1 \cup \Psi_2 \cup \Psi_3$ is satisfiable by an assignment with $\sum_i \alpha_i x_i < u$. As in the proof of Lemma 4.11 the set of solutions of $\Psi_1 \cup \Psi_2$ satisfying $\sum_i \alpha_i x_i < u$ is contained in the solutions of Ψ_3 . So, here too, it suffices to show that there is a solution of $\Psi_1 \cup \Psi_2$ with $\sum_i \alpha_i x_i < u$

taking values in $C_{\Phi,d}$. The proof of Lemma 4.11 shows that there is a solution of $\Psi_1 \cup \Psi_2$ taking values in $C_{\Phi,d}$ without necessarily meeting the requirement that $\sum_i \alpha_i x_i < u$. We will prove that, in fact, any such solution meets the additional constraint.

Let $x_i = a_i, b_i$ be two distinct satisfying assignments for $\Psi_1 \cup \Psi_2$ such that $\sum_i \alpha_i a_i < u$ and $\sum_i \alpha_i b_i \geq u$. We know that the first exists, and we assume the second towards a contradiction. The two assignments must differ, so, without loss of generality $a_1 \neq b_1$. For $t \in \mathbb{Q}^*$, with $t \geq 0$, define the assignment $x_i(t) = (1+t)a_i - tb_i$. Since all constraints in $\Psi_1 \cup \Psi_2$ are equalities, it is clear that the new assignment $x_i(t)$ satisfies $\Psi_1 \cup \Psi_2$ for all $t \in \mathbb{Q}^*$. Moreover, if $t \geq 0$

$$\sum_i \alpha_i x_i(t) \leq \sum_i \alpha_i a_i - t \left(\sum_i \alpha_i b_i - \sum_i \alpha_i a_i \right) < u$$

Let $t = \frac{2r}{|b_1 - a_1|}$. Then

$$x_1(t) = a_1 + \frac{2r}{|b_1 - a_1|} (a_1 - b_1)$$

is either $\geq 2r$ or < 0 depending on the sign of $(a_1 - b_1)$. In either case we have a solution $x_i = x_i(t)$ of $\Psi_1 \cup \Psi_2$ satisfying $\sum_i \alpha_i x_i(t) < u$, which must therefore be a solution of Φ , that does not satisfy $0 < x_i \leq r$. \blacktriangleleft

Proof of Lemma 5.5. We consider two cases: either all satisfying assignments satisfy the inequality $\sum_i \alpha_i x_i \geq u$ or there is a satisfying assignment $(x_1 \dots x_d)$ for Φ such that $\sum_i \alpha_i x_i < u$.

In the first case, we claim that all satisfying assignments, in fact, satisfy $\sum_i \alpha_i x_i = u$. In fact, assume that $x_i = a_i, b_i$ are two satisfying assignments such that $\sum_i \alpha_i a_i = u$ and $v := \sum_i \alpha_i b_i > u$. As in the proof of Lemma 5.4, consider assignments of the form $x_i(t) = (1+t)a_i - tb_i$ for $t \in \mathbb{Q}^*$. Clearly $\sum_i \alpha_i x_i(t) = u - t(v - u) < u$ for all $t > 0$. As in Lemma 5.4, the new assignment must satisfy all equality constraints in Φ . Each inequality constraint implies a strict inequality on t (remember that Φ only has strict inequalities). Since all of these must be satisfied by $t = 0$, there is an open interval of acceptable values of t around 0, and, in particular, an acceptable $t > 0$. Our claim is thus established. Therefore, in this case, it suffices to find any satisfying assignment for Φ taking values in $C_{\Phi,d}^*$. The assignment is now constructed as in the proof of Lemma 4.12, replacing the formal symbol ϵ in that proof by ϵ^3 . Namely take a satisfying assignment $x_i = b_i$ for Φ , and, by Lemma 5.4, one satisfying assignment $x_i = c_i$ for $\bar{\Phi}$ taking values in $C_{\Phi,d}$. Observe that the hypothesis that all solutions of Φ satisfy $l < x_i$ for all i is used here to ensure that all solutions of $\bar{\Phi}$ assign positive values to the variables, which is required by Lemma 5.4. Let $-d \leq n_1 \dots n_d \leq d$ be integers such that for all i, j

$$\begin{aligned} n_i < n_j & \text{ if and only if } \frac{b_i}{c_i} < \frac{b_j}{c_j} \\ 0 < n_i & \text{ if and only if } 1 < \frac{b_i}{c_i} \\ n_i < 0 & \text{ if and only if } \frac{b_i}{c_i} < 1 \end{aligned}$$

The assignment $y_i = c_i + n_i c_i \epsilon^3$ can be seen to satisfy all formulas of Φ by the same check as in the proof of Lemma 4.12. Observe that we have to replace ϵ in Lemma 4.12 by ϵ^3 here, so that $\mathbb{Q}_{-1,1}^* \cap \epsilon^3 \mathbb{Q}_{-1,1}^* = \emptyset$.

For the second case, fix a satisfying assignment $x_i = b_i$. By Lemma 5.4 there is an assignment $x_i = c_i \in C_{\Phi,d}$ such that $\sum_i \alpha_i c_i < u$ and this assignment satisfies $\bar{\phi}$ for all $\phi \in \Phi$.

From these two assignments construct the numbers n_i and then the assignment $y_i = c_i + n_i c_i \epsilon^3$ as before. For the same reason it is clear that the new assignment satisfies Φ . To conclude that $\sum_i \alpha_i y_i < u$ we write

$$\sum_i \alpha_i y_i = \sum_i \alpha_i c_i + \epsilon^3 \sum_i \alpha_i n_i c_i < u$$

because the first summand is in $\mathbb{Q}_{-1,1}^*$ and $< u$, therefore the second summand is neglected in the lexicographical order. \blacktriangleleft

A.4 Proof of the maximal tractability

In this appendix we prove Theorem 6.2. We will make use of the following result.

► **Theorem A.2** (Cohen-Cooper-Jeavons-Krokhin, [7], Theorem 6.7). *Let D be a finite totally ordered set. Then the valued constraint language consisting of all submodular cost functions over D is maximally tractable within the class of all valued constraint languages over D .*

We show that the class of submodular piecewise linear homogeneous languages is maximally tractable within the class of PLH valued constraint languages.

► **Definition A.3.** Given a finite set $D \subset \mathbb{Q}$, we define the partial function $\chi_D: \mathbb{Q} \rightarrow \mathbb{Q}$ by

$$\chi_D(x) = \begin{cases} 0 & x \in D \\ +\infty & x \in \mathbb{Q} \setminus D. \end{cases}$$

For every finite set $D \subset \mathbb{Q}$, the cost function χ_D is submodular and PLH.

► **Definition A.4.** Given a finite domain $D \subset \mathbb{Q}$ and a partial function $f: D^n \rightarrow \mathbb{Q}$ we define the *canonical extension* of f as $\hat{f}: \mathbb{Q}^n \rightarrow \mathbb{Q}$, by

$$\hat{f}(x) = \begin{cases} f(x) & x \in D^n \\ +\infty & \text{otherwise.} \end{cases}$$

Note that the canonical extension of a submodular function over a finite domain is submodular and PLH.

Proof of Theorem 6.2. Polynomial-time tractability of the VCSP for finite sets of submodular PLH cost functions has been shown in Theorem 5.1.

Now suppose that f is a cost function over \mathbb{Q} that is not submodular, i.e., there exists a couple of points, $a := (a_1, \dots, a_n)$, $b := (b_1, \dots, b_n) \in \mathbb{Q}^n$ such that

$$f(a) + f(b) < f(\min(a, b)) + f(\max(a, b)).$$

Let Γ_D be the language of all submodular functions on

$$D := \{a_1, \dots, a_n, b_1, \dots, b_n\} \subset \mathbb{Q}.$$

Notice that $f|_D$ is not submodular, for our choice of D . Therefore, by Theorem A.2, there exists a finite language $\Gamma'_D \subset \Gamma_D$ such that $\text{VCSP}(\Gamma'_D \cup \{f|_D\})$ is NP-hard.

We define the finite submodular PHL language Γ' by replacing every cost function g in Γ'_D by its canonical extension \hat{g} . Then $\Gamma' \cup \{f, \chi_D\}$, where χ_D is defined as in Definition A.3, has an NP-hard VCSP. Indeed, for every instance I of $\text{VCSP}(\Gamma'_D \cup \{f|_D\})$, we define an instance J of $\text{VCSP}(\Gamma' \cup \{f, \chi_D\})$ in the following way:

12:22 Submodular Functions and VCSPs over Infinite Domains

- replace every function symbol g in ϕ_I by the symbol for its canonical extension,
- replace the function symbol for $f|_D$ in ϕ_I by f , and
- add to J_ϕ the summand $\chi_D(v)$ for every variable $v \in V_I$.

Because of the terms involving χ_D , the infimum of ϕ_J is smaller than $+\infty$ if, and only if, it is attained in a point having coordinates in D . Therefore, the infimum of ϕ_J coincides with the infimum of ϕ_I . Since J is computable in polynomial-time from I , the NP-hardness of $\text{VCSP}(\Gamma' \cup \{f, \chi_D\})$ follows from the NP-hardness of $\text{VCSP}(\Gamma' \cup \{f|_D\})$. ◀

Graphical Conjunctive Queries

Filippo Bonchi

University of Pisa, Italy

Jens Seeber

IMT School for Advanced Studies Lucca, Italy

Paweł Sobociński

University of Southampton, UK

Abstract

The Calculus of Conjunctive Queries (CCQ) has foundational status in database theory. A celebrated theorem of Chandra and Merlin states that CCQ query inclusion is decidable. Its proof transforms logical formulas to graphs: each query has a *natural model* – a kind of *graph* – and query inclusion reduces to the existence of a graph homomorphism between natural models.

We introduce the diagrammatic language *Graphical Conjunctive Queries* (GCQ) and show that it has the same expressivity as CCQ. GCQ terms are *string diagrams*, and their algebraic structure allows us to derive a sound and complete axiomatisation of query inclusion, which turns out to be exactly Carboni and Walters’ notion of *cartesian bicategory of relations*. Our completeness proof exploits the combinatorial nature of string diagrams as (certain cospans of) hypergraphs: Chandra and Merlin’s insights inspire a theorem that relates such cospans with spans. Completeness and decidability of the (in)equational theory of GCQ follow as a corollary. Categorically speaking, our contribution is a model-theoretic completeness theorem of free cartesian bicategories (on a relational signature) for the category of sets and relations.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases conjunctive query inclusion, string diagrams, cartesian bicategories

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.13

1 Introduction

Conjunctive queries (CCQ) are first-order logic formulas that use only relation symbols, equality, truth, conjunction, and existential quantification. They are a kernel language of queries to relational databases and are the foundations of several languages: they are select-project-join queries in relational algebra [16], or select-from-where queries in SQL [13]. While expressive enough to encompass queries of practical interest, they admit algorithmic analysis: in [14], Chandra and Merlin showed that the problem of *query inclusion* is NP-complete.

For an example of query inclusion in action, consider formulas

$$\phi = \exists z_0 : (x_0 = x_1) \wedge R(x_0, z_0) \text{ and } \psi = \exists z_0, z_1 : R(x_0, z_0) \wedge R(x_1, z_0) \wedge R(x_0, z_1) \wedge R(x_1, z_1),$$

with free variables x_0, x_1 . Irrespective of model, and thus the interpretation of the relation symbol R , every free variable assignment satisfying ϕ satisfies ψ : i.e. ϕ is included in ψ .

Chandra and Merlin’s insight involves an elegant reduction to graph theory, namely the existence of a hypergraph homomorphism from a graphical encoding of ψ to that of ϕ . Below on the left we give a graphical rendering of ψ and ϕ , respectively: vertices represent variables, while edges are labelled with relation symbols. The dotted connections are not, strictly speaking, a part of the underlying hypergraphs. They constitute an *interface*: a mapping



© Filippo Bonchi, Jens Seeber, and Paweł Sobociński;
licensed under Creative Commons License CC-BY

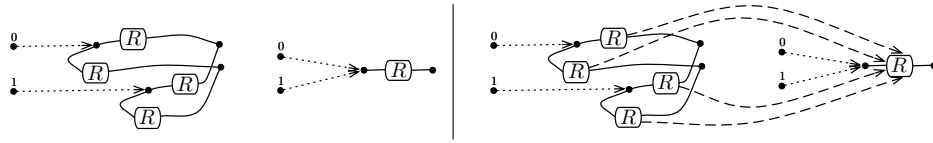
27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 13; pp. 13:1–13:23

Leibniz International Proceedings in Informatics



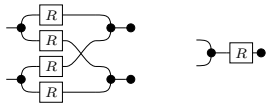
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



from the free variables $\{x_0, x_1\}$ to the vertices. The aforementioned query inclusion is witnessed by an interface-preserving hypergraph homomorphism, displayed above on the right. In category-theoretic terms, hypergraphs-with-interfaces are *discrete cospans*, and the homomorphisms are *cospan homomorphisms*.

In previous work [5], the first and third authors with Gadducci, Kissinger and Zanasi showed that such cospans characterise an important family of *string diagrams* – i.e. diagrammatic representations of the arrows of monoidal categories – namely those equipped with an algebraic structure known as a special Frobenius algebra. This motivated us to study the connection between this fashionable algebraic structure – which has been used in fields as diverse as quantum computing [1, 17, 30, 25], concurrency theory [7, 8, 10, 9], control theory [6, 3] and linguistics [31] – and conjunctive queries.

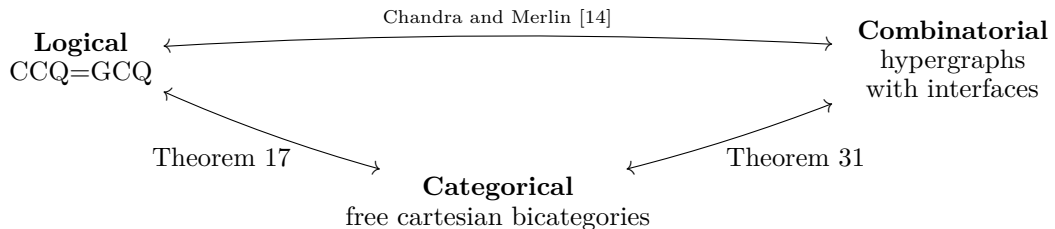
We introduce the logic of Graphical Conjunctive Queries (GCQ). Although superficially unlike CCQ, we show that it is equally expressive. Its syntax lends itself to string-diagrammatic representation and diagrammatic reasoning respects the underlying logical semantics. GCQ string diagrams for ψ and ϕ are drawn below. Note that, while GCQ syntax does not have variables, the concept of CCQ free variable is mirrored by “dangling” wires in diagrams.



While interesting in its own right as an example of a string-diagrammatic representation of a logical language – which has itself become a topic of recent interest [21] – GCQ comes into its own when reasoning about query inclusion, which is characterised by the laws of *cartesian bicategories*. This important categorical structure was introduced by Carboni and Walters [12] who were, in fact, aware of the logical interpretation, mentioning it in passing without giving the details. Our definition of GCQ, its expressivity, and *soundness* of the laws of cartesian bicategories w.r.t. query inclusion is testament to the depth of their insights.

The main contribution of our work is the *completeness* of the laws of cartesian bicategories for query inclusion (Theorem 17).

As a side result, we obtain a categorical understanding of the proof by Chandra and Merlin. This uncovers a beautiful triangle relating logical, combinatorial and categorical structures, similar to the Curry-Howard-Lambek correspondence relating intuitionistic propositional logic, λ -calculus and free cartesian closed categories.



The rightmost side of the triangle (Theorem 31) provides a combinatorial characterisation of free cartesian bicategories as discrete cospans of hypergraphs, with the Chandra and

Merlin ordering: the existence of a cospan homomorphism in the opposite direction. This result can also be regarded as an extension of the aforementioned [5] to an enriched setting. The fact that the Chandra and Merlin ordering is not antisymmetric forces us to consider preorder-enrichment as opposed to the usual [12] poset-enrichment of cartesian bicategories.¹

The step from posets to preorders is actually beneficial: it provides a one-to-one correspondence between hypergraphs and models which we see as functors, following the tradition of categorical logic. The model corresponding to a hypergraph G is exactly the (contravariant) Hom-functor represented by G . By a Yoneda-like argument, we obtain a “preorder-enriched analogue” of Theorem 17 (Theorem 37). With this result, proving Theorem 17 reduces to descending from the preorder-enriched setting down to poset-enrichment.

Working with both poset- and preorder-enriched categories means that there is a relatively large number of categories at play. We give a summary of the most important ones in the table below, together with pointers to their definitions. The remainder of this introduction is a roadmap for the paper, focussing on the roles played by the categories mentioned below.

	preordered	posetal
free categories	$\mathbb{CB}_{\Sigma}^{\leq}$ (Def 29)	\mathbb{CB}_{Σ} (Def 21)
semantic domains for the logic	$\mathbf{Span}^{\leq} \mathbf{Set}$ (Def 33)	\mathbf{Rel} (Ex 20)
combinatorial structures	$\mathbf{Csp}^{\leq} \mathbf{FHyp}_{\Sigma}$ (Def 26)	-

We begin by justifying the “equation” $\mathbf{CCQ} = \mathbf{GCQ}$ in the triangle above: we recall \mathbf{CCQ} and introduce \mathbf{GCQ} in Sections 2 and 3, respectively, and show that they have the same expressivity. We explore the algebraic structure of \mathbf{GCQ} in Sections 4 and 5, which – as we previously mentioned – is exactly that of cartesian bicategories. As instances of these, we introduce \mathbb{CB}_{Σ} , the free cartesian bicategory, and \mathbf{Rel} , the category of sets and relations.

In Section 6 we introduce *preordered* cartesian bicategories (the free one denoted by $\mathbb{CB}_{\Sigma}^{\leq}$) and the category of discrete cospans of hypergraphs with the Chandra and Merlin preorder, denoted by $\mathbf{Csp}^{\leq} \mathbf{FHyp}_{\Sigma}$. Theorem 31 states that these two are isomorphic.

Theorem 37 is proved in Section 7. Rather than \mathbf{Rel} , the preordered setting calls for models in $\mathbf{Span}^{\leq} \mathbf{Set}$, the preordered cartesian bicategory of spans of sets. In Section 8, we explain the passage from preorders to posets, completing the proof of Theorem 17.

We delay a discussion of the ramification of our work, a necessarily short and cursory account – due to space restrictions – of the considerable related work, and directions for future work to Section 9. We conclude with the observation that (i) the diagrammatic language for formulas, (ii) the semantics, e.g. of composition of diagrams – what we understand in modern terms as the combination of conjunction and existential quantification – and (iii) the use of diagrammatic reasoning as a powerful method of logical reasoning actually go back to the pre-Frege work of the 19th century American polymath CS Peirce on *existential graphs*. Interestingly, it is only recently (see, e.g. [29]) that this work has been receiving the attention that it richly deserves.

Preliminaries. We assume familiarity with basic categorical concepts, in particular symmetric monoidal, ordered and preordered categories. We do not assume familiarity with cartesian bicategories: the acquainted reader should note that what we call “cartesian bicategories”

¹ While cartesian bicategories were later generalised [11] to a bona fide higher-dimensional setting, our preorder-enriched variant seems to be an interesting stop along the way.

are “cartesian bicategories of relations” in [12]. A *prop* is a symmetric strict monoidal category where objects are natural numbers, and the monoidal product on objects is addition $m \oplus n := m + n$. Due to space restrictions, most proofs are in the Appendix.

2 Calculus of Conjunctive Queries

Assume a set Σ of relation symbols with arity function $ar : \Sigma \rightarrow \mathbb{N}$ and a countable set $Var = \{x_i \mid i \in \mathbb{N}\}$ of variables. The grammar for the calculus of conjunctive queries is:

$$\Phi ::= \top \mid \Phi \wedge \Phi \mid x_i = x_j \mid R(\vec{x}) \mid \exists x. \Phi \quad (\text{CCQ})$$

where $R \in \Sigma$, $ar(R) = n$, and \vec{x} is a list of length n of variables from Var . We assume the standard bound variable conventions and some basic metatheory of formulas; in particular we write $\phi[\vec{x}/\vec{y}]$, where \vec{x}, \vec{y} are variable lists of equal length, for the simultaneous substitution of variables from \vec{x} for variables in \vec{y} . We write $\vec{x}_{[m,n]}$, where $m \leq n$, for the list of variables x_m, x_{m+1}, \dots, x_n . Given a formula ϕ , $fv(\phi)$ is the set of its free variables.

The semantics of (CCQ) formulas is standard and inherited from first order logic.

► **Definition 1.** A model $\mathcal{M} = (X, \rho)$ is a set X and, for each $R \in \Sigma$, a set $\rho(R) \subseteq X^{ar(R)}$. Given a model $\mathcal{M} = (X, \rho)$, the semantics $\llbracket \phi \rrbracket_{\mathcal{M}}$ is the set of all assignments of elements from X to $fv(\phi)$ that makes it evaluate to truth, given the usual propositional interpretation.

In order to facilitate a principled definition of the semantics (Definition 3) and to serve the needs of our diagrammatic approach, we will need to take a closer look at free variables. To this end, we give an alternative, sorted presentation of (CCQ) that features explicit free variable management. As we shall see, the system of judgments below will allow us to derive $n \vdash \phi$ where $n \in \mathbb{N}$, whenever ϕ is a formula of CCQ and $fv(\phi) \subseteq \{x_0, \dots, x_{n-1}\}$.

$$\frac{}{0 \vdash \top} (\top) \quad \frac{R \in \Sigma \quad ar(R) = n}{n \vdash R(x_0, \dots, x_{n-1})} (\Sigma) \quad \frac{n \vdash \phi}{n-1 \vdash \exists x_{n-1}. \phi} (\exists)$$

$$\frac{}{2 \vdash x_0 = x_1} (=) \quad \frac{m \vdash \phi \quad n \vdash \psi}{m+n \vdash \phi \wedge (\psi[\vec{x}_{[m, m+n-1]}/\vec{x}_{[0, x_{n-1}]})} (\wedge)$$

Note that the above are restrictive: e.g. (\wedge) enforces disjoint sets of variables, and (\exists) allows quantification only over the last variable. To overcome these limitations we include three structural rules that allow us to manipulate (swap, identify, and introduce) free variables.

$$\frac{n \vdash \phi \quad (0 \leq k < n-1)}{n \vdash \phi[x_{k+1}, x_k/x_k, x_{k+1}]} (\text{Sw}_{n,k}) \quad \frac{n \vdash \phi}{n-1 \vdash \phi[x_{n-2}/x_{n-1}]} (\text{Id}_n) \quad \frac{n \vdash \phi}{n+1 \vdash \phi} (\text{Nu}_n)$$

Rule *Sw* allows us to swap two free variables. Alone, *Id* identifies the final and the penultimate free variable; used together with *Sw* it allows for the identification of any two. Finally, *Nu* introduces a free variable. The eight suffice for any CCQ formula, in the following sense:

► **Proposition 2.** ϕ is a formula derived from (CCQ) with $fv(\phi) \subseteq \{x_0, \dots, x_{n-1}\}$ iff $n \vdash \phi$.

We use the sorted presentation to define the semantics.

► **Definition 3.** Given a model $\mathcal{M} = (X, \rho)$, the semantics of $n \vdash \phi$ is a set of tuples $\llbracket n \vdash \phi \rrbracket_{\mathcal{M}} \subseteq X^n$. We define it in Figure 1 by recursion on the derivation of $n \vdash \phi$.

Finally, we define the concepts that are of central interest: *query equivalence* and *inclusion*.

► **Definition 4.** Given $n \vdash \phi$ and $n \vdash \psi$, we say that ϕ and ψ are equivalent and write $\phi \equiv \psi$ if for all models \mathcal{M} we have $\llbracket n \vdash \phi \rrbracket_{\mathcal{M}} = \llbracket n \vdash \psi \rrbracket_{\mathcal{M}}$. We write $\phi \leq \psi$ when, for all \mathcal{M} , $\llbracket n \vdash \phi \rrbracket_{\mathcal{M}} \subseteq \llbracket n \vdash \psi \rrbracket_{\mathcal{M}}$. Clearly $\phi \leq \psi$ and $\psi \leq \phi$ implies $\phi \equiv \psi$.

$$\begin{aligned}
 \llbracket 0 \vdash \top \rrbracket_{\mathcal{M}} &= \{\bullet\} \quad (\top) & (\vec{u}, v, w, \vec{x}) \in \llbracket n \vdash \phi[x_{k+1}, x_k/x_k, x_{k+1}] \rrbracket_{\mathcal{M}} &\Leftrightarrow (\vec{u}, w, v, \vec{x}) \in \llbracket n \vdash \phi \rrbracket_{\mathcal{M}} \quad (\text{Sw}_{n,k}) \\
 \llbracket n \vdash R(x_0, \dots, x_{n-1}) \rrbracket_{\mathcal{M}} &= \rho(R) \quad (\Sigma) & (\vec{v}, w) \in \llbracket n-1 \vdash \phi[x_{n-2}/x_{n-1}] \rrbracket_{\mathcal{M}} &\Leftrightarrow (\vec{v}, w, w) \in \llbracket n \vdash \phi \rrbracket_{\mathcal{M}} \quad (\text{Id}_n) \\
 \llbracket 2 \vdash x_0 = x_1 \rrbracket_{\mathcal{M}} &= \{(v, v) \mid v \in X\} \quad (=) & \vec{v} \in \llbracket n-1 \vdash \exists x_{n-1}. \phi \rrbracket_{\mathcal{M}} &\Leftrightarrow \exists w \in X. (\vec{v}, w) \in \llbracket n \vdash \phi \rrbracket_{\mathcal{M}} \quad (\exists) \\
 \llbracket n+1 \vdash \phi \rrbracket_{\mathcal{M}} &= \llbracket n \vdash \phi \rrbracket_{\mathcal{M}} \times X \quad (\text{Nu}_n) & \llbracket m+n \vdash \phi \wedge (\psi[\dots]) \rrbracket_{\mathcal{M}} &= \llbracket m \vdash \phi \rrbracket_{\mathcal{M}} \times \llbracket n \vdash \psi \rrbracket_{\mathcal{M}} \quad (\wedge)
 \end{aligned}$$

■ **Figure 1** Semantics of CCQ for a model $\mathcal{M} = (X, \rho)$. We write \bullet for the unique element of X^0 .

$$\frac{}{\dashv\!\!\!\dashv : (1, 2)} \quad \frac{}{\bullet : (1, 0)} \quad \frac{R \in \Sigma_{n,m}}{R : (n, m)} \quad \frac{}{\dashv\!\!\!\dashv : (2, 1)} \quad \frac{}{\bullet : (0, 1)} \quad \frac{}{\boxed{} : (0, 0)} \quad \frac{}{\dashv\!\!\!\dashv : (1, 1)} \quad \frac{}{\bowtie : (2, 2)} \quad \frac{c : (n, z) \quad d : (z, m)}{c;d : (n, m)} \quad \frac{c : (n, m) \quad d : (p, q)}{c \oplus d : (n+p, m+q)}$$

■ **Figure 2** Sort inference rules.

3 Graphical conjunctive queries

We introduce an alternative logic, called Graphical Conjunctive Queries (GCQ). GCQ and CCQ are – superficially – quite different. Nevertheless, in Propositions 9 and 10 we show that they have the same expressive power. The grammar of GCQ formulas is given below.

$$c ::= \dashv\!\!\!\dashv \mid \bullet \mid \dashv\!\!\!\dashv \mid \dashv\!\!\!\dashv \mid \bullet \mid \boxed{} \mid \dashv\!\!\!\dashv \mid \bowtie \mid c \oplus c \mid c ; c \mid R \quad (\text{GCQ})$$

GCQ syntax is a radical departure from (CCQ). Rather than use CCQ’s existential quantification and conjunction, GCQ uses the operations of monoidal categories: composition and monoidal product. There are no variables, thus no assumptions of their countable supply, nor any associated metatheory of capture-avoiding substitution.

The price is a simple sorting discipline. A *sort* is a pair (n, m) , with $n, m \in \mathbb{N}$. We consider only terms sortable according to Figure 2. There and in (GCQ), R ranges over the symbols of a *monoidal* signature Σ , a set of relation symbols equipped with both an arity and a *coarity*: $\Sigma_{n,m}$ consists of the symbols in Σ with arity n and coarity m . A GCQ signature plays a similar role to relation symbols in CCQ: we abuse notation for this reason. A simple induction shows sort uniqueness: if $c : (n, m)$ and $c : (n', m')$ then $n = n'$ and $m = m'$.

In (GCQ) we used a graphical rendering of GCQ constants. Indeed, we will not write terms of GCQ as formulas, but instead represent them as 2-dimensional diagrams. The justification for this is twofold: the diagrammatic conventions introduced in this section mean that a diagram is a readable, faithful and unambiguous representation of a sorted (GCQ) term. More importantly, our characterisation of query inclusion in subsequent sections consists of intuitive topological deformations of the diagrammatic representations of formulas.

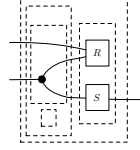
A GCQ term $c : (n, m)$ is drawn as a diagram with n “dangling wires” on the left, and m on the right. Roughly speaking, dangling wires are GCQ’s answer to the free variables of CCQ. Composing $(;)$ means connecting diagrams in series and tensoring means stacking. The shorthand $\overset{m}{\parallel}$ stands for m wires in parallel. The box $\overset{n}{\square}_R^m$ stands for a relation symbol $R \in \Sigma_{n,m}$. Thus, given $c : (n, m)$, $c' : (m, k)$, $c ; c' : (n, k)$

is drawn $\overset{n}{\parallel} \boxed{c} \overset{m}{\parallel} \boxed{c'} \overset{k}{\parallel}$, and given $d : (p, q)$, $c \oplus d : (n+p, m+q)$ is drawn $\overset{n}{\parallel} \begin{array}{|c|} \hline \boxed{c} \\ \hline \boxed{d} \\ \hline \end{array} \overset{m}{\parallel} \overset{q}{\parallel}$.

$$\begin{aligned}
 \llbracket \text{---} \circlearrowleft \rrbracket_{\mathcal{M}} &= \{(x, \binom{x}{z}) \mid x \in X\} & \llbracket \text{---} \bullet \rrbracket_{\mathcal{M}} &= \{(x, \bullet) \mid x \in X\} & \llbracket c \oplus d \rrbracket_{\mathcal{M}} &= \llbracket c \rrbracket_{\mathcal{M}} \oplus \llbracket d \rrbracket_{\mathcal{M}} & \llbracket \text{---} \square \rrbracket_{\mathcal{M}} &= \{(\bullet, \bullet)\} \\
 \llbracket \text{---} \circlearrowright \rrbracket_{\mathcal{M}} &= \{(\binom{x}{z}, x) \mid x \in X\} & \llbracket \bullet \text{---} \rrbracket_{\mathcal{M}} &= \{(\bullet, x) \mid x \in X\} & \llbracket c ; d \rrbracket_{\mathcal{M}} &= \llbracket c \rrbracket_{\mathcal{M}} ; \llbracket d \rrbracket_{\mathcal{M}} & \llbracket R \rrbracket_{\mathcal{M}} &= \rho(R) \\
 \llbracket \text{---} \bowtie \rrbracket_{\mathcal{M}} &= \{(\binom{x}{y}, \binom{y}{z}) \mid x, y \in X\} & \llbracket \text{---} \rrbracket_{\mathcal{M}} &= \{(x, x) \mid x \in X\}
 \end{aligned}$$

■ **Figure 3** Semantics of GCQ for a model $\mathcal{M} = (X, \rho)$. We used the notation $R ; S = \{(x, z) \mid \exists y \in Y \text{ s.t. } (x, y) \in R \text{ and } (y, z) \in S\}$ and $R \oplus S = \{(\binom{x}{u}, \binom{y}{v}) \mid (x, y) \in R \text{ and } (u, v) \in S\}$. \bullet is the unique element of X^0 and $\begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix}$ an element of X^n .

▶ **Example 5.** Consider $((\text{---} \oplus \text{---} \circlearrowleft) \oplus \text{---} \square); (R \oplus S) : (2, 1)$, assuming $R \in \Sigma_{2,0}$, $S \in \Sigma_{1,1}$. Its diagrammatic rendering is on the right. Note that the use of the dotted boxes induces a tree-like quality to diagrams. Indeed, they are a faithful representation for syntactic terms constructed from (GCQ).



We now turn to semantics. First, the notion of model of GCQ is similar to a model of CCQ.

▶ **Definition 6.** A model $\mathcal{M} = (X, \rho)$ is a set X and, for each $R \in \Sigma_{n,m}$, $\rho(R) \subseteq X^n \times X^m$.

Given a model $\mathcal{M} = (X, \rho)$, the semantics of $c : (n, m)$ is the relation $\llbracket c \rrbracket_{\mathcal{M}} \subseteq X^n \times X^m$ defined recursively in Figure 3. Armed with a notion of semantics, we can define query equivalence (\equiv) and inclusion (\leq) for GCQ terms analogously to Definition 4.

▶ **Example 7.** Consider the GCQ term $\llbracket \text{---} \bullet \rrbracket_{\mathcal{M}}$ of sort $(0, 0)$. For a model $\mathcal{M} = (X, \rho)$, its semantics $\llbracket \text{---} \bullet \rrbracket_{\mathcal{M}} \subseteq X^0 \times X^0$ is either the empty relation \emptyset , if X is empty, or the relation $\{(\bullet, \bullet)\}$, if X is not empty. Since $\emptyset \subseteq \{(\bullet, \bullet)\}$, and since $\llbracket \text{---} \square \rrbracket_{\mathcal{M}} = \{(\bullet, \bullet)\}$ for all models \mathcal{M} , it holds that $\llbracket \text{---} \bullet \rrbracket_{\mathcal{M}} \leq \llbracket \text{---} \square \rrbracket_{\mathcal{M}}$. Intuitively, the first term corresponds to the CCQ formula $\exists x. \top$, holding in all non empty models, while the second corresponds to the formula \top . In the remainder of this section we will make this intuition precise.

3.1 Expressivity

We now give a semantics preserving translation Θ from CCQ to GCQ. For each CCQ relation symbol $R \in \Sigma$ of arity n , we assume a corresponding GCQ symbol $R \in \Sigma_{n,0}$. Using Proposition 2, it suffices to consider judgments $n \vdash \phi$. For each, we obtain a GCQ term $\Theta(n \vdash \phi) : (n, 0)$. The translation Θ , given in Figure 4, is defined by recursion on the derivation of $n \vdash \phi$. Given a CCQ model $\mathcal{M} = (X, \rho)$, let $\Theta(\mathcal{M}) = (X, \rho')$ be the obvious corresponding GCQ model: $\rho'(R) = \rho(R) \times \{\bullet\}$. The following confirms that semantics is preserved.

▶ **Proposition 8.** For a CCQ model $\mathcal{M} = (X, \rho)$: $\vec{v} \in \llbracket n \vdash \phi \rrbracket_{\mathcal{M}}$ iff $(\vec{v}, \bullet) \in \llbracket \Theta(n \vdash \phi) \rrbracket_{\Theta(\mathcal{M})}$.

Furthermore, to characterise query inclusion in CCQ, it is enough to characterise it in GCQ.

▶ **Proposition 9.** For all CCQ formulas $n \vdash \phi$ and $n \vdash \psi$, $\phi \leq_{CCQ} \psi$ iff $\Theta(\phi) \leq_{GCQ} \Theta(\psi)$.

Proposition 8 yields the left-to-right direction. For right-to-left, we give a semantics-preserving translation Λ from GCQ to CCQ in Appendix A. Modulo \equiv , Λ is inverse of Θ .

▶ **Proposition 10.** There exists a semantics preserving translation Λ from GCQ to CCQ such that for all GCQ terms $c, d : (n, m)$, it holds that $c \leq_{GCQ} d$ iff $\Lambda(c) \leq_{CCQ} \Lambda(d)$.

$$\begin{aligned}
 \Theta(0 \vdash \top) &= \boxed{} \quad (\top) \quad \Theta(n \vdash \phi[x_{k+1}, x_k/x_k, x_{k+1}]) = \boxed{\boxed{\boxed{\phi(x_{k+1}, x_k)}}} \quad (\text{Sw}_{n,k}) \\
 \Theta(2 \vdash x_0 = x_1) &= \boxed{\bullet \bullet} \quad (=) \quad \Theta(n-1 \vdash \phi[x_{n-2}/x_{n-1}]) = \boxed{\boxed{\phi(x_{n-2})}} \quad (\text{Id}_n) \\
 \Theta(n \vdash R(x_0, \dots, x_{n-1})) &= \boxed{\boxed{R}} \quad (\Sigma) \quad \Theta(n-1 \vdash \exists x_{n-1}. \phi) = \boxed{\boxed{\phi}} \quad (\exists) \\
 \Theta(n+1 \vdash \phi) &= \boxed{\boxed{\phi}} \quad (\text{Nu}_n) \quad \Theta(m+n \vdash \phi \wedge (\psi[\dots])) = \boxed{\boxed{\phi}} \quad \boxed{\boxed{\psi}} \quad (\wedge)
 \end{aligned}$$

■ **Figure 4** Translation Θ from CCQ to GCQ.

Propositions 9 and 10 together imply that *CCQ* and *GCQ* have the same expressive power.

► **Example 11.** Recall from Example 7, that $\boxed{\bullet \bullet}$ is related to $\exists x. \top$. By translating the CCQ formula $0 \vdash \exists x_0. \top$ via Θ , one obtains $\boxed{\boxed{\bullet \bullet}}$. The latter and $\boxed{\bullet \bullet}$ are different – syntactically – but they are equal modulo \equiv . Note that their diagrams are similar: in the next section, we prove that terms differing only by dashed boxes are equal modulo \equiv .

4 From terms to string diagrams

The first step towards an equational characterisation of query inclusion is to move from GCQ, where the graphical notation was a faithful representation of ordinary syntactic terms, to bona fide *string diagrams*; that is, graphical notation for the arrows of a prop, a particularly simple kind of symmetric monoidal category (SMC). This is an advantage of GCQ syntax: its operations are amenable to an elegant axiomatisation. A hint of the good behaviour of GCQ operations is that query inclusion (and, therefore, query equivalence) is a (pre)congruence.

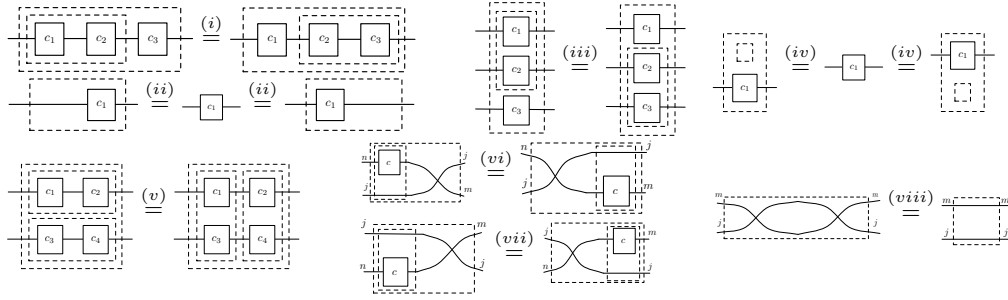
► **Lemma 12.**

- (i) Let $c, c' : (n, m)$ and $d, d' : (m, k)$ with $c \leq c'$ and $d \leq d'$. Then $(c ; d) \leq (c' ; d')$.
- (ii) Let $c, c' : (n, m)$ and $d, d' : (p, q)$ with $c \leq c'$ and $d \leq d'$. Then $(c \oplus d) \leq (c' \oplus d')$.

We now consider the laws of strict symmetric monoidal categories (Figure 5) and discover that any two GCQ terms identified by them are logically equivalent. This means that we can eliminate the clutter of dashed boxes from our graphical notation.

► **Proposition 13.** \equiv satisfies the axioms in Figure 5.

The terms of GCQ up-to query equivalence, therefore, organise themselves as arrows of a *monoidal category* (axioms (i)-(v)), and the operation of “erasing all dotted boxes” from diagrams is well-defined. The resulting structure is the well-known combinatorial/topological concept of *string diagram*. Equality reduces to the connectivity of their components, and is



■ **Figure 5** Axioms of strict symmetric monoidal categories. Wire annotations in (i)-(v) have been omitted for clarity.

thus stable under intuitive topological transformations, known as *diagrammatic reasoning*. For instance, axioms (ii) and (v) in Figure 5 imply that for $c_1 : (m_1, n_1)$ and $c_2 : (m_2, n_2)$

$$\begin{array}{c} \boxed{c_1} \\ \hline \end{array} \begin{array}{c} \hline \\ \boxed{c_2} \end{array} \equiv \begin{array}{c} \hline \\ \boxed{c_2} \end{array} \begin{array}{c} \boxed{c_1} \\ \hline \end{array}$$

Axioms (vi)-(viii) assert that GCQ terms modulo \equiv form a *symmetric* monoidal category (SMC). Therein, $\begin{array}{c} \xrightarrow{n} \\ \xleftarrow{m} \end{array}$ stands for the crossing of n wires over m wires. This has a standard recursive definition, using $\begin{array}{c} \xrightarrow{n} \\ \xleftarrow{m} \end{array}$, $-$ and the operations of GCQ. Intuitively, boxes “slide over” wire crossings. Moreover, it is well-known that (vi) and (vii) of Figure 5 imply the Yang-Baxter equation for crossings, which – with (viii) – implies that in diagrammatic reasoning wires do not “tangle” and crossings act like permutations of finite sets.

5 Axiomatisation

We have seen that, up-to query equivalence, GCQ enjoys the structural properties of SMCs. Here we give further properties that *characterise* query equivalence (\equiv) and inclusion (\leq).

Our first observation is that $\begin{array}{c} \curvearrowright \\ \bullet \end{array}$ and $\begin{array}{c} \bullet \\ \curvearrowleft \end{array}$ form, modulo \equiv , a *commutative monoid*, i.e., they satisfy axioms (A), (C) and (U) in Figure 6. Similarly, $\begin{array}{c} \curvearrowleft \\ \bullet \end{array}$ and $\begin{array}{c} \bullet \\ \curvearrowright \end{array}$ form a *cocommutative comonoid* (axioms (A^{op}), (C^{op}) and (U^{op})). Monoid and comonoid together give rise to a *special Frobenius bimonoid* (axioms (S) and (F)), a well-known algebraic structure that is important in various domains [1, 17, 7, 6].

► **Proposition 14.** \equiv satisfies the axioms in Figure 6.

Figure 7 shows a set of properties of query inclusion. The two axioms on the left state that $\begin{array}{c} \bullet \\ \curvearrowleft \end{array}$ is the left adjoint of $\begin{array}{c} \curvearrowright \\ \bullet \end{array}$ and the central axioms assert that $\begin{array}{c} \bullet \\ \curvearrowleft \end{array}$ is the left adjoint of $\begin{array}{c} \curvearrowright \\ \bullet \end{array}$. For the rightmost ones, it is convenient to introduce some syntactic sugar: $\begin{array}{c} \curvearrowright \\ \bullet \end{array}^n$, $\begin{array}{c} \bullet \\ \curvearrowleft \end{array}^n$ and $\begin{array}{c} \bullet \\ \curvearrowright \end{array}^n$ stand for the n -fold versions of monoid and comonoid. Now, axiom (L₁) asserts that $\begin{array}{c} \curvearrowright \\ \bullet \end{array}^n \begin{array}{c} \square \\ \text{R} \end{array}^m$ laxly commutes with $\begin{array}{c} \bullet \\ \curvearrowleft \end{array}^m$, while axiom (L₂) states that it laxly commutes with $\begin{array}{c} \bullet \\ \curvearrowright \end{array}^m$. In a nutshell, $\begin{array}{c} \curvearrowright \\ \bullet \end{array}^n \begin{array}{c} \square \\ \text{R} \end{array}^m$ is required to be a *lax comonoid morphism*.

► **Proposition 15.** \leq satisfies the axioms of Figure 7.

Interestingly, the observations we made so far suffice to characterise query equivalence and inclusion. This is the main theorem which we will prove in the remainder of this paper.

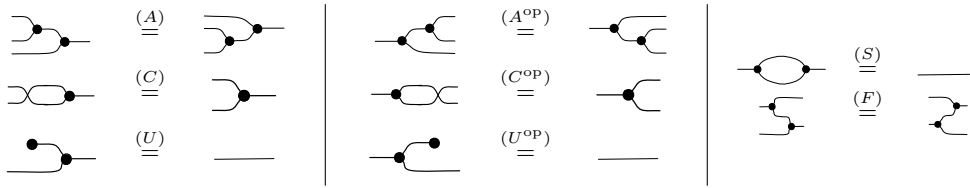


Figure 6 Axioms for special Frobenius bimonoids.

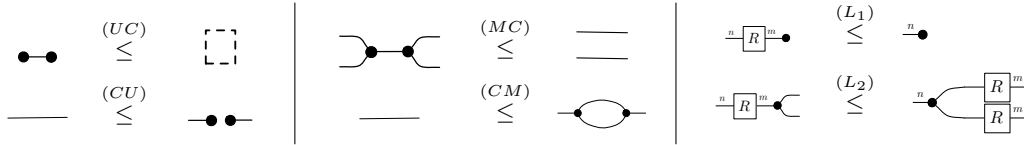


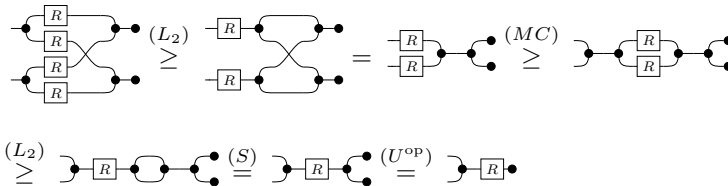
Figure 7 Axioms for adjointness of $\dashv\bullet$ and $\bullet\dashv$ (left) adjointness of $\dashv\bullet$ and $\bullet\dashv$ (center) lax comonoid morphism (right).

► **Definition 16.** The relation \leq_{CB_Σ} on the terms of GCQ is the smallest precongruence containing the equalities in Figures 5, 6, their converses and the inequalities in Figure 7. The relation $=_{\text{CB}_\Sigma}$ is the intersection of \leq_{CB_Σ} and its converse.

► **Theorem 17.** $\leq_{\text{CB}_\Sigma} = \leq$

► **Remark.** There is an apparent redundancy in Figure 7: (CM) follows immediately from (S) in Figure 6, while (S) can be derived from (CU) , (U^{op}) and (U) for one inclusion and (CM) for the other. We kept both (CM) and (S) because, as we shall see in §6, it is important to keep the algebraic structures of Figures 6 and 7 separate.

► **Example 18.** Recall the example from the Introduction. We can now prove the inclusion of queries using diagrammatic reasoning, as shown below. In the unlabeled equality we make use of the well-known *spider theorem*, which holds in every special Frobenius algebra [27].



5.1 Cartesian bicategories

The structure in Figures 6 and 7 is not arbitrary: these are exactly the laws of *cartesian bicategories*, a concept introduced by Carboni and Walters [12], that we recall below.

► **Definition 19.** A cartesian bicategory is a symmetric monoidal category \mathcal{B} with tensor \oplus and unit I , enriched over the category of partially ordered sets, such that:

1. every object X has a special Frobenius bimonoid: a monoid $\dashv\bullet^X : X \oplus X \rightarrow X$, $\bullet^X : I \rightarrow X$, a comonoid $\bullet^X\dashv : X \rightarrow X \oplus X$, $\dashv\bullet^X : X \rightarrow I$ satisfying the axioms in Figure 6;
2. the monoid and comonoid on X are adjoint (axioms in Figure 7, left and center);
3. every arrow $R : X \rightarrow Y$ is a lax comonoid morphism (axioms in Figure 7, right).

Furthermore, a morphism \mathcal{F} of cartesian bicategories is a functor $\mathcal{F} : \mathcal{B}_1 \rightarrow \mathcal{B}_2$ preserving the tensor, the partial orders and the monoid and comonoid on every object.

► **Example 20.** The archetypal cartesian bicategory is the category of sets and relations \mathbf{Rel} , with cartesian product \times as tensor and $1 = \{\bullet\}$ as unit. To be precise, \mathbf{Rel} has sets as objects and relations $R \subseteq X \times Y$ as arrows $X \rightarrow Y$. Composition and tensor are defined as in Figure 3. For each set X , the monoid and comonoid structure is: ${}^X\text{-}\bullet = \{(x, (\frac{x}{x})) \mid x \in X\}$, $\overset{x}{\bullet} = \{(x, \bullet) \mid x \in X\}$, $\overset{x}{\circlearrowleft} = \{((\frac{x}{x}), x) \mid x \in X\}$, $\bullet^x = \{(\bullet, x) \mid x \in X\}$.

Cartesian bicategories allow us to employ the usual construction from categorical logic: the arrows of the cartesian bicategory freely generated from Σ are GCQ terms modulo $=_{\mathbb{CB}_\Sigma}$, and morphisms from this cartesian bicategory to \mathbf{Rel} are exactly GCQ models.

► **Definition 21.** The ordered prop \mathbb{CB}_Σ has GCQ terms of sort (n, m) modulo $=_{\mathbb{CB}_\Sigma}$ as arrows $n \rightarrow m$. These are ordered by $\leq_{\mathbb{CB}_\Sigma}$.

► **Lemma 22.** \mathbb{CB}_Σ is a cartesian bicategory.

► **Proposition 23.** Models of GCQ (Definition 6) are in bijective correspondence with morphisms of cartesian bicategories $\mathbb{CB}_\Sigma \rightarrow \mathbf{Rel}$.

6 Discrete cospans of hypergraphs

In order to prove Theorem 17, in this section we give a combinatorial characterisation of free cartesian bicategories as hypergraphs-with-interfaces, formalised as a (bi)category of *cospans* equipped with an ordering inspired by Merlin and Chandra [14].

Indeed, the appearance of graph-like structures in the context of conjunctive queries should not come as a shock. Merlin and Chandra, to compute inclusion $\varphi \leq \psi$ of CCQ queries, translate them into hypergraphs G_φ, G_ψ with “interfaces” that represent free variables. Then $\varphi \leq \psi$ iff there exists an interface-preserving homomorphism from G_ψ to G_φ .

6.1 Hypergraphs and Cospans

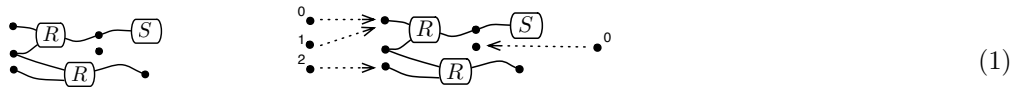
Our goal in this part is the characterisation of GCQ diagrams as certain combinatorial structures. We start by introducing the notion of Σ -hypergraph.

► **Definition 24** (Σ -hypergraph). Let Σ be a monoidal signature. A Σ -hypergraph G is a set G_V of vertices and, for each $R \in \Sigma_{n,m}$, a set of R -labeled hyperedges G_R , with source and target functions $s_R: G_R \rightarrow (G_V)^n, t_R: G_R \rightarrow (G_V)^m$. A morphism $f: G \rightarrow G'$ is a function $f_V: G_V \rightarrow G'_V$ and a family $f_R: G_R \rightarrow G'_R$, for each $R \in \Sigma_{n,m}$, s.t. the following commutes.

$$\begin{array}{ccccc} (G_V)^n & \xleftarrow{s_R} & G_R & \xrightarrow{t_R} & (G_V)^m \\ f_V \downarrow & & f_R \downarrow & & \downarrow f_V \\ (G'_V)^n & \xleftarrow{s'_R} & G'_R & \xrightarrow{t'_R} & (G'_V)^m \end{array}$$

A Σ -hypergraph G is finite if G_V and G_R are finite. Σ -hypergraphs and morphisms form the category \mathbf{Hyp}_Σ . Its full subcategory of finite Σ -hypergraphs is denoted by \mathbf{FHyp}_Σ .

We visualise hypergraphs as follows: \bullet is a vertex and $\overline{\text{[R]}}$ is a hyperedge with ordered tentacles. An example is shown below left, where $S \in \Sigma_{1,0}$ and $R \in \Sigma_{2,1}$.



In order to capture GCQ diagrams, we need to equip hypergraphs with interfaces, as illustrated in (1) on the right. Roughly speaking, an interface consists of two sets, called the left boundary and the right boundary. Each has an associated function to the underlying set of hypergraph vertices, depicted by the dotted arrows. Graphical structures with interfaces are common in computer science, (e.g., in automata theory [22], graph rewriting [18], Petri nets [32]). Categorically speaking, they are (discrete) cospans.

► **Definition 25 (Cospan).** Let \mathcal{C} be a finitely cocomplete category. A *cospan* from X to Y is a pair of arrows $X \rightarrow A \leftarrow Y$ in \mathcal{C} . A morphism $\alpha: (X \rightarrow A \leftarrow Y) \Rightarrow (X \rightarrow B \leftarrow Y)$ is an arrow $\alpha: A \rightarrow B$ in \mathcal{C} s.t. the diagram on the right commutes. Cospans $X \rightarrow A \leftarrow Y$ and $X \rightarrow B \leftarrow Y$ are *isomorphic* if

$$\begin{array}{ccc} & A & \\ X \rightarrow & \downarrow \alpha & \leftarrow Y \\ & B & \end{array} \quad (2)$$

there exists an isomorphism $A \rightarrow B$. For $X \in \mathcal{C}$, the *identity cospan* is $X \xrightarrow{\text{id}_X} X \xleftarrow{\text{id}_X} X$. The composition of $X \rightarrow A \xleftarrow{f} Y$ and $Y \xrightarrow{g} B \leftarrow Z$ is $X \rightarrow A +_{f,g} B \leftarrow Z$, obtained by taking the pushout of f and g . This data is the bicategory [4] $\text{Cospan}(\mathcal{C})$: the objects are those of \mathcal{C} , the arrows are cospans and 2-cells are homomorphisms. Finally, $\text{Cospan}(\mathcal{C})$ has monoidal product given by the coproduct in \mathcal{C} , with unit the initial object $0 \in \mathcal{C}$.

To avoid the complications of non-associative composition, it is common to consider a *category* of cospans, where isomorphic cospans are equated: let therefore $\text{Cospan}^{\leq} \mathcal{C}$ be the monoidal category that has isomorphism classes of cospans as arrows. Note that, when going from bicategory to category, after identifying isomorphic arrows it is usual to simply discard the 2-cells. Differently, we consider $\text{Cospan}^{\leq} \mathcal{C}$ to be locally preordered with $(X \rightarrow A \leftarrow Y) \leq (X \rightarrow B \leftarrow Y)$ if there exists a morphism α going *the other way*: $\alpha: (X \rightarrow B \leftarrow Y) \Rightarrow (X \rightarrow A \leftarrow Y)$. It is an easy exercise to verify that this (pre)ordering is well-defined and compatible with composition and monoidal product. Note that, in general, \leq is a genuine preorder: i.e. it is possible that both $(X \rightarrow A \leftarrow Y) \leq (X \rightarrow B \leftarrow Y)$ and $(X \rightarrow B \leftarrow Y) \leq (X \rightarrow A \leftarrow Y)$ without the cospans being isomorphic.

Armed with the requisite definitions, we can be rigorous about hypergraphs with interfaces.

► **Definition 26.** The preorder-enriched category $\text{Csp}^{\leq} \mathbf{FHyp}_{\Sigma}$ is the full subcategory of $\text{Cospan}^{\leq} \mathbf{FHyp}_{\Sigma}$ with objects the finite ordinals and arrows (isomorphism classes of) finite hypergraphs, inheriting the preorder. We call its arrows discrete cospans.

The above deserves an explanation: an ordinal n can be considered as the discrete hypergraph with vertices $\{0, \dots, n-1\}$. An arrow $n \rightarrow m$ in $\text{Csp}^{\leq} \mathbf{FHyp}_{\Sigma}$ is thus a cospan $n \rightarrow G \leftarrow m$ where G is a hypergraph and $n \rightarrow G$ and $m \rightarrow G$ are functions to its vertices. The picture in (1) shows a discrete cospan $3 \rightarrow 1$, with dotted lines representing the two morphisms.

6.2 Preordered cartesian bicategories

Here we explore the algebraic structure of $\text{Cospan}^{\leq} \mathcal{C}$. It is closely related to that of cartesian bicategories, yet – given the discussion above – it is more natural to consider $\text{Cospan}^{\leq} \mathcal{C}$ as a locally *preordered* category. We therefore need a slight generalisation of Definition 19.

► **Definition 27.** A *preordered cartesian bicategory* has the same structure as a cartesian bicategory (Definition 19), with one difference: the ordering is not required to be a partial order, merely a preorder – it is for this reason we separated the equational and inequational theories in Figures 6 and 7. The definition of morphism is as expected.

► **Proposition 28.** $\text{Cospan}^{\leq} \mathcal{C}$ is a preordered cartesian bicategory.

$$\begin{aligned}
 \llbracket \bullet \rightarrow \bullet \rrbracket &= 1 \rightarrow 1 \leftarrow 2, & \llbracket \bullet \rightarrow \bullet \rrbracket &= 2 \rightarrow 1 \leftarrow 1, & \llbracket \text{---} \rrbracket &= 1 \rightarrow 1 \leftarrow 1 \\
 \llbracket \bullet \rightarrow \bullet \rrbracket &= 1 \rightarrow 1 \leftarrow 0, & \llbracket \bullet \rightarrow \bullet \rrbracket &= 0 \rightarrow 1 \leftarrow 1, & \llbracket \text{---} \rrbracket &= 0 \rightarrow 0 \leftarrow 0 \\
 \llbracket \text{---} \rrbracket &= \text{---}, & \llbracket c ; d \rrbracket &= \llbracket c \rrbracket ; \llbracket d \rrbracket, & \llbracket c \oplus d \rrbracket &= \llbracket c \rrbracket \oplus \llbracket d \rrbracket \\
 \llbracket R \rrbracket &= \text{---}
 \end{aligned}$$

■ **Figure 8** Inductive definition of the isomorphism $\llbracket \cdot \rrbracket : \mathbb{CB}_{\Sigma}^{\leq} \rightarrow \mathbf{Csp}^{\leq} \mathbf{FHyp}_{\Sigma}$. In the first two lines, the finite ordinal n denotes the discrete hypergraph with n vertexes, and the functions between ordinals are uniquely determined by initiality of 0 and finality of 1.

As a consequence, $\mathbf{Cosp}^{\leq} \mathbf{FHyp}_{\Sigma}$, and thus also $\mathbf{Csp}^{\leq} \mathbf{FHyp}_{\Sigma}$, are preordered cartesian bicategories. The latter is of particular interest: the main result of this section, Theorem 31, states that $\mathbf{Csp}^{\leq} \mathbf{FHyp}_{\Sigma}$ is the *free* preordered cartesian bicategory on Σ , defined as follows.

► **Definition 29.** The preordered prop $\mathbb{CB}_{\Sigma}^{\leq}$ has, as arrows $n \rightarrow m$, GCQ terms of sort (n, m) modulo the smallest congruence generated by $=$ in Figures 5 and 6. These are ordered by the smallest precongruence generated by \leq in Figure 7.

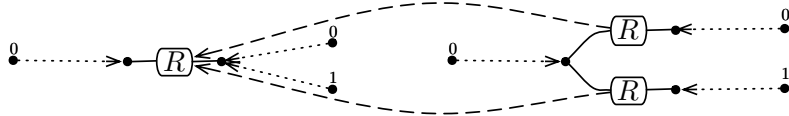
► **Remark.** Intuitively, the ordered prop \mathbb{CB}_{Σ} of Definition 21 is the ‘‘poset reduction’’ of the preordered prop $\mathbb{CB}_{\Sigma}^{\leq}$ introduced above. We will make this formal in Section 8.

Theorem 3.3 in [5] states that $\mathbf{Csp}^{\leq} \mathbf{FHyp}_{\Sigma}$ and $\mathbb{CB}_{\Sigma}^{\leq}$ are isomorphic as mere categories, i.e. forgetting the preorders. We thus need only to prove that the preorder of the two categories coincides, that is for all c, d in $\mathbb{CB}_{\Sigma}^{\leq}$,

$$c \leq d \text{ iff } \llbracket c \rrbracket \leq \llbracket d \rrbracket \tag{3}$$

where $\llbracket \cdot \rrbracket : \mathbb{CB}_{\Sigma}^{\leq} \rightarrow \mathbf{Csp}^{\leq} \mathbf{FHyp}_{\Sigma}$ is the isomorphism from [5] recalled in Figure 8. The ‘only-if’ part is immediate from Proposition 28. An alternative proof consists of checking, for each of the inclusions $c \leq d$ in Figure 7, that there exists a morphism of cospans from $\llbracket d \rrbracket$ to $\llbracket c \rrbracket$, as illustrated by the following example.

► **Example 30.** The left and the right hand side of (L_2) in Figure 7 for $R \in \Sigma_{1,1}$ are translated via $\llbracket \cdot \rrbracket$ into the cospans on the left and right below. The morphism from the rightmost hypergraph to the leftmost one, depicted by the dashed lines, witnesses the preorder.



The ‘if’ part of (3) requires some work. Its proof is given in full detail in Appendix B.2.

► **Theorem 31.** $\mathbf{Csp}^{\leq} \mathbf{FHyp}_{\Sigma} \cong \mathbb{CB}_{\Sigma}^{\leq}$ as preordered cartesian bicategories.

► **Example 32.** Recall Example 18. The derivation corresponds via $\llbracket \cdot \rrbracket$ to the homomorphism of cospans of hypergraphs illustrated in the Introduction.

7 Completeness for spans

Having established a combinatorial characterisation of the free preordered cartesian bicategory, here we prove our central completeness result, Theorem 37. In the preordered setting, completeness holds for ‘‘multirelational’’ models: the role of the poset-enriched category \mathbf{Rel} of sets and relations is taken by a (preorder-enriched) bicategory of spans of functions.

► **Definition 33** ($\text{Span}, \text{Span}^{\leq}$). Given a finitely complete category \mathcal{C} , the bicategory $\text{Span}(\mathcal{C})$ is dual to that of cospans of Definition 25: it can be defined as $\text{Cospan}(\mathcal{C}^{\text{op}})$. More explicitly, objects are those of \mathcal{C} , arrows of type $X \rightarrow Y$ are spans $X \leftarrow A \rightarrow Y$, composition \circ is defined by pullback and \oplus by categorical product. The 2-cells from $X \leftarrow A \rightarrow Y$ to $X \leftarrow B \rightarrow Y$ are span homomorphisms, that is arrows $\alpha: A \rightarrow B$ such that the diagram on the right commutes. As before, the bicategory $\text{Span}(\mathcal{C})$ can be seen as a category by identifying isomorphic spans. We obtain a category $\text{Span}^{\leq}\mathcal{C}$, on which we define a preorder in a similar way to $\text{Cospan}^{\leq}\mathcal{C}$, but in the *reverse* direction: $(X \rightarrow A \leftarrow Y) \leq (X \rightarrow B \leftarrow Y)$ when there is a homomorphism (4).

$$\begin{array}{ccc} & B & \\ \swarrow & \uparrow \alpha & \searrow \\ X & & Y \\ \nwarrow & \downarrow & \nearrow \\ & A & \end{array} \quad (4)$$

► **Lemma 34.** $\text{Span}^{\leq}\mathcal{C}$ is a preordered cartesian bicategory.

Models are now morphisms $\mathcal{M}: \mathbb{C}\mathbb{B}_{\Sigma}^{\leq} \rightarrow \text{Span}^{\leq}\mathbf{Set}$ of preordered cartesian bicategories. Observe that, since the interpretation of the monoid and comonoid structure is predetermined, a morphism is uniquely determined by its value on the object 1 and on $R \in \Sigma$. In other words, a model consists of a set $\mathcal{M}(1)$ and, for each $R \in \Sigma_{n,m}$, a span $\mathcal{M}(1)^n \leftarrow Y \rightarrow \mathcal{M}(1)^m$. This data is exactly the definition of a (possibly infinite) Σ -hypergraph (Definition 24).

► **Proposition 35.** Morphisms $\mathcal{M}: \mathbb{C}\mathbb{B}_{\Sigma}^{\leq} \rightarrow \text{Span}^{\leq}\mathbf{Set}$ are in bijective correspondence with Σ -hypergraphs.

Given this correspondence and the fact that $\mathbb{C}\mathbb{B}_{\Sigma}^{\leq} \cong \text{Csp}^{\leq}\mathbf{FHyp}_{\Sigma}$, each hypergraph G induces a morphism $\mathcal{U}_G: \text{Csp}^{\leq}\mathbf{FHyp}_{\Sigma} \rightarrow \text{Span}^{\leq}\mathbf{Set}$. Moreover, G acts like a representing object of a contravariant Hom-functor, in the following sense: \mathcal{U}_G maps $n \xrightarrow{\iota} G' \xleftarrow{\omega} m$ to

$$\mathbf{Hyp}_{\Sigma}[n, G] \xleftarrow{\iota; -} \mathbf{Hyp}_{\Sigma}[G', G] \xrightarrow{\omega; -} \mathbf{Hyp}_{\Sigma}[m, G]$$

where $\mathbf{Hyp}_{\Sigma}[G', G]$ is the set of hypergraph homomorphisms from G' to G , and $(\iota; -)$ and $(\omega; -)$ are defined, given $f \in \mathbf{Hyp}_{\Sigma}[G', G]$, by $(\iota; -)(f) = \iota; f$ and $(\omega; -)(f) = \omega; f$.

► **Proposition 36.** Suppose that $n \xrightarrow{\iota} G' \xleftarrow{\omega} m$ a discrete cospan in $\text{Csp}^{\leq}\mathbf{FHyp}_{\Sigma}$. Then

$$\mathcal{U}_G(n \xrightarrow{\iota} G' \xleftarrow{\omega} m) = \mathbf{Hyp}_{\Sigma}[n, G] \xleftarrow{\iota; -} \mathbf{Hyp}_{\Sigma}[G', G] \xrightarrow{\omega; -} \mathbf{Hyp}_{\Sigma}[m, G].$$

Proof. The conclusion of Theorem 31 allows us to use induction on $n \xrightarrow{\iota} G' \xleftarrow{\omega} m$. The inductive cases follow since the contravariant Hom-functor sends colimits to limits. Four of the base cases, $\llbracket \leftarrow \bullet \rrbracket$, $\llbracket \rightarrow \bullet \rrbracket$, $\llbracket \bullet \rrbracket$ and $\llbracket \bullet \rrbracket$, follow by the same argument, and the others ($\llbracket - \rrbracket$, $\llbracket \bowtie \rrbracket$ and $\llbracket R \rrbracket$) are easy to check. The details are in Appendix B.3. ◀

► **Theorem 37** (Completeness for $\text{Span}^{\leq}\mathbf{Set}$). Let $n \xrightarrow{\iota} G \xleftarrow{\omega} m$ and $n \xrightarrow{\iota'} G' \xleftarrow{\omega'} m$ be arrows in $\text{Csp}^{\leq}\mathbf{FHyp}_{\Sigma}$. If, for all morphisms $\mathcal{M}: \text{Csp}^{\leq}\mathbf{FHyp}_{\Sigma} \rightarrow \text{Span}^{\leq}\mathbf{Set}$, we have $\mathcal{M}(n \xrightarrow{\iota} G \xleftarrow{\omega} m) \leq \mathcal{M}(n \xrightarrow{\iota'} G' \xleftarrow{\omega'} m)$, then $(n \xrightarrow{\iota} G \xleftarrow{\omega} m) \leq (n \xrightarrow{\iota'} G' \xleftarrow{\omega'} m)$.

Proof. If the inequality holds for all morphisms, it holds for \mathcal{U}_G . By the conclusion of Proposition 36, there is a function $\alpha: \mathbf{Hyp}_{\Sigma}[G, G] \rightarrow \mathbf{Hyp}_{\Sigma}[G', G]$ making the diagram on

$$\begin{array}{ccc} & \mathbf{Hyp}_{\Sigma}[G, G] & \\ \swarrow \iota; - & \downarrow \alpha & \searrow \omega; - \\ \mathbf{Hyp}_{\Sigma}[n, G] & & \mathbf{Hyp}_{\Sigma}[m, G] \\ \nwarrow \iota'; - & \downarrow \alpha(\text{id}_G) & \nearrow \omega'; - \\ & \mathbf{Hyp}_{\Sigma}[G', G] & \end{array} \quad \begin{array}{ccc} & G' & \\ \swarrow \iota' & \downarrow \alpha(\text{id}_G) & \searrow \omega' \\ n & & m \\ \nwarrow \iota & \downarrow & \nearrow \omega \\ & G & \end{array}$$

the left commute. We take the identity $\text{id}_G \in \mathbf{Hyp}_{\Sigma}[G, G]$ and consider $\alpha(\text{id}_G): G' \rightarrow G$. By the commutativity of the left diagram, we have that $\iota = \iota'; \alpha(\text{id}_G)$ and $\omega = \omega'; \alpha(\text{id}_G)$. This means that the right diagram commutes, that is $(n \xrightarrow{\iota} G \xleftarrow{\omega} m) \leq (n \xrightarrow{\iota'} G' \xleftarrow{\omega'} m)$. ◀

► **Remark.** The reader may have noticed that, in the above proof, \mathcal{U}_G plays a role analogous to Chandra and Merlin’s [14] *natural model* for the formula corresponding to $n \xrightarrow{L} G \xleftarrow{\omega} m$.

Given the completeness theorem of this section, proving completeness for models of \mathbb{CB}_Σ in **Rel** is a simple step that we illustrate in the next section.

8 Completeness for relations

We conclude by showing how Theorem 37 leads to a proof of Theorem 17. The key observation lies in the tight connection between the preordered setting and the posetal one.

► **Definition 38.** Let \mathcal{C} be a preorder-enriched category. The poset-reduction of \mathcal{C} is the category \mathcal{C}^\sim having the same objects as \mathcal{C} and morphisms in \mathcal{C}^\sim are equivalence classes of those in \mathcal{C} modulo $\sim = \leq \cap \geq$. Composition is inherited from \mathcal{C} ; this is well-defined as \sim is a congruence wrt composition.

This assignment extends to a functor $(\cdot)^\sim$ from the category of preorder-enriched categories and functors to the category of poset-enriched ones. See Appendix B.4 for details.

We have already seen, although implicitly, an example of this construction in passing from $\mathbb{CB}_\Sigma^{\leq}$ (Definition 29) to \mathbb{CB}_Σ (Definition 21): it is indeed immediate to see that $(\mathbb{CB}_\Sigma^{\leq})^\sim = \mathbb{CB}_\Sigma$. Another crucial instance is provided by the following observation, where $\text{Span}^\sim \mathcal{C}$ is a shorthand for $(\text{Span}^{\leq} \mathcal{C})^\sim$.

► **Proposition 39.** $\text{Span}^\sim \mathbf{Set} \cong \mathbf{Rel}$ as cartesian bicategories.

The above proposition implicitly makes use of the following fact.

► **Proposition 40.** The functor $(\cdot)^\sim$ maps preorder-enriched cartesian bicategories and morphisms into poset-enriched cartesian bicategories and morphisms.

To conclude, it is convenient to establish a general theory of completeness results.

► **Definition 41.** Let \mathcal{C}, \mathcal{D} be preorder-enriched categories and let \mathcal{F} be a class of preordered functors $\mathcal{C} \rightarrow \mathcal{D}$. We say that \mathcal{C} is \mathcal{F} -complete for \mathcal{D} if for all arrows x, y in \mathcal{C} , $\mathcal{M}(x) \leq \mathcal{M}(y)$ for all $\mathcal{M} \in \mathcal{F}$ entails that $x \leq y$.

► **Lemma 42 (Transfer lemma).** Let \mathcal{C}, \mathcal{D} be preorder-enriched categories and \mathcal{F} a class of preordered functors $\mathcal{C} \rightarrow \mathcal{D}$. Assume \mathcal{C} to be \mathcal{F} -complete for \mathcal{D} .

1. Then \mathcal{C}^\sim is \mathcal{F}^\sim -complete for \mathcal{D}^\sim , where $\mathcal{F}^\sim = \{F^\sim \mid F \in \mathcal{F}\}$.
2. If $\mathcal{F} \subseteq \mathcal{F}'$, then \mathcal{C} is \mathcal{F}' -complete for \mathcal{D} .

All the pieces are now in place for a

Proof of Theorem 17. We need to show completeness – that is – assuming $c \leq c'$, we need to prove $c \leq_{\mathbb{CB}_\Sigma} c'$ for all GCQ terms c and c' . Observe that $c \leq_{\mathbb{CB}_\Sigma} c'$ if and only if

$$c \leq c' \text{ as arrows of } \mathbb{CB}_\Sigma \text{ (Definition 21).} \quad (\dagger)$$

Moreover, using Proposition 23, $c \leq c'$ iff

$$\mathcal{M}c \leq \mathcal{M}c', \text{ for all morphisms of cartesian bicategories } \mathcal{M}: \mathbb{CB}_\Sigma \rightarrow \mathbf{Rel}. \quad (\ddagger)$$

Our task becomes, therefore, to show that (\ddagger) implies (\dagger) . In other words, we need to prove \mathbb{CB}_Σ to be \mathcal{G} -complete for **Rel**, where \mathcal{G} is the class of morphisms of cartesian bicategories

of type $\mathcal{C} : \mathbb{C}\mathbb{B}_\Sigma \rightarrow \mathbf{Rel}$. Let \mathcal{F} be the class of morphisms of preorder-enriched cartesian bicategories from $\mathbb{C}\mathbb{B}_\Sigma^{\leq}$ to $\mathbf{Span}^{\leq} \mathbf{Set}$. Since, by Theorem 37, $\mathbb{C}\mathbb{B}_\Sigma^{\leq}$ is \mathcal{F} -complete for $\mathbf{Span}^{\leq} \mathbf{Set}$, we can conclude by Lemma 42.1 that $(\mathbb{C}\mathbb{B}_\Sigma^{\leq})^\sim$ is \mathcal{F}^\sim -complete for $(\mathbf{Span}^{\leq} \mathbf{Set})^\sim$. By Proposition 39, this is equivalent to $\mathbb{C}\mathbb{B}_\Sigma$ being \mathcal{F}^\sim -complete for \mathbf{Rel} . Now, by Proposition 40 $\mathcal{F}^\sim \subseteq \mathcal{G}$, so the claim follows by Lemma 42.2. \blacktriangleleft

9 Discussion, related and future work

We introduced a string diagrammatic language for conjunctive queries and demonstrated a sound and complete axiomatisation for query equivalence and inclusion. To prove completeness, we showed that our language provides an algebra able to express all hypergraphs and that our axioms characterise both hypergraph isomorphisms and existence of hypergraph morphisms. A recent result [19] introduced an extension of the allegorical fragment of the algebra of relations [33] that is able to express all graphs with tree-width at most 2. Furthermore, the isomorphism of these graphs can be axiomatised. The algebra in [19], which is clearly less expressive than ours, can be elegantly encoded into our string diagrams. The same holds for the representable allegories by Freyd and Scedrov [20].

We also prove completeness with respect to $\mathbf{Span}^{\leq} \mathbf{Set}$, the structure of which is closely related to the *bag semantics* of conjunctive queries in SQL. Indeed, the join of two SQL-tables is given by composition in $\mathbf{Span}^{\leq} \mathbf{Set}$ and not in \mathbf{Rel} : in the resulting table the same row can occur several times. As we have seen, with the relational semantics, query inclusion can be decided with Chandra and Merlin’s algorithm [14] and its reduction to existence of a hypergraph homomorphism. On the other hand, decidability of inclusion for the bag semantic is, famously, open. Originally posed by Vardi and Chaudhuri [15], it has been studied for different fragments and extensions of conjunctive queries [23, 2, 24]. It is worth mentioning that it is known [26] that there is a reduction to the homomorphism domination problem, which seems intimately related with our Proposition 36. Unfortunately, the preorder in $\mathbf{Span}^{\leq} \mathbf{Set}$ – the existence of a span morphism – does *not* directly correspond to bag inclusion: one must restrict to the existence of an *injective* morphism. We leave this promising path for future work.

References

- 1 Samson Abramsky and Bob Coecke. Categorical quantum mechanics. *CoRR*, abs/1401.4973, 2008. URL: <http://arxiv.org/abs/0808.1023>.
- 2 Foto N. Afrati, Matthew Damigos, and Manolis Gergatsoulis. Query containment under bag and bag-set semantics. *Inf. Process. Lett.*, 110(10):360–369, 2010. doi:10.1016/j.ipl.2010.02.017.
- 3 John Baez and Jason Erbele. Categories in control. *Theory and Application of Categories*, 30:836–881, 2015.
- 4 Jean Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, pages 1–77. Springer, 1967.
- 5 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 710–719. ACM, 2016.
- 6 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. Full abstraction for signal flow graphs. In *POPL 2015*, pages 515–526. ACM, 2015.
- 7 Roberto Bruni, Ivan Lanese, and Ugo Montanari. A basic algebra of stateless connectors. *Theoretical Computer Science*, 366(1–2):98–120, 2006.

- 8 Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari. A connector algebra for P/T nets interactions. In *CONCUR 2011*, volume 6901 of *LNCS*, pages 312–326. Springer, 2011. doi:10.1093/jigpal/6.2.349.
- 9 Roberto Bruni, Hernán C. Melgratti, Ugo Montanari, and Paweł Sobociński. Connector algebras for C/E and P/T nets' interactions. *Log Meth Comput Sci*, 9(16), 2013.
- 10 Roberto Bruni, Ugo Montanari, Gordon D. Plotkin, and Daniele Terreni. On hierarchical graphs: Reconciling bigraphs, gs-monoidal theories and gs-graphs. *Fundam. Inform.*, 134(3-4):287–317, 2014.
- 11 Aurelio Carboni, G Max Kelly, Robert FC Walters, and Richard J Wood. Cartesian bicategories ii. *Theory and Applications of Categories*, 19(6):93–124, 2008.
- 12 Aurelio Carboni and Robert FC Walters. Cartesian bicategories i. *Journal of pure and applied algebra*, 49(1-2):11–32, 1987.
- 13 Donald D Chamberlin and Raymond F Boyce. Sequel: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, pages 249–264. ACM, 1974.
- 14 Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90. ACM, 1977.
- 15 Surajit Chaudhuri and Moshe Y. Vardi. Optimization of *Real* conjunctive queries. In Catriel Beeri, editor, *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC, USA*, pages 59–70. ACM Press, 1993. URL: <http://dl.acm.org/citation.cfm?id=153850>, doi:10.1145/153850.153856.
- 16 Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.
- 17 B. Coecke and A. Kissinger. *Picturing Quantum Processes. A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2016.
- 18 A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Loewe. Algebraic approaches to graph transformation, part i: Basic concepts and double pushout approach. In *Handbook of Graph Grammars*, pages 163–246. World Scientific, 1997.
- 19 Enric Cosme-Llópez and Damien Pous. K4-free graphs as a free algebra. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, volume 83 of *LIPICs*, pages 76:1–76:14, 2017.
- 20 Peter J Freyd and Andre Scedrov. *Categories, allegories*, volume 39. Elsevier, 1990.
- 21 Dan R. Ghica and Aliaume Lopez. A structural and nominal syntax for diagrams. *CoRR*, abs/1702.01695, 2017. arXiv:1702.01695, doi:10.4204/EPTCS.266.4.
- 22 Victor Mikhaylovich Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16(5):1, 1961.
- 23 Yannis E. Ioannidis and Raghu Ramakrishnan. Containment of conjunctive queries: Beyond relations as sets. *ACM Trans. Database Syst.*, 20(3):288–324, 1995. doi:10.1145/211414.211419.
- 24 T. S. Jayram, Phokion G. Kolaitis, and Erik Vee. The containment problem for REAL conjunctive queries with inequalities. In Stijn Vansummeren, editor, *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 80–89. ACM, 2006. URL: <http://dl.acm.org/citation.cfm?id=1142351>, doi:10.1145/1142351.1142363.
- 25 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A complete axiomatisation of the zx-calculus for clifford+ t quantum mechanics. *arXiv preprint arXiv:1705.11151*, 2017.
- 26 Swastik Kopparty and Benjamin Rossman. The homomorphism domination exponent. *Eur. J. Comb.*, 32(7):1097–1114, 2011. doi:10.1016/j.ejc.2011.03.009.

- 27 Stephen Lack. Composing props. *Theory and Applications of Categories*, 13(9):147–163, 2004.
- 28 Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.
- 29 Paul-André Melliès and Noam Zeilberger. A bifibrational reconstruction of lawvere’s presheaf hyperdoctrine. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS ’16, New York, NY, USA, July 5-8, 2016*, pages 555–564, 2016. doi:10.1145/2933575.2934525.
- 30 Kang Feng Ng and Quanlong Wang. A universal completion of the zx-calculus. *arXiv preprint arXiv:1706.09877*, 2017.
- 31 Mehrnoosh Sadrzadeh, Stephen Clark, and Bob Coecke. The frobenius anatomy of word meanings I: subject and object relative pronouns. *CoRR*, abs/1404.5278, 2014.
- 32 Vladimiro Sassone and Paweł Sobociński. A congruence for Petri nets. *Electronic Notes in Theoretical Computer Science*, 127(2):107–120, 2005.
- 33 Alfred Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, 1941.

A

 A translation from GCQ to CCQ

To translate GCQ diagrams to CCQ formulas we need to introduce a minor syntactic variant of CCQ, this time assuming two countable sets of variables $Var_l = \{x_i \mid i \in \mathbb{N}\}$ and $Var_r = \{y_i \mid i \in \mathbb{N}\}$. The idea is that a diagram $c : (n, m)$ will translate to a formula that has its free variables in $\{x_0, \dots, x_{n-1}\} \cup \{y_0, \dots, y_{m-1}\}$, i.e. there are “left” free variables \vec{x} and “right” free variables \vec{y} .

► **Definition 43.** We write $n, m \vdash \phi$ if $fr(\phi) \subseteq \{x_0, \dots, x_{n-1}\} \cup \{y_0, \dots, y_{m-1}\}$ and $n + m \vdash \phi[x_{[n, n+m-1]}/y_{[0, m-1]}]$.




Next, for $R \in \Sigma_{n,m}$ we assume a CCQ signature in which R is a relation symbol with arity $n + m$. Then, given a GCQ model $\mathcal{M} = (X, \rho)$ we can obtain a CCQ model $\Lambda(\mathcal{M}) = (X, \rho')$ in the obvious way. With this in place, we can give a recursive translation Λ from GCQ terms to CCQ formulas. The details are given in Figure 9. and Λ preserves the semantics:

► **Proposition 44.** Fix a GCQ model $\mathcal{M} = (X, \rho)$ and suppose that $c : (n, m)$ is a GCQ formula. Then $(\vec{v}, \vec{w}) \in \llbracket c : (n, m) \rrbracket_{\mathcal{M}}$ iff $(\vec{v}, \vec{w}) \in \llbracket n + m \vdash \Lambda(c) \rrbracket_{\Lambda(\mathcal{M})}$.

Proof. Induction on the derivation of $c : (n, m)$. ◀

The following is immediate from the definition of the translations Θ and Λ .

► **Lemma 45.** Let $n \vdash \phi$ and \mathcal{M} be a CCQ model. Then $\llbracket n \vdash \phi \rrbracket_{\mathcal{M}} = \llbracket \Lambda\Theta(n \vdash \phi) \rrbracket_{\Lambda(\mathcal{M})}$. ◀

► **Example 46.** An interesting case is . It is the translation, via Θ , of $2 \vdash x_0 = x_1$. Returning to CCQ via Λ , we obtain $2, 0 \vdash \exists z. (x_0 = z) \wedge (x_1 = z) \wedge \top$. The formulas are quite different, but they are logically equivalent. The case of GCQ terms  and  is also interesting. The first translates via Λ to $0, 0 \vdash \top$, the second to $0, 0 \vdash \exists z_0. \top \wedge \top$.

$$\begin{array}{c}
 \Lambda(\text{---}\bigcirc)=1, 2 \vdash (x_0=y_0) \wedge (x_0=y_1), \quad \Lambda(\text{---}\bullet)=1, 0 \vdash \top, \quad \Lambda(\bigcirc\text{---})=2, 1 \vdash (x_0=y_0) \wedge (x_1=y_0), \quad \Lambda(\bullet\text{---})=0, 1 \vdash \top, \\
 \Lambda(\text{---}\text{---})=0, 0 \vdash \top, \quad \Lambda(\text{---})=1, 1 \vdash x_0=y_0, \quad \Lambda(\text{---}\text{---})=2, 2 \vdash (x_0=y_1) \wedge (x_1=y_0) \\
 \hline
 \frac{\begin{array}{c} \xrightarrow{m_1} \boxed{c_1} \xrightarrow{n_1} \\ \xrightarrow{m_2} \boxed{c_2} \xrightarrow{n_2} \end{array} \mapsto m_1, n_1 \vdash \Lambda(c_1) \quad \xrightarrow{m_2} \boxed{c_2} \xrightarrow{n_2} \mapsto m_2, n_2 \vdash \Lambda(c_2)}{\xrightarrow{m_1} \boxed{c_1} \xrightarrow{n_1} \quad \xrightarrow{m_2} \boxed{c_2} \xrightarrow{n_2} \mapsto m_1+m_2, n_1+n_2 \vdash \Lambda(c_1) \wedge (\Lambda(c_2)[x_{[m_1, m_1+m_2-1]}, y_{[n_1, n_1+n_2-1]} / x_{[0, m_2-1]}, y_{[0, n_2-1]})} (\oplus) \\
 \hline
 \frac{\xrightarrow{k} \boxed{c_1} \xrightarrow{m} \mapsto k, m \vdash \Lambda(c_1) \quad \xrightarrow{m} \boxed{c_2} \xrightarrow{n} \mapsto m, n \vdash \Lambda(c_2)}{\xrightarrow{k} \boxed{c_1} \xrightarrow{m} \boxed{c_2} \xrightarrow{n} \mapsto k, n \vdash \exists z. (\Lambda(c_1)[\vec{z}/\vec{y}]) \wedge (\Lambda(c_2)[\vec{z}/\vec{x}])} (;)
 \end{array}$$

■ **Figure 9** Translation Λ from GCQ to CCQ.

B Proofs

B.1 Proofs of Sections 2, 3, 4, and 5

Proof of Proposition 2. The ‘only if’ direction is a straightforward induction on the derivation of a formula generated by (CCQ). The ‘if’ is a trivial induction on derivations obtained from $\{\top, (R), (\exists), (=), (\wedge), (\text{Sw}_{n,k}), (\text{Id}_n), (\text{Nu}_n)\}$. ◀

Proof of Proposition 8. Easy induction on the derivation of $n \vdash \phi$. ◀

Proof of Proposition 9 and 10. The translation is given in Figure 9. The rest follows easily from Propositions 8 and 44 and Lemma 45. ◀

Proof of Lemma 12. Follows immediately from the definition of semantics and relational composition / tensor in Figure 3. ◀

Proof of Proposition 13. For each fixed model the axioms of Figure 5 are satisfied because the category of relations with monoidal product \times is symmetric monoidal. ◀

Proof of Lemma 22. For every object n , the monoid and comonoid are given by $\overset{n}{\text{---}}\bigcirc$, $\overset{n}{\bullet}\text{---}$, $\bigcirc\overset{n}{\text{---}}$ and $\bullet\overset{n}{\text{---}}$, standing for the ‘‘stacking’’ of n of these diagrams respectively in the usual manner. An easy induction shows that these satisfy the required laws. The definition of \mathbb{CB}_Σ asserts that for every $R \in \Sigma_{n,m}$, $\overset{n}{\text{---}}\boxed{R}^m$ is a lax comonoid morphism in \mathbb{CB}_Σ , but cartesian bicategories require this for *all* arrows. This follows by another induction. ◀

Proof of Lemma 23. In the easy direction, to extract a model $\mathcal{M} = (X, \rho)$ from a morphism of cartesian bicategories $\mathcal{F}: \mathbb{CB}_\Sigma \rightarrow \mathbf{Rel}$, define $X := \mathcal{F}(1)$ and let $\rho(R) := \mathcal{F}(R)$ for $R \in \Sigma$.

Conversely, given a model $\mathcal{M} = (X, \rho)$, we observe that the semantics map $\llbracket \cdot \rrbracket_{\mathcal{M}}$ (Figure 3) gives rise to a morphism of cartesian bicategories $\llbracket \cdot \rrbracket_{\mathcal{M}}: \mathbb{CB}_\Sigma \rightarrow \mathbf{Rel}$. To prove that it is well defined and preserves the ordering, one can easily see that the axioms of $=_{\mathbb{CB}_\Sigma}$ and $\leq_{\mathbb{CB}_\Sigma}$ are sound. By the inductive definition, $\llbracket \cdot \rrbracket_{\mathcal{M}}$ preserves composition ; and tensor \oplus . Finally, we observe that, by definition, $\llbracket \cdot \rrbracket_{\mathcal{M}}$ maps the monoids and comonoids of \mathbb{CB}_Σ into those of \mathbf{Rel} . ◀

B.2 Proofs of Section 6

We first prove Proposition 28 and then we work towards a proof of Theorem 31.

Proof of Proposition 28. We endow every object with a monoid and comonoid structure, prove these structures to be adjoint and satisfy the special Frobenius property.

1. Define the monoid/comonoid structure on every object: Considering \mathcal{C} as a cocartesian monoidal category via its coproduct, it is well known, that every object comes with a natural monoid structure. There is a monoidal functor $F: \mathcal{C} \rightarrow \mathbf{Cospan}^{\leq} \mathcal{C}$ sending $f: X \rightarrow Y$ to the cospan $X \xrightarrow{f} Y \xleftarrow{\text{id}} Y$ and mapping the natural monoid structure on every object through F yields the monoid structures in $\mathbf{Cospan}^{\leq} \mathcal{C}$. Furthermore, there is a duality operation \bullet^{op} on $\mathbf{Cospan}^{\leq} \mathcal{C}$ given by mapping $X \xrightarrow{f} Z \xleftarrow{g} Y$ to $Y \xrightarrow{g} Z \xleftarrow{f} X$. Now define the comonoid structure on every object as the dual of the monoid. It is easy to see that every morphism in $\mathbf{Cospan}^{\leq} \mathcal{C}$ is a lax comonoid homomorphism, which follows from the fact that every morphism in \mathcal{C} preserves the monoid structure.
2. The monoid and comonoid structures are adjoint: In general, for $f: X \rightarrow Y$ a morphism in \mathcal{C} , we have $F(f); F(f)^{\text{op}} \leq \text{id}_X$ and $\text{id}_Y \leq F(f)^{\text{op}}; F(f)$. This follows easily from the definition and implies the adjointness of monoid and comonoid.
3. The monoid and comonoid enjoy the special Frobenius property: The Frobenius law is a consequence of associativity of the natural monoid and its definition. The special law follows from the multiplication being epi. \blacktriangleleft

We will prove in Theorem 31 that $\mathbf{Csp}^{\leq} \mathbf{FHyp}_{\Sigma} \cong \mathbf{CB}_{\Sigma}^{\leq}$. It is convenient to begin with $\Sigma = \emptyset$. Consider the category \mathbb{F} : objects are finite ordinals $n = \{0, \dots, n-1\}$ and arrows all functions. Then $\mathbf{Cospan}^{\leq} \mathbb{F}$ is the free preordered cartesian bicategory on the empty signature.

► **Theorem 47.** $\mathbf{Cospan}^{\leq} \mathbb{F} \cong \mathbf{CB}_{\emptyset}^{\leq}$ as preordered cartesian bicategories.

Proof. The translation in Figure 8 defines an isomorphism $\llbracket \cdot \rrbracket: \mathbf{CB}_{\emptyset}^{\leq} \rightarrow \mathbf{Cospan}^{\leq} \mathbb{F}$ (first three lines). The translation $\llbracket \cdot \rrbracket: \mathbf{Cospan}^{\leq} \mathbb{F} \rightarrow \mathbf{CB}_{\emptyset}^{\leq}$ can be found in [5, Theorem 3.3], where it is proved that it defines an isomorphism between categories, i.e. forgetting the ordering. It thus suffices to prove, that both translations preserve the ordering. For $c, d \in \mathbf{CB}_{\emptyset}^{\leq}$, we have

$c \leq d$ implies $\llbracket c \rrbracket \leq \llbracket d \rrbracket$ by Proposition 28. Consider a morphism of cospans $n \begin{array}{c} \nearrow \\ \downarrow \alpha \\ \searrow \end{array} m$.

We want to prove $\llbracket n \rightarrow T \leftarrow m \rrbracket \leq \llbracket n \rightarrow S \leftarrow m \rrbracket$. Since every function $\alpha: S \rightarrow T$ can be decomposed into sums and compositions of $2 \rightarrow 1$ and $0 \rightarrow 1$ as demonstrated for example in [28, VII.5], we can consider only these cases. In the case $\alpha: 0 \rightarrow 1$, we have $n = m = 0$ and we have to prove $\bullet \longrightarrow \bullet \leq \llbracket \cdot \rrbracket$ which is axiom (UC). The case $\alpha: 2 \rightarrow 1$, can be

further reduced by the following observation: Given a diagram $n_1 + n_2 \begin{array}{c} \nearrow \\ \downarrow \\ \searrow \end{array} 1$, one

easily computes the composite of spans $n_1 + n_2 \rightarrow 2 \xleftarrow{\text{id}} 2 \xrightarrow{\text{id}} 2 \xleftarrow{\text{id}} 2 \xrightarrow{\text{id}} 2 \leftarrow m_1 + m_2$ to be

$n_1 + n_2 \rightarrow 2 \leftarrow m_1 + m_2$ and the composite $n_1 + n_2 \rightarrow 2 \xleftarrow{\text{id}} 2 \rightarrow 1 \leftarrow 2 \xrightarrow{\text{id}} 2 \leftarrow m_1 + m_2$ to be $n_1 + n_2 \rightarrow 1 \leftarrow m_1 + m_2$. By compositionality, it suffices to consider the case $2 \begin{array}{c} \nearrow \\ \downarrow \\ \searrow \end{array} 1$

which corresponds to $\bullet \text{---} \text{---} \bullet \leq \text{---} \text{---}$ which is axiom (MC). \blacktriangleleft

A cospan of hypergraphs is said to be *disconnected* if it is of the form $\llbracket R_0 \rrbracket \oplus \llbracket R_1 \rrbracket \oplus \dots \llbracket R_n \rrbracket$ for $R_0, \dots, R_n \in \Sigma$.

► **Lemma 48.** *Let $n \xrightarrow{\iota} E \xleftarrow{\omega} m$ and $n' \xrightarrow{\iota'} E' \xleftarrow{\omega'} m'$ be disconnected cospans. If there are functions $f: n \rightarrow n'$, $g: m \rightarrow m'$ and $h: E \rightarrow E'$ s.t. the following commutes*

$$\begin{array}{ccccc} n & \xrightarrow{\iota} & E & \xleftarrow{\omega} & m \\ f \downarrow & & h \downarrow & & \downarrow g \\ n' & \xrightarrow{\iota'} & E' & \xleftarrow{\omega'} & m' \end{array} \quad (5)$$

then $\llbracket n \xrightarrow{f; \iota'} E' \xleftarrow{g; \omega'} m \rrbracket \leq \llbracket n \xrightarrow{\iota} E \xleftarrow{\omega} m \rrbracket$.

Proof. First note that in the case of disconnected cospans, h uniquely determines f and g . To give a hypergraph homomorphism $h: E \rightarrow E'$ is the same as giving a label-preserving function between their sets of hyperedges, so we identify E and E' with their sets of hyperedges. We can now consider the labels separately, so assume to have only one label. Furthermore, we can consider each fiber over elements of E' separately, so assume $E' = 1$. So $n' \rightarrow E' \leftarrow m'$ consist of a single hyperedge with label $R \in \Sigma_{i,j}$, yielding

$$\llbracket n' \xrightarrow{\iota'} E' \xleftarrow{\omega'} m' \rrbracket = \text{---} \boxed{R} \text{---}^j$$

and thus $n' = i$ and $m' = j$. It now suffices to consider cases where the size of E is either 0 or

2, yielding diagrams $\begin{array}{ccccc} 0 & \longrightarrow & 0 & \longleftarrow & 0 \\ i \downarrow & & i \downarrow & & \downarrow i \\ i & \xrightarrow{\iota'} & 1 & \xleftarrow{\omega'} & j \end{array} \quad (6) \quad \text{and} \quad \begin{array}{ccccc} i+i & \xrightarrow{\iota'+\iota'} & 2 & \xleftarrow{\omega'+\omega'} & j+j \\ \nabla \downarrow & & \downarrow \nabla & & \downarrow \nabla \\ i & \xrightarrow{\iota'} & 1 & \xleftarrow{\omega'} & j \end{array} \quad (7)$

The result for $|E| \geq 2$ can be obtained inductively from these base cases. For (6),

$$\llbracket 0 \rightarrow 0 \leftarrow 0 \rrbracket = \text{---} \text{---}, \quad \text{and} \quad \llbracket 0 \rightarrow 1 \leftarrow 0 \rrbracket = \bullet \text{---} \boxed{R} \text{---}^j \bullet.$$

The following derivation thus suffices: $\bullet \text{---} \boxed{R} \text{---}^j \bullet \stackrel{(L_1)}{\leq} \bullet \text{---} \bullet \stackrel{(UC)}{\leq} \text{---} \text{---}$. For (7),

$$\llbracket i+i \rightarrow 1 \leftarrow j+j \rrbracket = \begin{array}{c} i \quad i \quad j \quad j \\ \text{---} \bullet \text{---} \boxed{R} \text{---} \bullet \text{---} \\ i \quad i \quad j \quad j \end{array}, \quad \llbracket i+i \xrightarrow{\iota'+\iota'} 2 \xleftarrow{\omega'+\omega'} j+j \rrbracket = \begin{array}{c} i \quad j \\ \text{---} \boxed{R} \text{---} \\ i \quad j \end{array}.$$
 This de-

rivation thus completes the proof: $\begin{array}{c} i \quad i \quad j \quad j \\ \text{---} \bullet \text{---} \boxed{R} \text{---} \bullet \text{---} \\ i \quad i \quad j \quad j \end{array} \stackrel{(L_2)}{\leq} \begin{array}{c} i \quad i \quad j \quad j \\ \text{---} \bullet \text{---} \begin{array}{c} \boxed{R} \\ \boxed{R} \end{array} \text{---} \\ i \quad i \quad j \quad j \end{array} \stackrel{(MC)}{\leq} \begin{array}{c} i \quad j \\ \text{---} \boxed{R} \text{---} \\ i \quad j \end{array} \quad \blacktriangleleft$

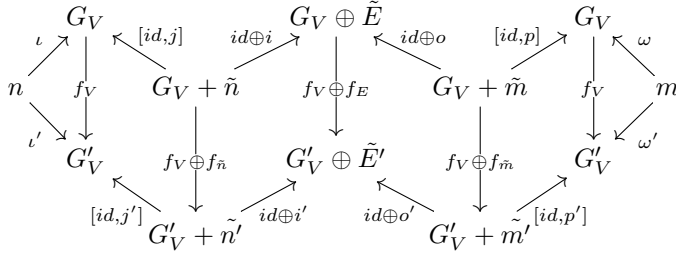
We have now all the ingredients to prove Theorem 31.

Proof of Theorem 31. The proof relies on a result appearing in the proof of Theorem 3.3 in [5]: every discrete cospan of hypergraphs $n \xrightarrow{\iota} G \xleftarrow{\omega} m$ can be written as the composition

$$\begin{array}{ccccccc} & & G_V & & G_V \oplus \tilde{E} & & G_V \\ & \nearrow \iota & \swarrow [id, j] & \nearrow id \oplus i & \swarrow id \oplus o & \nearrow [id, p] & \swarrow \omega \\ n & & G_V + \tilde{n} & & G_V + \tilde{m} & & m \end{array}$$

where $\tilde{n} \xrightarrow{i} \tilde{E} \xleftarrow{o} \tilde{m}$ is disconnected, G_V is the set of vertices of G^2 , $j: \tilde{n} \rightarrow G_V$ and $j: \tilde{m} \rightarrow G_V$ maps the vertices of $\tilde{n} \rightarrow \tilde{E} \leftarrow \tilde{m}$ into those of G . We only need to prove the right-to-left implication of (3). We will show that if $n \rightarrow G' \leftarrow m \leq n \rightarrow G \leftarrow m$ then $\llbracket n \rightarrow G' \leftarrow m \rrbracket \leq \llbracket n \rightarrow G \leftarrow m \rrbracket$.

Assume now that $n \xrightarrow{l'} G' \xleftarrow{\omega'} m \leq n \xrightarrow{l} G \xleftarrow{\omega} m$, i.e., there exists an $f: G \rightarrow G'$ such that $f l = l'$ and $f \omega = \omega'$. The morphism f induces $f_V: G_V \rightarrow G'_V$, $f_E: \tilde{E} \rightarrow \tilde{E}'$, $f_{\tilde{n}}: \tilde{n} \rightarrow \tilde{n}'$ and $f_{\tilde{m}}: \tilde{m} \rightarrow \tilde{m}'$ making the following commute.



From the commutativity of the above diagram, one has:

$$\begin{aligned}
 (\gamma_1 :=) \quad n \rightarrow G'_V \leftarrow G_V + \tilde{n} &\leq n \rightarrow G_V \leftarrow G_V + \tilde{n} & (=: \delta_1) \\
 (\gamma_2 :=) \quad G_V + \tilde{m} \rightarrow G'_V \leftarrow m &\leq G_V + \tilde{m} \rightarrow G_V \leftarrow m & (=: \delta_2) \\
 (\gamma_3 :=) \quad G_V \rightarrow G'_V \leftarrow G_V &\leq G_V \rightarrow G_V \leftarrow G_V & (=: \delta_3) \\
 (\gamma_4 :=) \quad \tilde{n} \rightarrow \tilde{E}' \leftarrow \tilde{m} &\leq \tilde{n} \rightarrow \tilde{E} \leftarrow \tilde{m} & (=: \delta_4)
 \end{aligned}$$

Since the first three inequations only involve sets and functions, one can use the conclusion of Theorem 47 and deduce that: $\llbracket \gamma_i \rrbracket \leq \llbracket \delta_i \rrbracket$ for $i \in \{1, 2, 3\}$. From the fourth inequation, via Lemma 48, one obtains furthermore $\llbracket \gamma_4 \rrbracket \leq \llbracket \delta_4 \rrbracket$ and concludes as follows.

$$\begin{aligned}
 \llbracket n \rightarrow G' \leftarrow m \rrbracket &= \llbracket \gamma_1 ; (\gamma_3 \oplus \gamma_4) ; \gamma_2 \rrbracket = \llbracket \gamma_1 \rrbracket ; (\llbracket \gamma_3 \rrbracket \oplus \llbracket \gamma_4 \rrbracket) ; \llbracket \gamma_2 \rrbracket \\
 &\leq \llbracket \delta_1 \rrbracket ; (\llbracket \delta_3 \rrbracket \oplus \llbracket \delta_4 \rrbracket) ; \llbracket \delta_2 \rrbracket = \llbracket \delta_1 ; (\delta_3 \oplus \delta_4) ; \delta_2 \rrbracket = \llbracket n \rightarrow G \leftarrow m \rrbracket \quad \blacktriangleleft
 \end{aligned}$$

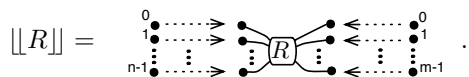
B.3 Proofs of Section 7

Proof of Lemma 34. Immediate from Proposition 28 by duality. ◀

Proof of Proposition 35. As stated in the main text, \mathcal{M} is uniquely determined by the set $\mathcal{M}(1)$ and, for each $R \in \Sigma_{n,m}$, a span $\mathcal{M}(R): \mathcal{M}(1)^n \rightarrow \mathcal{M}(1)^m$. This data is that of a (possibly infinite) hypergraph (Definition 24). ◀

Proof of Proposition 36. By definition, $\mathcal{U}_G(1) = G_V$ and $\mathcal{U}_G(\llbracket R \rrbracket) = (G_V)^n \xrightarrow{\mathcal{R}} G_R \xleftarrow{\mathcal{L}_R} (G_V)^m$ for each $R \in \Sigma_{n,m}$. Below, we also use the fact that $(G_V)^n$ is $\mathbf{Hyp}_{\Sigma}[n, G]$.

The conclusion of Theorem 31 allows us to argue by induction on $n \xrightarrow{l} G' \xleftarrow{\omega} m$. The base cases are $\llbracket \text{---} \bullet \rrbracket$, $\llbracket \text{---} \bullet \text{---} \rrbracket$, $\llbracket \text{---} \bullet \rrbracket$, $\llbracket \bullet \text{---} \rrbracket$, $\llbracket \text{---} \rrbracket$, $\llbracket \text{---} \times \text{---} \rrbracket$ and $\llbracket R \rrbracket$. Let us consider the last of these, where $n \xrightarrow{l} G' \xleftarrow{\omega} m$ is



² Since cospans are taken up-to isomorphism and since G is finite one can always assume, without loss of generality, that G_V is a finite ordinal.

Any homomorphism $f: G' \rightarrow G$ maps its single hyperedge to an R -hyperedge of G , call it e_f , the n vertices in the image of ι to the source of e_f ($\iota; f = s_R(e_f)$) and the m vertices in the image of ω to the target of e_f ($\omega; f = t_R(e_f)$). This means that the following commutes:

$$\begin{array}{ccc}
 & G_R & \\
 s_R \swarrow & & \searrow t_R \\
 (G_V)^n & & (G_V)^m \\
 \iota; - \swarrow & \text{Hyp}_\Sigma[G', G] & \searrow \omega; - \\
 & e_- \uparrow &
 \end{array}$$

The function $e_-: \text{Hyp}_\Sigma[G', G] \rightarrow G_R$ is clearly an isomorphism of spans. The other base cases are simpler or, as stated in the main text, follow from the fact that $\text{Hyp}_\Sigma[_, G]$ maps colimits to limits, which also immediately implies the inductive case. \blacktriangleleft

B.4 Proofs of Section 8

Observe that we have a canonical identity-on-objects-functor $\mathcal{A}_C: \mathcal{C} \rightarrow \mathcal{C}^\sim$ that sends a morphism in \mathcal{C} to its \sim -equivalence class in \mathcal{C}^\sim . We will omit the subscript on \mathcal{A} whenever possible. An immediate consequence of the definition is that \mathcal{A} preserves and reflects the ordering in the following sense:

► **Lemma 49.** *For \mathcal{C} a preorder-enriched category, and x, y morphisms in \mathcal{C} , we have $\mathcal{A}(x) \leq \mathcal{A}(y)$ if and only if $x \leq y$.* \blacktriangleleft

The functors \mathcal{A} exhibit the following universal property:

► **Lemma 50.** *For every preordered functor $F: \mathcal{C} \rightarrow \mathcal{D}$ between the preordered category \mathcal{C} and poset-enriched category \mathcal{D} , there is a unique poset-enriched functor $G: \mathcal{C}^\sim \rightarrow \mathcal{D}$ making the left diagram commute. Hence, for every preordered functor $H: \mathcal{C} \rightarrow \mathcal{C}'$, \mathcal{C}' preorder-enriched, there is a unique functor $H^\sim: \mathcal{C}^\sim \rightarrow \mathcal{C}'^\sim$ making the right diagram commute.*

$$\begin{array}{ccc}
 \mathcal{C} & \xrightarrow{F} & \mathcal{D} \\
 \mathcal{A}_C \downarrow & \nearrow G & \\
 \mathcal{C}^\sim & &
 \end{array}
 \quad
 \begin{array}{ccc}
 \mathcal{C} & \xrightarrow{H} & \mathcal{C}' \\
 \mathcal{A}_C \downarrow & & \downarrow \mathcal{A}_{C'} \\
 \mathcal{C}^\sim & \xrightarrow{H^\sim} & \mathcal{C}'^\sim
 \end{array}$$

Proof. For a morphism $f \in \mathcal{C}$ let $[f]$ denote the equivalence class of f modulo \sim . Then setting $G([f]) = F(f)$ is well-defined, since \mathcal{D} is a poset-enriched category. G defines a functor since \sim is a congruence, hence compatible with composition. Since \mathcal{A}_C is surjective on objects and morphisms, there can be at most one such functor G , hence G is unique. \blacktriangleleft

In other words, we get a function, $(\cdot)^\sim$, that turns functors between preorder-enriched categories into functors between the associated poset-enriched ones.

Proof of Proposition 39. We recall a well-known construction of the ordinary category of relations: a span $X \xleftarrow{f} A \xrightarrow{g} Y$ induces a relation $R_A \subseteq X \times Y$ by factorising $A \xrightarrow{[f, g]} X \times Y$ as a surjection followed by an injection; the injection, when composed with the projections, yields a *jointly-injective* span. These, up-to span isomorphism, are the same thing as subsets $R_A \subseteq X \times Y$. This procedure respects composition and monoidal product, yielding a functorial mapping $\text{Span}^{\leq} \mathbf{Set} \rightarrow \mathbf{Rel}$ on objects and arrows. Given the above, it suffices to show that there exists a span homomorphism $(X \leftarrow A \rightarrow Y) \rightarrow (X \leftarrow B \rightarrow Y)$ iff $R_A \subseteq R_B$

as relations. The ‘only if’ direction is implied by the dotted function below, which is an injection since it is the first part of a factorisation of an injection.

$$\begin{array}{ccccc}
 A & \longrightarrow & B & \twoheadrightarrow & R_B \\
 \downarrow & & & \searrow & \downarrow \\
 R_A & \xrightarrow{\quad} & X \times Y & &
 \end{array}$$

For the ‘if’ part, since (by the axiom of choice) surjective functions split, we obtain $R_B \rightarrow B$. Then $A \twoheadrightarrow R_A \rightarrow R_B \rightarrow B$ is easily shown to be a homomorphism of spans. ◀

Proof of Proposition 40. We stated the axioms of preordered cartesian bicategories and cartesian bicategories in a way that makes the first part obvious. Given a morphism $F: \mathcal{B}_1 \rightarrow \mathcal{B}_2$ of preorder-enriched cartesian bicategories, clearly F^\sim is still an order-preserving monoidal functor. It also preserves the monoid and comonoid structures. ◀

Proof of Lemma 42. The second item is trivial. For the first one, let x, y be morphisms in \mathcal{C}^\sim such that $G(x) \leq G(y)$ for all $G \in \mathcal{F}^\sim$. We want to prove $x \leq y$. Now let $F \in \mathcal{F}$ be arbitrary. Then $F^\sim(x) \leq F^\sim(y)$ by assumption on x, y . Since morphisms in \mathcal{C}^\sim are just equivalence classes of morphisms in \mathcal{C} , choose representatives, i.e. morphisms f, g in \mathcal{C} such that $\mathcal{A}(f) = x$ and $\mathcal{A}(g) = y$. Since the diagram

$$\begin{array}{ccc}
 \mathcal{C} & \xrightarrow{F} & \mathcal{D} \\
 \mathcal{A} \downarrow & & \downarrow \mathcal{A} \\
 \mathcal{C}^\sim & \xrightarrow{F^\sim} & \mathcal{D}^\sim
 \end{array}$$

commutes, we get $\mathcal{A}(F(f)) = F^\sim(\mathcal{A}(f)) = F^\sim(x) \leq F^\sim(y) = F^\sim(\mathcal{A}(g)) = \mathcal{A}(F(g))$. Since \mathcal{A} reflects the ordering (Lemma 49), we get $F(f) \leq F(g)$. But $F \in \mathcal{F}$ was arbitrary, therefore $f \leq g$, since \mathcal{C} is \mathcal{F} -complete for \mathcal{D} . But therefore $x = \mathcal{A}(f) \leq \mathcal{A}(g) = y$. ◀

Approximating Probabilistic Automata by Regular Languages

Rohit Chadha¹

University of Missouri
Columbia, USA
chadhar@missouri.edu

A. Prasad Sistla²

University of Illinois, Chicago
Chicago, USA
sistla@uic.edu

Mahesh Viswanathan³

University of Illinois, Urbana-Champaign
Urbana, USA
vmahesh@illinois.edu

Abstract

A probabilistic finite automaton (PFA) \mathcal{A} is said to be regular-approximable with respect to (x, y) , if there is a regular language that contains all words accepted by \mathcal{A} with probability at least $x + y$, but does not contain any word accepted with probability at most x . We show that the problem of determining if a PFA \mathcal{A} is regular-approximable with respect to (x, y) is not recursively enumerable. We then show that many tractable sub-classes of PFAs identified in the literature – hierarchical PFAs, polynomially ambiguous PFAs, and eventually weakly ergodic PFAs – are regular-approximable with respect to all (x, y) . Establishing the regular-approximability of a PFA has the nice consequence that its value can be effectively approximated, and the emptiness problem can be decided under the assumption of isolation.

2012 ACM Subject Classification Theory of computation → Probabilistic computation

Keywords and phrases Probabilistic Finite Automata, Regular Languages, Ambiguity

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.14

1 Introduction

Probabilistic finite automata (PFA), introduced by Rabin [26], are finite state machines that read symbols from an input string and whose state transitions are determined by the input symbol being read and the result of a coin toss. For an input string w , the probability of accepting w is the measure of all runs of the automaton on w that end in an accepting state. Given a threshold x , the language recognized by a PFA is the collection of all words w whose probability of acceptance is at least x . Probabilistic finite automata serve as convenient models of open stochastic systems. Despite their simplicity, PFAs are a surprisingly powerful model of computation and typical decision problems of PFAs are undecidable. For example, the classical decision problem that arises when verifying a design described by a PFA against regular specifications, namely emptiness, is undecidable [11].

¹ NSF CNS 1314338 and NSF CNS 1553548

² NSF CNS 1314485 and NSF CCF 1564296

³ NSF CSR 1422798 and NSF CPS 1329991



The reason for the computational hardness of problems involving PFAs is because they can “simulate” powerful computational models like Turing machines. The question we ask is if, despite this evidence of expressive power, languages recognized by PFAs can be “approximated” by regular languages, in a sense that we will make precise later in this introduction. If PFAs can be approximated by regular languages, it opens up the possibility of solving some of these decision problems partially. For example, if we want to verify that a stochastic open system modeled by a PFA meets a regular specification, we could approximate the PFA language by a regular language, and then check containment/emptiness. This approach would be similar to the effective role finite state abstractions have played in verifying real world designs.

So what type of regular approximations are we talking about? For a PFA \mathcal{A} , let $L_{\geq x}(\mathcal{A})$ and $L_{\leq x}(\mathcal{A})$ be the sets of strings accepted with probability $\geq x$ and $\leq x$, respectively. We say that \mathcal{A} is *regular-approximable with respect to* (x, y) if there is a regular language L that separates $L_{\geq x+y}(\mathcal{A})$ and $L_{\leq x}(\mathcal{A})$, i.e., $L_{\geq x+y}(\mathcal{A}) \subseteq L$ and $L \cap L_{\leq x}(\mathcal{A}) = \emptyset$ (i.e., $L \subseteq L_{>x}(\mathcal{A})$). Thus, L is a “over-approximation” of $L_{\geq x+y}(\mathcal{A})$ and an “under-approximation” of $L_{>x}(\mathcal{A})$. Such a notion of separability has been previously studied in the context of PFAs [24]. Separability using regular languages have played a significant role in understanding the expressive power of formal languages and coming up with decision procedures [12, 25].

First, even if $L_{\geq x+y}(\mathcal{A})$ and $L_{\leq x}(\mathcal{A})$ are not regular, \mathcal{A} maybe regular-approximable with respect to (x, y) (see Example 7). On the other hand, there are PFAs \mathcal{A} and (x, y) such that \mathcal{A} is not regular-approximable with respect to (x, y) (see [24] and Theorem 8). So how difficult is it to check regular-approximability? We show that the problem of determining if a PFA \mathcal{A} is regular-approximable with respect to (x, y) is not recursively enumerable (Theorem 9). Our proof relies on showing that a closely related problem of determining if a PFA \mathcal{A} is regular-approximable with respect to *some* (x, y) is Σ_2^0 -hard; Σ_2^0 is the second level of the arithmetic hierarchy.

Given that determining if a PFA \mathcal{A} is regular-approximable with respect to (x, y) is undecidable, we try to identify sufficient conditions that guarantee the regular-approximability of PFAs in a very strong sense. In particular, we identify conditions under which a PFA is guaranteed to be regular-approximable with respect to *every* pair (x, y) . Further, we’d like to identify when the regular language approximating the PFA can be effectively constructed from \mathcal{A} and (x, y) . PFAs that satisfy such strong properties are amenable to automated analysis. We show that problems that are undecidable (or open) for general PFAs, become decidable in such situations. We give examples of two such problems. The first is the value problem for PFAs, where the goal is to compute the supremum of the acceptance probabilities of all input words. When a PFA \mathcal{A} represents the product of an open probabilistic system and an incorrectness property given as deterministic automaton on the system executions, then value of \mathcal{A} gives a tight upper bound on the probability of incorrectness of the system on all input sequences. Decision versions of the value problem are known to be Σ_2^0 -complete. The second problem is checking emptiness under isolation. A threshold x is said to be isolated for PFA \mathcal{A} with a degree of isolation ϵ if the acceptance probability of every word is ϵ -bounded away from x . A classical result is that when x is isolated, the language $L_{>x}(\mathcal{A})$ is regular [26]. The emptiness under isolation problem, is to determine if the language $L_{>x}(\mathcal{A})$ is empty, under the promise that x is an isolated cut-point for \mathcal{A} (but no degree of isolation is given). The decidability of this is a long standing open problem. We prove that for PFAs that are effectively regular-approximable (that is regular separator L can be constructed for every (x, y)), the value problem can be approximated with arbitrary precision (Theorem 11) and the emptiness under isolation is decidable (Corollary 12).

Our semantic condition that identifies when a PFA is regular-approximable is as follows. A *leaky* transition is a transition whose probability is less than 1. A PFA \mathcal{A} is said to be *leak monotonic* if for every ϵ , there is a number N_ϵ such that, for any input u , the measure of all accepting runs ρ on u that have at least N_ϵ leaks is $< \epsilon$. In other words, runs with many leaky transitions contribute very little to the acceptance probability of a word. We prove that leak monotonic PFAs are regular-approximable with respect to every (x, y) (Corollary 20). If a leak monotonic PFA in addition has the property that N_ϵ can be computed from ϵ , then one can show that the regular separator of $L_{\geq x+y}(\mathcal{A})$ and $L_{\leq x}(\mathcal{A})$ can also be effectively constructed (Corollary 20). The deterministic automaton B that recognizes the regular separator has the property that its computation on any input u can be used to approximately compute \mathcal{A} 's acceptance probability as follows – one can associate a function from states of B to $[0, 1]$ such that the label of the state reached on reading u is an approximation of the acceptance probability of u .

Our last set of results in the paper show that many of the tractable sub-classes of PFAs discovered, enjoy the nice decidability properties because of regular-approximability. Hierarchical PFAs [9] are those that obey the restriction that states can be partitioned into a hierarchy of ranks, and transitions from a state only go to states of the same or higher rank (for a precise definition, see paragraph before Theorem 26). Another class of PFAs are those with polynomial ambiguity [16]. These are PFAs with the property that on any input u , the number of accepting runs on u (not its probability) is bounded by a polynomial function of the input length $|u|$. Both these sub-classes of PFAs are effectively leak monotonic, and hence effectively regular-approximable. Thus their value can be effectively approximated, and the emptiness problem is decidable under the promise of isolation for these classes. These results for hierarchical PFAs subsume [8], and are new for polynomial ambiguous PFAs. Our results also show the existence of a large class of non-trivial PFAs that exhibit exponential ambiguity but are nonetheless still leak monotonic and hence regular approximable; Theorem 21 gives a method of obtaining such PFAs (Figure 2a shows such a PFA \mathcal{A}_z). In this paper, we also show that the emptiness problem is undecidable for linearly ambiguous PFAs, thus resolving an open problem posed in [16], and tightening the decidability results presented in [16]. Another tractable class of PFAs is that of eventually weakly ergodic PFAs [10]. We show that though eventually weakly ergodic PFAs are not leak monotonic, they are effectively regular-approximable. Once again, as a consequence, the decidability results proved in [10] follow from observations made here.

The rest of the paper is organized as follows. We conclude this section with a discussion of closely related work. Basic definitions and notations are introduced in Section 2. Regular-approximability is defined and the undecidability of deciding if a PFA is regular-approximable with respect to (x, y) is proved in Section 3. Next, in Section 4, we give the semantic definition of leak monotonicity, its relation to regular-approximability, and its application to computing the value and deciding the emptiness problem. Section 5 presents results establishing the regular-approximability of hierarchical PFAs and polynomially ambiguous PFAs, and Section 6 shows that eventually weakly ergodic PFAs are also regular-approximable. Conclusions are presented in Section 7. All missing proofs can be found in the Appendix.

Related Work

The problem of checking whether the language recognized by a PFA is regular is known to be undecidable [17, 4]. As mentioned above, regular-approximability of PFAs was first studied in [24], where Paz gave an alternate, semantic characterization of regular-approximable PFAs. We are not aware of any further work on this topic in the context of PFAs, though

separation using regular languages has been used to obtain expressiveness and decidability results [12, 25]. \natural -acyclic automata and their generalization *leak-tight automata* [15, 14], are special classes of PFAs for which the value 1 problem is decidable. The classes of leaktight and leak monotonic automata (introduced in this paper) are incomparable – PFA \mathcal{A}_3 in Figure 1c on page 7 is leaktight but not leak monotonic. On the other hand, consider any PFA \mathcal{A} that is not leaktight, and let \mathcal{B} be PFA that is identical to \mathcal{A} , but with an empty set of final states. \mathcal{B} is still not leaktight, but \mathcal{B} is trivially leak monotonic (and hence regular approximable). The relationship between \natural -acyclic automata/leaktight automata and regular-approximable automata still needs further investigation. In particular, it is open whether \natural -acyclic and leaktight automata are a subclass of regular approximable automata. Bounding the ambiguity of PFAs as been a way to identify subclasses of PFAs for which certain computational problems become decidable [6, 8, 16]. However, all these results only pertain to automata with *constant* ambiguity and their subclasses. In this paper, we obtain positive results for more general classes of PFAs that go beyond polynomially ambiguous automata. The undecidability of the emptiness problem for linearly ambiguous automata was also independently observed in [13].

2 Preliminaries

We assume that the reader is familiar with probability distributions, stochastic matrices, finite-state automata, and regular languages. The set of natural numbers will be denoted by \mathbb{N} , the closed unit interval by $[0, 1]$ and the open unit interval by $(0, 1)$. The power-set of a set X will be denoted by 2^X . For any function $f: X \rightarrow Y$ and $Y_1 \subseteq Y$, $f^{-1}(Y_1)$ is the set $\{x \in X \mid f(x) \in Y_1\}$. If X is a finite set $|X|$ will denote its cardinality. We assume that the reader is familiar with the arithmetic hierarchy.

Sequences. Given a finite sequence $s = s_0s_1 \dots$ over S , $|s|$ will denote the length of s and $s[i]$ will denote the i th element s_i of the sequence with $s[0]$ being the first. We will use λ to denote the (unique) empty string/sequence. For natural numbers i, j , $i \leq j < |s|$, $s[i : j]$ is the sequence $s_i \dots s_j$. As usual S^* will denote the set of all finite sequences/strings/words over S , S^+ will denote the set of all finite non-empty sequences/strings/words over S .

Given $u \in S^*$ and $v \in S^*$, uv is the sequence obtained by concatenating the two sequences in order. Given $L_1 \subseteq S^*$ and $L_2 \subseteq S^*$, the set L_1L_2 is defined to be $\{uv \mid u \in L_1 \text{ and } v \in L_2\}$.

Ambiguity and Pumping Lemma

Let \mathcal{A} be a nondeterministic automaton recognizing a regular language over alphabet Σ . The degree of ambiguity [22, 21, 27] of \mathcal{A} on input word $u \in \Sigma^*$, denoted $d_{\mathcal{A}}(u)$, is the number of accepting runs of \mathcal{A} on u . It is shown in [28, 20] that the degree of ambiguity of a NFA \mathcal{A} is one the following.

1. \mathcal{A} is *finitely ambiguous* if there is a constant k such that $d_{\mathcal{A}}(u) \leq k$ for all input words $u \in \Sigma^*$.
2. \mathcal{A} is *polynomially ambiguous* if there is a non-constant polynomial $p: \mathbb{N} \rightarrow \mathbb{N}$ such that $d_{\mathcal{A}}(u) \leq p(|u|)$ for all all input words $u \in \Sigma^*$; if p has degree 1 or 2 then \mathcal{A} is said to be linearly or quadratically ambiguous, respectively.
3. \mathcal{A} is *exponentially ambiguous* if for every polynomial $p: \mathbb{N} \rightarrow \mathbb{N}$, there is a word $u \in \Sigma^*$ such that $d_{\mathcal{A}}(u) > p(|u|)$.

A *trim* NFA is an automaton that does not have any silent edges. The following can be concluded from the results of [28]:

► **Lemma 1.** *The problems of deciding whether a trim \mathcal{A} is finitely ambiguous, whether \mathcal{A} is polynomially ambiguous and whether \mathcal{A} is exponentially ambiguous are decidable in polynomial time. If \mathcal{A} is polynomially ambiguous then a constant C and a constant ℓ can be computed in polynomial time such that $d_{\mathcal{A}}(u) \leq C|u|^\ell$ for all input words $u \in \Sigma^*$.*

The following lemma, used in parts of the paper, states a simple property of regular languages and is easily proved along the same lines as the standard pumping lemma.

► **Lemma 2.** *For a regular language $L \subseteq \Sigma^*$, where $|\Sigma| \geq 2$, there exists an integer constant $N > 0$ such that the following property holds for each $a \in \Sigma$ and each $k \geq 1$: if there exists a string of the form $u_1 a u_2 a \dots u_k a \in L$, where each $u_i \in (\Sigma \setminus \{a\})^*$, for $1 \leq i \leq k$, then there exists such a string such that $|u_i| \leq N$, for each i , $1 \leq i \leq k$.*

Probabilistic automaton (PFA)

Informally, a PFA is like a finite-state deterministic automaton except that the transition function from a state on a given input is described as a probability distribution which determines the probability of the next state.

► **Definition 3.** A finite-state probabilistic automaton (PFA) [26, 24] on finite strings over a finite alphabet Σ is a tuple $\mathcal{A} = (Q, q_s, \delta, Q_f)$ where Q is a finite set of states, $q_s \in Q$ is the initial state, $\delta : Q \times \Sigma \times Q \rightarrow [0, 1]$ is the transition relation such that for all $q \in Q$ and $a \in \Sigma$, $\delta(q, a, q')$ is a rational number and $\sum_{q' \in Q} \delta(q, a, q') = 1$, and $Q_f \subseteq Q$ is the set of accepting/final states. We say that the PFA \mathcal{A} is a deterministic automaton if, for every $q \in Q, a \in \Sigma$ there exists exactly one $q' \in Q$ such that $\delta(q, a, q') = 1$.

► **Notation.** The transition function δ of PFA \mathcal{A} on input a can be seen as a square matrix δ_a of order $|Q|$ with the rows labeled by “current” state, columns labeled by “next state” and the entry $\delta_a(q, q')$ equal to $\delta(q, a, q')$. Given a word $u = a_0 a_1 \dots a_n \in \Sigma^+$, δ_u is the matrix product $\delta_{a_0} \delta_{a_1} \dots \delta_{a_n}$. For the empty word $\lambda \in \Sigma^*$ we take δ_λ to be the identity matrix. Finally for any $Q_0 \subseteq Q$, we say that $\delta_u(q, Q_0) = \sum_{q' \in Q_0} \delta_u(q, q')$. Given a state $q \in Q$ and a word $u \in \Sigma^+$, $\text{post}(q, u) = \{q' \mid \delta_u(q, q') > 0\}$. For a set $C \subseteq Q$, let $\text{post}(C, u) = \cup_{q \in C} \text{post}(q, u)$.

Intuitively, the PFA starts in the initial state q_s and if after reading a_0, a_1, \dots, a_i it is in state q , then the PFA moves to state q' with probability $\delta_{a_{i+1}}(q, q')$ on symbol a_{i+1} . A run of the PFA \mathcal{A} starting in a state $q \in Q$ on an input $u \in \Sigma^*$ is a sequence $\rho \in Q^*$ such that $|\rho| = 1 + |u|$, $\rho[0] = q$ and for each $i \geq 0$, $\delta_{u[i]}(\rho[i], \rho[i+1]) > 0$. The probability measure of such a run ρ on u is defined to be the value $\prod_{0 \leq i < |\rho|} \delta_{u[i]}(\rho[i], \rho[i+1])$. We say that the run ρ is an accepting run if $\rho[|\rho|] \in Q_f$, i.e., it ends in an accepting state. Unless otherwise stated, a run for us will mean a run starting in the initial state q_s . The probability of acceptance of $u \in \Sigma^*$ by the PFA \mathcal{A} , denoted by $P_{\mathcal{A}}(u)$, is defined to be the sum of probability measures of all accepting runs of \mathcal{A} on u . Note that $P_{\mathcal{A}}(u) = \delta_u(q_s, Q_f)$.

PFA languages

Given a PFA \mathcal{A} , a rational threshold $x \in [0, 1]$ and $\diamond \in \{<, \leq, =, \geq, >\}$, the language $L_{\diamond x}(\mathcal{A}) = \{u \in \Sigma^* \mid P_{\mathcal{A}}(u) \diamond x\}$ is the set of finite words accepted by \mathcal{A} with probability $\diamond x$. If \mathcal{A} is a deterministic automaton then we let $L(\mathcal{A})$ denote the language $L_{\geq 1}(\mathcal{A})$. In general, the language $L_{\diamond x}(\mathcal{A})$ for a PFA \mathcal{A} , threshold x , and $\diamond \in \{<, \leq, =, \geq, >\}$, may be non-regular. However, when x is an extremal threshold ($x \in \{0, 1\}$), it is regular.

► **Proposition 4.** *For any PFA \mathcal{A} , the languages $L_{\diamond x}(\mathcal{A})$ is effectively regular for $x \in \{0, 1\}$ and $\diamond \in \{<, \leq, =, \geq, >\}$.*

14:6 Approximating PFAs by regular languages

Given a PFA \mathcal{A} and rational threshold x , the problem of checking whether $L_{>x}(\mathcal{A}) = \emptyset$ is known to be **co-R.E.**-complete [24, 11].

Isolated cut-points

For a PFA \mathcal{A} , a rational threshold $x \in [0, 1]$ is said to be an *isolated cut-point* of \mathcal{A} if there is an $\epsilon > 0$ such that for each word $u \in \Sigma^*$, $|\mathcal{P}_{\mathcal{A}}(u) - x| > \epsilon$. If such an ϵ exists, then ϵ is said to be a *degree of isolation*. An important observation about PFAs with isolated cut-points, is that their language is regular; however, the deterministic finite automaton recognizing this language is known to be constructible only given a degree of isolation.

► **Theorem 5** (Rabin [26]). *For any PFA \mathcal{A} with an isolated cut-point x , the languages $L_{\diamond x}(\mathcal{A})$ are regular, where $\diamond \in \{<, \leq, =, \geq, >\}$.*

The *isolation decision problem* is the problem of deciding for a given PFA \mathcal{A} and a rational $x \in [0, 1]$ whether x is an isolated cut-point of \mathcal{A} . The isolation decision problem is known to be undecidable [3], even when x is 0 or 1 [18]. The problem is known to be Σ_2^0 -complete [10].

The value problem. For a PFA \mathcal{A} , let $\text{value}(\mathcal{A})$ denote the least upper bound of the set $\{\mathcal{P}_{\mathcal{A}}(u) \mid u \in \Sigma^*\}$. The *value computation problem* for a PFA is the problem of computing $\text{value}(\mathcal{A})$ for a given \mathcal{A} . The *value decision problem* is the problem of deciding for a given PFA \mathcal{A} and a rational threshold $x \in [0, 1]$ whether $\text{value}(\mathcal{A}) = x$. The value decision problem is known to be undecidable [3, 18] and known to be Π_2^0 -complete [10] even when x is taken to be 1 [10].

3 Approximability and Value problem

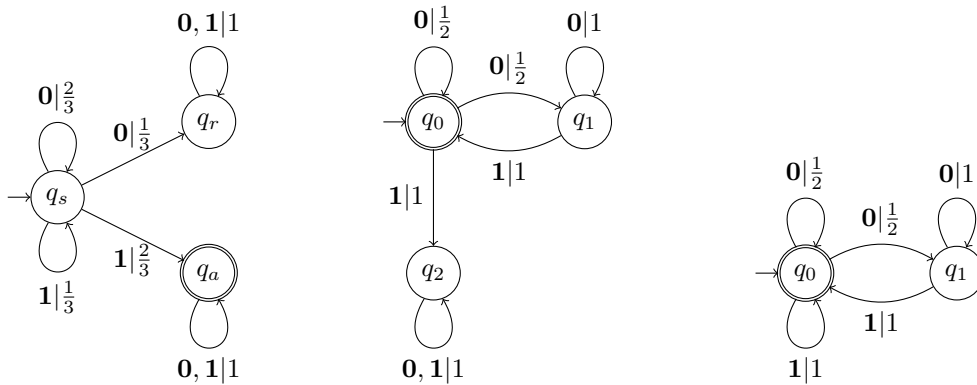
3.1 Regular Approximability.

The problem of approximating a PFA by a regular language was first discussed by Paz [24]. We will say that PFA \mathcal{A} can be approximated by a regular language L at a threshold x with precision y if L separates the languages $L_{\geq x+y}(\mathcal{A})$ and $L_{\leq x}(\mathcal{A})$. Formally,

► **Definition 6.** Given $x, y \in [0, 1]$ such that $y > 0$, a PFA $\mathcal{A} = (Q, q_s, \delta, Q_f)$ over Σ is said to be *regular-approximable* with respect to the pair (x, y) if there is a regular language L such that $L_{\geq x+y}(\mathcal{A}) \subseteq L \subseteq L_{>x}(\mathcal{A})$.

It is easy to see that \mathcal{A} is regular-approximable with respect to (x, y) if either $L_{>x}(\mathcal{A})$ or $L_{\geq x+y}(\mathcal{A})$ is a regular set. We say that the pair (x, y) , $x, y \in [0, 1]$, is a *trivial* pair if either $x = 0$ or $x + y \geq 1$. It is seen that every PFA is regular-approximable with respect to every trivial pair thanks to Proposition 4.

► **Example 7.** Consider the PFA \mathcal{A}_1 , shown in Figure 1a. It has been shown in [7] that both $L_{>\frac{1}{2}}(\mathcal{A}_1)$ and $L_{\geq\frac{1}{2}}(\mathcal{A}_1)$ are non-regular. Further, given this observation, we can also conclude that $L_{\geq\frac{3}{4}}(\mathcal{A}_1)$ is non-regular. This is because $L_{\geq\frac{1}{2}}(\mathcal{A}_1) = \mathbf{1}\{\mathbf{0}, \mathbf{1}\}^* \cup \mathbf{0}L_{\geq\frac{3}{4}}(\mathcal{A}_1)$. In spite of this, we can show that a regular language can separate $L_{\geq\frac{3}{4}}(\mathcal{A}_1)$ and $L_{\leq\frac{1}{2}}(\mathcal{A}_1)$, i.e., \mathcal{A}_1 is regular-approximable with respect to the pair $(\frac{1}{2}, \frac{1}{4})$. Observe that $L_{>\frac{2}{3}}(\mathcal{A}_1) = \mathbf{1}\{\mathbf{0}, \mathbf{1}\}^*$ is a regular set. Since $L_{\geq\frac{3}{4}}(\mathcal{A}_1) \subseteq L_{\geq\frac{2}{3}}(\mathcal{A}_1) \subseteq L_{>\frac{1}{2}}(\mathcal{A}_1)$, we can conclude that \mathcal{A}_1 is regular-approximable with respect to the pair $(\frac{1}{2}, \frac{1}{4})$. In fact, as we will show later, \mathcal{A}_1 is regular-approximable with respect to every pair (x, y) where $y > 0$.



■ **Figure 1** On the left (a) is PFA \mathcal{A}_1 , in the middle (b) is PFA \mathcal{A}_2 , and on the right (c) is PFA \mathcal{A}_3 . In these pictures, for states q and q' and input letter a , if $\delta(q, a, q') > 0$ then we label the edge from q to q' by $a|\delta(q, a, q')$. The initial state is indicated by a dangling \rightarrow and the final state by two concentric circles.

While \mathcal{A}_1 is an example of a PFA that is regular-approximable with respect to every pair (x, y) such that $y > 0$, the following theorem shows the existence of a PFA that is not regular-approximable with respect to any non-trivial pair.

► **Theorem 8.** *There exists a PFA \mathcal{A} that is not regular-approximable with respect to any pair (x, y) such that $x, y > 0$ and $x + y < 1$.*

Proof. We prove the theorem by construction. Consider the PFA \mathcal{A}_2 over the input alphabet $\Sigma = \{0, 1\}$, shown in Figure 1b. This automaton was used in [1] to show that the language recognized by a Probabilistic Büchi automaton (PBA) with threshold 0 can be nonregular.

We make the following observations, which are easily seen. Every word starting with 1 or that contains two consecutive 1s is accepted by \mathcal{A}_2 with probability zero. For every $k > 0$ and every $z, 0 < z < 1$, there is a word in $(0^*1)^k$ that is accepted with probability $\geq z$.

Consider any pair (x, y) such that $x, y > 0$ and $x + y < 1$. We show that \mathcal{A}_2 is not regular-approximable with respect to (x, y) , by contradiction. Assume for contradiction, that there is a regular language L such that $L_{\geq x+y}(\mathcal{A}_2) \subseteq L \subseteq L_{>x}(\mathcal{A}_2)$. Since L is a regular language, let N be the constant satisfying Lemma 2. Now, let $k \in \mathbb{N}$ be any integer such that $(1 - \frac{1}{2^N})^k \leq x$. Such a k exists since $x > 0$. From our earlier observation, we see that there exists a string $u \in (0^*1)^k$ that is in $L_{\geq x+y}(\mathcal{A}_2)$. Clearly, $u \in L$. Now, from Lemma 2, we see that there exists a string $v = 0^{n_1}10^{n_2}1 \dots 0^{n_k}1$ where $n_i \leq N$, for $1 \leq i \leq k$ such that $v \in L$. Word v is accepted by \mathcal{A}_2 , with probability $\prod_{1 \leq i \leq k} (1 - \frac{1}{2^{n_i}})$. Since each $n_i \leq N$, we have $(1 - \frac{1}{2^{n_i}}) \leq (1 - \frac{1}{2^N})$. From this we see that the probability of acceptance of v by \mathcal{A}_2 is $\leq (1 - \frac{1}{2^N})^k \leq x$. Hence $v \notin L_{>x}(\mathcal{A}_2)$ which contradicts our assumption that $L \subseteq L_{>x}(\mathcal{A}_2)$. ◀

The following theorem shows that the problem of checking if a given PFA \mathcal{A} is regular-approximable with respect to a given pair (x, y) is undecidable.

► **Theorem 9.** *Given a PFA \mathcal{A} and rational values $x, y \in [0, 1]$, the problem of checking if \mathcal{A} is approximable with respect to (x, y) , is undecidable. Formally the language $\mathbf{Approx} = \{(\mathcal{A}, x, y) \mid x, y \in [0, 1], \mathcal{A} \text{ is a PFA that is regular-approximable w.r.t. } (x, y)\}$ is undecidable.*

3.2 Value Problem and Emptiness under isolation

PFAs that are effectively regular-approximable for every pair (x, y) enjoy nice properties.

► **Definition 10.** We say that \mathcal{A} is *regular-approximable* if it is regular-approximable with respect to every pair (x, y) such that $x, y \in [0, 1]$ and $y > 0$. We further say that \mathcal{A} is *effectively regular-approximable* if there is a procedure that, given x and y terminates and outputs a deterministic automaton that accepts a language L where $L_{\geq x+y}(\mathcal{A}) \subseteq L \subseteq L_{> x}(\mathcal{A})$. A class \mathcal{C} of regular-approximable PFAs is said to be *effectively regular-approximable* if there is a procedure that, given $\mathcal{A} \in \mathcal{C}$, x and y terminates and outputs a deterministic automaton that accepts a language L where $L_{\geq x+y}(\mathcal{A}) \subseteq L \subseteq L_{> x}(\mathcal{A})$.

We shall establish later that the class of hierarchical probabilistic automata (HPAs) is effectively regular-approximable (See Theorem 26). It has been shown in [5, 8, 2] that the emptiness problem and the value decision problem continues to be undecidable if we restrict our attention to HPAs. Thus, there is no algorithm that given an effectively regular-approximable PFA \mathcal{A} computes its value. Nevertheless, we now show that if \mathcal{A} is effectively regular-approximable then its value can be computed to a given precision.

► **Theorem 11.** *There is a procedure `ComputeVal` that given an effectively regular-approximable PFA \mathcal{A} and $\epsilon > 0$ terminates and returns an interval $[z_1, z_2]$ such that $\text{value}(\mathcal{A}) \in [z_1, z_2]$ and $z_2 - z_1 \leq \epsilon$.*

Proof. `ComputeVal` works as follows. Initially, it checks if there is u such that $P_{\mathcal{A}}(u) = 1$ or if for every u , $P_{\mathcal{A}}(u) = 0$. If either of these conditions hold then it returns the corresponding value as $\text{value}(\mathcal{A})$. Observe that these conditions can be checked thanks to Proposition 4. If neither of these conditions holds, it acts as follows. It maintains two variables z_1, z_2 such that $0 \leq z_1 < z_2 \leq 1$ and $\text{value}(\mathcal{A}) \in [z_1, z_2]$. Initially z_1, z_2 are set to 0, 1 respectively.

The following procedure is iterated until $z_2 - z_1 \leq \epsilon$. In each iteration, it first computes a deterministic automaton \mathcal{B} such that $L_{\geq x+y}(\mathcal{A}) \subseteq L(\mathcal{B}) \subseteq L_{> x}(\mathcal{A})$ where $x = z_1 + \frac{z_2 - z_1}{3}$ and $y = \frac{z_2 - z_1}{3}$. Such an automaton \mathcal{B} can be computed since \mathcal{A} is effectively regular-approximable. (Observe that both $x + y - z_1$ and $z_2 - x$ are equal to $\frac{2}{3}(z_2 - z_1)$.) Now, the algorithm checks if $L(\mathcal{B}) = \emptyset$. If $L(\mathcal{B}) = \emptyset$ then this implies $L_{\geq x+y}(\mathcal{A}) = \emptyset$ and hence $\text{value}(\mathcal{A})$ lies in the interval $[z_1, x + y]$; in this case, it repeats the above procedure by setting $z_2 = x + y$ and keeping z_1 unchanged. On the other hand, if $L(\mathcal{B}) \neq \emptyset$, then this implies that $\text{value}(\mathcal{A})$ lies in the interval $[x, z_2]$; so, in this case the algorithm sets $z_1 = x$, keeps z_2 unchanged and repeats the above procedure.

Notice that the length of the interval (z_1, z_2) at the beginning of each succeeding iteration is $\frac{2}{3}$ rd of its value at the beginning of the preceding iteration; further, at the beginning of the first iteration, its value is 1. From this we see that this algorithm terminates after k iterations where k is the least value such that $(\frac{2}{3})^k \leq \epsilon$, that is, $k = \lceil \log_{\frac{2}{3}}(\frac{1}{\epsilon}) \rceil$. From our arguments, we see that at the beginning of each iteration, we have $\text{value}(\mathcal{A}) \in (z_1, z_2)$ and when it terminates $z_2 - z_1 \leq \epsilon$. Thus, it returns an interval in which $\text{value}(\mathcal{A})$ lies and its length is at most ϵ . Observe that, in the above procedure, we only need to check the emptiness of $L(\mathcal{B})$ in each iteration; no explicit computation of \mathcal{B} is needed. ◀

An immediate consequence of the above observation is that if \mathcal{A} is effectively regular-approximable and x is an isolated cut-point of \mathcal{A} , then we can check the emptiness of $L_{> x}(\mathcal{A})$.

► **Corollary 12.** *There is a procedure `IsoEmpty` that given an effectively regular-approximable PFA \mathcal{A} and a threshold x such that x is an isolated cut-point of \mathcal{A} , terminates and decides if $L_{> x}(\mathcal{A}) = \emptyset$.*

Proof. Observe that \mathcal{A} is isolated at x with a degree of isolation ϵ_0 then either $\text{value}(\mathcal{A}) \geq x + \epsilon_0$ or $\text{value}(\mathcal{A}) \leq x - \epsilon_0$. `IsoEmpty` works iteratively as follows. Initially it sets $\epsilon = \frac{1}{2}$ and uses the algorithm `ComputeVal` in Theorem 11 to compute $[z_1, z_2]$ such that $\text{value}(\mathcal{A}) \in [z_1, z_2]$ and $z_2 - z_1 \leq \epsilon$. If $x \in [z_1, z_2]$ then it sets $\epsilon = \frac{\epsilon}{2}$ and repeats. Otherwise if $z_1 > x$ then it returns 1 and if $z_2 < x$ then it returns 0. It is easy to see that `IsoEmpty` always returns the correct answer and terminates when ϵ takes a value $< \epsilon_0$. ◀

4 Leak monotonicity and complexity

We shall now identify a *semantic* class of PFAs that are regular-approximable. Our proof of the fact that polynomial ambiguous automata are regular-approximable shall be established by showing that they belong to this class. In order to define these classes, we shall need the concept of a *leak*. Intuitively, a leak happens at a position i in a run $q_0q_1 \dots q_n$ of \mathcal{A} on input u if the probability of transitioning from q_i to q_{i+1} is non-zero and yet is less than 1.

► **Definition 13.** Consider a PFA $\mathcal{A} = (Q, q_s, \delta, Q_f)$ over an alphabet Σ . We say that a triple (q, a, q') , where $q, q' \in Q$ and $a \in \Sigma$, is a *leaky* transition of \mathcal{A} if $0 < \delta(q, a, q') < 1$. Let $u \in \Sigma^*$ be a finite word and ρ be a run of \mathcal{A} on u . We let $\text{NbrLeaks}(\mathcal{A}, u, \rho)$ to be the number of leaky transitions in ρ with respect to the word u ; formally, it is $|\{i \mid 0 \leq i < |\rho|, \delta(\rho[i], u[i], \rho[i+1]) < 1\}|$.

4.1 Leak Monotonicity

The class of PFAs that we will be interested in are PFAs in which the measure of accepting a word is concentrated mostly in runs with a few leaks. We formalize this intuition below:

► **Definition 14.** Let $\epsilon \in (0, 1)$ be a rational number. We say that \mathcal{A} is ϵ -*leak monotonic* if there exists some $N_\epsilon \in \mathbb{N}$ such that for all $u \in \Sigma^*$, the measure of accepting runs of \mathcal{A} on u having at least N_ϵ leaks is strictly less than ϵ . Such an N_ϵ will be called a horizon of ϵ -leak monotonicity of \mathcal{A} .

► **Example 15.** The PFA \mathcal{A}_1 in Figure 1a on page 7, can be shown to be ϵ -leak monotonic by taking N_ϵ to be any integer n such that $(\frac{2}{3})^n \leq \epsilon$. In contrast, the PFA \mathcal{A}_2 in Figure 1b is not ϵ -leak monotonic for any $\epsilon \in (0, 1)$. This is an immediate consequence of Theorem 8 and Theorem 16 established below.

The following theorem connects ϵ -leak monotonicity with regular-approximability.

► **Theorem 16.** *If \mathcal{A} is a PFA over an alphabet Σ which is ϵ -leak monotonic then \mathcal{A} is regular-approximable with respect to every pair (x, ϵ) , for $x \in [0, 1]$ and $\epsilon > 0$.*

Proof. Let PFA $\mathcal{A} = (Q, q_s, \delta, Q_f)$ over alphabet Σ be ϵ -leak monotonic. Let $N \in \mathbb{N}$ be an integer such that $\forall u \in \Sigma^*$, the probability measure, of all accepting runs of \mathcal{A} on u having at least N leaks, is at most ϵ . Let $x \in [0, 1]$. Now, we give the construction of a deterministic automaton \mathcal{B} on alphabet Σ such that $L_{\geq x+\epsilon}(\mathcal{A}) \subseteq L(\mathcal{B}) \subseteq L_{>x}(\mathcal{A})$.

Without loss of generality, let $Q = \{q_0, q_1, \dots, q_{n-1}\}$ with the start state $q_s = q_0$. For any $u \in \Sigma^*$, let LeakPr_u be a $n \times N$ matrix such that, for $0 \leq i < n$ and $0 \leq j < N$, $\text{LeakPr}_u(i, j)$ is the probability measure of all runs ρ of \mathcal{A} on input u starting from q_0 , ending in state q_i and having exactly j leaky transitions, i.e., $\text{NbrLeaks}(\mathcal{A}, u, \rho) = j$.

Consider the automaton (not necessarily finite) $\mathcal{B} = (R, r_0, \delta', R_f)$ where $R = \{\text{LeakPr}_u \mid u \in \Sigma^*\}$; r_0 is the matrix such that $r_0(0, 0) = 1$ and $r_0(i, j) = 0$ when $i \neq 0$ or $j \neq 0$; $R_f = \{r \mid (\sum_{i:q_i \in Q_f} \sum_{0 \leq j < N} r(i, j)) > x\}$. We define δ' as follows. Let $r \in R$ and $a \in \Sigma$.

14:10 Approximating PFAs by regular languages

By definition, there exists $u \in \Sigma^*$ such that $r = \text{LeaksPr}_u$. Let $r' = \text{LeaksPr}_{ua}$. Fix any i, j such that $0 \leq i < n$ and $0 \leq j < N$. Let p_1 be the sum of all $r(i', j)$ such that $\delta(q_{i'}, a, q_i) = 1$, i.e., the transition $(q_{i'}, a, q_i)$ is not a leaky transition of \mathcal{A} . Let p_2 be a value defined as follows: if $j = 0$ then $p_2 = 0$, otherwise p_2 is the sum of $r(i', j-1) \cdot \delta(q_{i'}, a, q_i)$ where the sum is taken over all i' such that $\delta(q_{i'}, a, q_i) < 1$, i.e., $(q_{i'}, a, q_i)$ is a leaky transition of \mathcal{A} . It is easily shown that $r'(i, j) = p_1 + p_2$. We call r' as the a -successor of r . Observe that the values p_1, p_2 for a given pair i, j are independent of u and hence, the relationship between r, r' , as given above, is independent of u . This leads us to the following definition of δ' . We define δ' so that $\delta'(r, a, r') = 1$ iff r' is the a -successor of r . Now, by induction on $|u|$, we can easily show that, for any $r \in R$, $\delta'_u(r_0, r) = 1$ iff $r = \text{LeaksPr}_u$.

Now, we show that R is a finite set and bound its size. Let D be the maximum of the denominators of the non-zero transition probabilities of \mathcal{A} . The probability of any run of \mathcal{A} , on some input, having less than N leaks is a rational number $\frac{x'}{y'}$ where y' is a positive integer such that $y' \leq D^N$. For any state $r \in R$ and for any i, j , $0 \leq i < n, 0 \leq j < N$, the value of the entry $r(i, j)$ is the sum of the probabilities of some runs of \mathcal{A} each having fewer than N leaks; the least common multiple of the denominators of these probabilities is bounded by $D^{N \cdot D^N}$. Hence $r(i, j)$ is either zero, or is a rational number whose denominator is bounded by $D^{N \cdot D^N}$. This implies that the number of distinct values $r(i, j)$ can take is bounded by $1 + (D^{N \cdot D^N})^2 = 1 + D^{2N \cdot D^N}$. Since r has $n \cdot N$ such entries, we see that $|R|$, which is the number of distinct values r can take, is bounded by $(1 + D^{2N \cdot D^N})^{n \cdot N}$ and hence is finite.

Now we show that $L_{\geq x+\epsilon}(\mathcal{A}) \subseteq L(\mathcal{B}) \subseteq L_{>x}(\mathcal{A})$. Consider any $u \in \Sigma^*$. The set of accepting runs of \mathcal{A} on u can be partitioned into two sets X_1, X_2 which are, respectively, the sets of runs having less than N leaks, or having at least N leaks. Let z_1, z_2 , respectively, be the probability measures of these two sets of runs. Clearly, $P_{\mathcal{A}}(u) = z_1 + z_2$. Based on the value of N , we have $z_2 \leq \epsilon$. Suppose that r is the unique state in R such that $\delta'_u(r_0, r) = 1$. Then, from our earlier observations, we see that $\sum_{i: q_i \in Q_f} \sum_{j < N} r(i, j) = z_1$. If $u \in L_{\geq x+\epsilon}(\mathcal{A})$ then $z_1 > x$ since $z_2 < \epsilon$, and from the definition of R_f , it follows that $r \in R_f$ and $u \in L(\mathcal{B})$. Thus, we see that $L_{\geq x+\epsilon}(\mathcal{A}) \subseteq L(\mathcal{B})$. If $u \in L(\mathcal{B})$ then, from the definition of R_f , we have $z_1 > x$ and hence $u \in L_{>x}(\mathcal{A})$. Thus, we see that $L(\mathcal{B}) \subseteq L_{>x}(\mathcal{A})$. \blacktriangleleft

► **Remark.** The deterministic automaton \mathcal{B} that we construct for an ϵ -leak monotonic PFA \mathcal{A} in the proof of Theorem 16 has the following property: for each input string u , the state r that is reached in \mathcal{B} on input u , starting from its initial state, gives the probability of acceptance of u by \mathcal{A} with precision ϵ . Equivalently, there is a function f from the states of \mathcal{B} to $[0, 1]$ such that $f(q) \leq P_{\mathcal{A}}(u) < f(q) + \epsilon$. $f(q)$ can be computed in time polynomial in the size of the representation of q . The above observations imply that the value of \mathcal{A} lies in the interval $[v, v + \epsilon]$ where $v = \max f(q)$. Thus, if \mathcal{B} can be constructed then value of \mathcal{A} can be approximated within ϵ .

However, there are regular-approximable PFAs that are not ϵ -leak monotonic for any ϵ .

► **Proposition 17.** *There is a PFA \mathcal{A} that is regular-approximable but not ϵ -leak monotonic for any $\epsilon \in (0, 1)$.*

Proof. Consider the PFA \mathcal{A}_3 shown in Figure 1c on page 7. Given $x \in (0, 1)$, let n_x be the largest integer such that $\frac{1}{2}^{n_x} > x$. It is easy to see that $L_{>x}(\mathcal{A}_3) = \lambda + \{\mathbf{0}, \mathbf{1}\}^* \mathbf{1} (\lambda + \mathbf{0} + \mathbf{0}^2 + \dots \mathbf{0}^{n_x})$ where λ is the empty word. Thus, $L_{>x}(\mathcal{A}_3)$ is regular for each x and hence regular-approximable. Furthermore, observe that for each n , the word $u_n = (\mathbf{01})^n$ is accepted by \mathcal{A}_3 with probability 1. In addition, for each n , u_n has exactly 2^n runs, each of which is accepting and has exactly n leaks. From these observations, it is easy to see that \mathcal{A}_3 is not ϵ -leak monotonic for any ϵ – for every possible horizon N_ϵ there are infinitely many words such that the measure of accepting runs having at least N_ϵ leaks is 1. \blacktriangleleft

The following theorem shows that the problem of checking if a given PFA is ϵ -leak monotonic with respect to given $\epsilon \in (0, 1)$ is undecidable.

► **Theorem 18.** *Given a PFA \mathcal{A} and a rational value $\epsilon \in (0, 1)$, the problem of checking if \mathcal{A} is ϵ -leak monotonic is undecidable. Formally the set $\text{LeakMon} = \{(\mathcal{A}, \epsilon) \mid \epsilon \in (0, 1), \mathcal{A} \text{ is a PFA that is } \epsilon\text{-leak monotonic}\}$ is undecidable.*

It is easy to see that we can give a simple algorithm that takes as input \mathcal{A}, x, N and constructs the deterministic automaton \mathcal{B} defined in the proof of Theorem 16. Such an algorithm starts with an initial set of states of \mathcal{B} which is taken to be r_0 and enlarges this set by choosing an unexplored state from it, and explores it by constructing and adding all its a -successors, that are not already present, to the set of states, for each $a \in \Sigma$. This algorithm terminates when no new states can be added. Hence if we can compute a horizon of ϵ -leak monotonicity of an ϵ -leak monotonic \mathcal{A} then we can compute the regular language that approximates $L_{>x}(\mathcal{A})$ for every threshold x .

► **Definition 19.** We say that a PFA \mathcal{A} is *leak monotonic* if \mathcal{A} is ϵ -leak monotonic with respect to every $\epsilon \in (0, 1)$. \mathcal{A} is said to be *effectively leak monotonic* if there is an algorithm that given ϵ outputs a horizon of ϵ -leak monotonicity of \mathcal{A} . A class \mathcal{C} of leak monotonic PFAs is said to be *effectively leak monotonic* if there is a procedure that, given $\mathcal{A} \in \mathcal{C}$ and $\epsilon > 0$ terminates and outputs a horizon of ϵ -leak monotonicity of \mathcal{A} .

The PFA \mathcal{A}_1 given in Figure 1a on page 7 is leak monotonic. We have the following as a consequence of Theorem 16.

► **Corollary 20.** *If a PFA is (effectively) leak monotonic then it is (effectively) regular-approximable.*

The following theorem allows us to construct leak monotonic PFAs from smaller leak monotonic PFAs.

► **Theorem 21.** *If a PFA $\mathcal{A} = (Q, \delta, q_s, Q_f)$ over Σ is such that Q can be partitioned into sets Q_0, \dots, Q_m such that $q_s \in Q_0$ and the following conditions hold:*

1. *For each $i \geq 1, q \in Q_i$ and $a \in \Sigma$, $\text{post}(q, a) \subseteq Q_i$.*
2. *There is a constant $m > 0$ such that from every state in Q_0 and on every input u of length at least m , some state outside Q_0 is reachable, and*
3. *For $i > 0$, the restriction of \mathcal{A} to each Q_i , when started in any state $q \in Q_i$, is leak monotonic,*

then \mathcal{A} is leak monotonic.

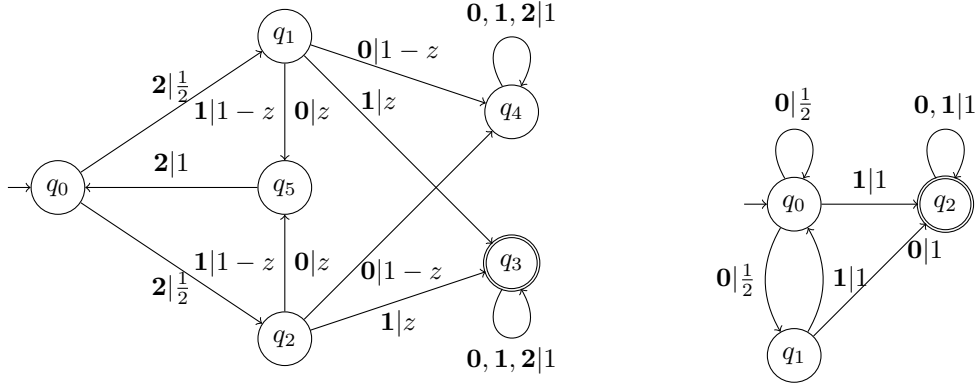
4.2 Leak Complexity

In this subsection, we introduce a *syntactic* class of PFAs that are leak monotonic. The syntactic class of PFAs shall be defined through the concept of *leak complexity* defined below.

► **Definition 22.** Let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a function. We say that the *leak complexity* of \mathcal{A} is given by f (or is simply f) if for all $u \in \Sigma^*$, for all $\ell \in \mathbb{N}$, the number of accepting runs of \mathcal{A} on u having exactly ℓ leaks is at most $f(\ell)$, i.e., $|\{\rho \mid \rho \text{ is an accepting run of } \mathcal{A} \text{ on } u \text{ and } \text{NbrLeaks}(\mathcal{A}, u, \rho) = \ell\}| \leq f(\ell)$.

Notice that we are only using the accepting runs to define the leak complexity. Further, observe that if f, g are functions from \mathbb{N} to \mathbb{N} such that $f(\ell) \leq g(\ell)$ for all $\ell \in \mathbb{N}$, and the leak complexity of \mathcal{A} is given by f , then its leak complexity is also given by g . We try to use the tightest function to specify the leak complexity of a PFA.

We shall be interested in PFAs whose leak complexity is given by special functions.



■ **Figure 2** Automaton \mathcal{A}_z on the left (a) and Automaton \mathcal{A}_5 on the right (b).

► **Definition 23.** Let $\mathcal{A} = (Q, \delta, q_s, Q_f)$ be a PFA.

- \mathcal{A} is said to have *polynomial leak complexity* if its leak complexity is given by a polynomial function h .
- For \mathcal{A} , let $MaxTrPr(\mathcal{A})$ be maximum probability of a leaky transition, i.e., the value $\max\{\delta(q, a, q') \mid 0 < \delta(q, a, q') < 1, q, q' \in Q, a \in \Sigma\}$. We say that \mathcal{A} has *sub-exponential leak complexity* if there exist constants $c, d > 0$ such that $d < \frac{1}{MaxTrPr(\mathcal{A})}$ and the leak complexity of \mathcal{A} is $c \cdot d^\ell$.

Clearly, if \mathcal{A} has polynomial leak complexity then it has sub-exponential leak complexity.

► **Example 24.** For the PFA \mathcal{A}_1 in Figure 1a on page 7, on any input, the number of accepting runs having ℓ leaks is at most 1 and hence its leak complexity is constant. Figure 2a shows a PFA \mathcal{A}_z over the input alphabet $\Sigma = \{0, 1, 2\}$ that has sub-exponential leak complexity, but not polynomial leak complexity. Here $z \in (0, 1)$ is a number that is left unspecified. In the figure, all unspecified transitions, from states q_0, q_1, q_2, q_5 , on the appropriate input symbols, go to the reject state q_4 with probability 1. Both q_3, q_4 are absorbing states in which q_3 is the accepting state. It is not difficult to see that all accepting runs of \mathcal{A}_z on an input word have an even number of leaks. Furthermore, for an even ℓ , the number of accepting runs having ℓ leaks is exactly $2^{\frac{\ell}{2}}$, i.e., $(\sqrt{2})^\ell$. Observe that $MaxTrPr(\mathcal{A}_z) = z$ if $z > \frac{1}{2}$ else it is $1 - z$. Hence, \mathcal{A}_z has subexponential leak complexity iff $1 - \frac{1}{\sqrt{2}} < z < \frac{1}{\sqrt{2}}$. Thus, for example, \mathcal{A}_z has sub-exponential leak complexity if $z = \frac{2}{3}$. On the other hand \mathcal{A}_z does not have subexponential leak complexity if $z = \frac{3}{4}$. However, note that \mathcal{A}_z is leak monotonic for each $z \in (0, 1)$ as \mathcal{A}_z satisfies conditions of Theorem 21 with $m = 2, Q_0 = \{q_0, q_1, q_5, q_2\}, Q_1 = \{q_4\}$ and $Q_2 = \{q_3\}$.

We show that every PFA that has sub-exponential leak complexity is leak monotonic.

► **Theorem 25.** *If a PFA \mathcal{A} over an alphabet Σ has sub-exponential leak complexity then \mathcal{A} is leak monotonic and hence regular-approximable.*

Proof. Let $\mathcal{A} = (Q, q_s, \delta, Q_f)$ be a PFA over alphabet Σ with sub-exponential leak complexity. This means, there exist constants $c, d > 0$ such that $d < \frac{1}{MaxTrPr(\mathcal{A})}$ and the leak complexity of \mathcal{A} is $c \cdot d^\ell$, i.e. on every word $u \in \Sigma^*$ the number of accepting runs of \mathcal{A} on u having ℓ leaks is bounded by $c \cdot d^\ell$. We prove the theorem by showing that \mathcal{A} is leak monotonic and appealing to Corollary 20. Let $\epsilon \in [0, 1]$ be such that $\epsilon > 0$. Let $p = d \cdot MaxTrPr(\mathcal{A})$.

Observe that $0 < p < 1$ since $d < \frac{1}{\text{MaxTrPr}(\mathcal{A})}$. Now, let $N \in \mathbb{N}$ be the smallest integer such that

$$N > \frac{\log\left(\frac{c}{\epsilon \cdot (1-p)}\right)}{\log \frac{1}{p}} \quad (1)$$

Consider any $u \in \Sigma^*$. Let z_2 be the probability measure of accepting runs of \mathcal{A} having at least N leaks. The probability of any single run having ℓ leaks is bounded by $(\text{MaxTrPr}(\mathcal{A}))^\ell$. Since there are at most $c \cdot d^\ell$ accepting runs of \mathcal{A} on u having ℓ leaks, we see that $z_2 \leq \sum_{\ell \geq N} c \cdot d^\ell \cdot (\text{MaxTrPr}(\mathcal{A}))^\ell$. Using $p = d \cdot \text{MaxTrPr}(\mathcal{A})$, we have

$$z_2 \leq \sum_{\ell \geq N} c \cdot p^\ell = c \cdot p^N \cdot \sum_{\ell \geq 0} p^\ell.$$

From this we see that $z_2 \leq c \cdot p^N \cdot \frac{1}{1-p}$. Now using inequality (1) and raising both its two sides to the power of 2, after simplification, we get $\left(\frac{1}{p}\right)^N > \frac{c}{\epsilon \cdot (1-p)}$, which leads to $\epsilon > p^N \cdot \frac{c}{1-p} \geq z_2$. Hence, we see that \mathcal{A} is ϵ -leak monotonic. Clearly this holds for every $\epsilon \in [0, 1]$ such that $\epsilon > 0$. Hence \mathcal{A} is leak monotonic. \blacktriangleleft

Observe that the proof of Theorem 25 also shows that if the (sub-exponential) leak complexity function of \mathcal{A} is known (or can be computed) then \mathcal{A} is effectively regular-approximable. Theorem 25 can be used to identify classes of PFAs that are leak monotonic. In conjunction with Theorem 21 and Theorem 16, it can be used to identify regular-approximable PFAs. We conclude by showing that the class of Hierarchical PFAs (HPA)s (introduced in [9, 6]) is effectively leak monotonic.

Hierarchical PFAs (HPA)s

(HPAs), introduced in [9, 6], are defined as follows. A k -HPA \mathcal{A} on Σ is a probabilistic automaton whose states can be partitioned into $k + 1$ levels Q_0, Q_1, \dots, Q_k such that for any state q and input symbol $a \in \Sigma$, at most one successor state is at the same level, and others are higher level states. In other words for each $q \in Q_i$ and $a \in \Sigma$, $\text{post}(q, a) \subseteq Q_i \cup Q_{i+1} \cdots \cup Q_k$ and $|\text{post}(q, a) \cap Q_i| \leq 1$. Without loss of generality, we can assume that the initial state is at level 0. The following theorem shows that the class of HPAs are effectively leak monotonic and hence regular-approximable.

► **Theorem 26.** *Every k -HPA \mathcal{A} with n -states and $k > 0$, has leak complexity at most $n^k \ell^{k-1}$. Hence, the class of hierarchical probabilistic automata is effectively leak monotonic and hence regular-approximable.*

► **Example 27.** The automaton \mathcal{A}_1 in Figure 1a on page 7 is a 1-HPA whose leak complexity is 1. Automaton \mathcal{A}_z in Figure 2a on page 12 is not a HPA.

Thanks to Theorem 11 and Corollary 12, the values of HPAs can be approximated and emptiness checked under isolation. These facts are also established in [2] through an alternative proof.

5 Ambiguity and Approximability

We now identify a large class of PFAs which are effectively leak monotonic. Any PFA \mathcal{A} over Σ can be viewed as a non-deterministic finite automaton $\text{nfa}(\mathcal{A})$ over Σ by ignoring the probability of transitioning from one state to another: $\text{nfa}(\mathcal{A})$ has the same set of states as

\mathcal{A} and there is a transition from state q to q' on a in $\text{nfa}(\mathcal{A})$ iff $\delta(q, a, q') > 0$. The degree of ambiguity of \mathcal{A} on word u is the degree of ambiguity of $\text{nfa}(\mathcal{A})$ on word u . We will be interested in PFAs that are polynomially ambiguous. We have the following observation.

► **Proposition 28.** *If a PFA \mathcal{A} has polynomial leak complexity with polynomial $h(\ell)$ then \mathcal{A} is polynomially ambiguous with polynomial $nh(n)$.*

Proof. Let \mathcal{A} have polynomial leak complexity with polynomial $h(\ell)$. Any accepting run of \mathcal{A} on a word of length n can have at most n leaks. Thus the number of accepting runs of \mathcal{A} on a word of length n is bounded above by $\sum_{\ell=1}^n h(\ell) \leq nh(n)$. ◀

From the proof of Theorem 26 and Proposition 28, we can conclude that every HPA is polynomially ambiguous. However, the converse is not true. We give an example of a linearly ambiguous PFA that is not a HPA.

► **Example 29.** Consider the PFA \mathcal{A}_5 on $\Sigma = \{\mathbf{0}, \mathbf{1}\}$ shown in Figure 2b on page 12. \mathcal{A}_5 is not hierarchical. This can be seen as follows. Since $S = \{q_0, q_1\}$ form a strongly connected component, they must be in the same level. However, then $\text{post}(q_0, \mathbf{0}) = \{q_0, q_1\}$ has two successors in the same level. Next, observe that on input $\mathbf{0}^k$ there are only two runs that remain in S . Thus, on input $\mathbf{0}^k$ there are $k - 1$ accepting runs. On the other hand, on input $\mathbf{0}^k \mathbf{1}$ there is exactly one run that remains in S , and this run ends in q_0 . Further, the number of accepting runs on $\mathbf{0}^k \mathbf{1}$ is k . Now a general input over Σ is either $u = \mathbf{0}^{k_1} \mathbf{1} \mathbf{0}^{k_2} \mathbf{1} \dots \mathbf{1}^{k_n}$ or $u \mathbf{1}$. Putting the above observations together, we have the number of accepting runs on u is $k_1 + k_2 + \dots + k_{n-1} + (k_n - 1)$ and on $u \mathbf{1}$ is $k_1 + k_2 + \dots + k_n$. Thus, \mathcal{A}_5 has linear ambiguity.

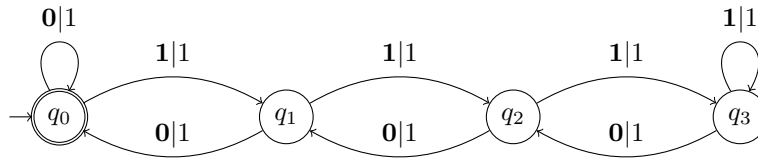
Thanks to Theorem 26 and Proposition 28, we can conclude that a k -HPA is polynomially ambiguous with polynomial $O(n^k)$. Since the value decision problem and emptiness problem of 2-HPAs are undecidable [8, 2], we get that the value decision problem and emptiness problem for quadratically ambiguous PFAs is also undecidable. The emptiness problem for quadratically ambiguous PFAs is shown to be undecidable in [16] as well. The problem of emptiness of linearly ambiguous PFAs was left open. A close examination of the 2-HPAs constructed in the undecidability proof of the emptiness problem for 2-HPAs established in [2], shows that the resulting automata have only linear ambiguity (instead of quadratic ambiguity). This observation proves that the emptiness problem of linearly ambiguous automata is undecidable. This result (with a different proof) was also independently observed in [13].

► **Theorem 30.** *The emptiness problem for linearly ambiguous PFAs is undecidable.*

In contrast, we will show that polynomially ambiguous automata are effectively regular-approximable, which will imply that their value can be approximated and emptiness under isolation be checked thanks to Theorem 11 and Corollary 12. We establish this by showing that every polynomially ambiguous PFA has polynomial leak complexity. This is a consequence of Lemma 32 below, which will allow us to bound leak complexity from bounds on degree of ambiguity. We need one further definition.

► **Definition 31.** For a PFA \mathcal{A} on Σ , word $u \in \Sigma^*$ and $\ell \in \mathbb{N}$, let $\text{accrns}(\mathcal{A}, u, \ell)$ be the set of accepting runs of \mathcal{A} on u with leaks $\leq \ell$. Formally, $\text{accrns}(\mathcal{A}, u, \ell)$ is the set $\{\rho \mid \rho \text{ is accepting and } \text{NbrLeaks}(\mathcal{A}, u, \rho) \leq \ell\}$.

We now show that for any word u and any integer ℓ , there is a *short* word v such that v has at least as many accepting runs with at most ℓ leaks as u does.



■ **Figure 3** Deterministic automaton \mathcal{A}_6 that is not eventually weakly ergodic.

► **Lemma 32.** *Let \mathcal{A} be a PFA with m states. For any word u and integer $\ell > 0$, there is a word v of length $\leq m + ((\ell + 1)m)^m$ such that $|\text{accruns}(\mathcal{A}, v, \ell)| \geq |\text{accruns}(\mathcal{A}, u, \ell)|$.*

Polynomial ambiguity implies polynomial leak complexity follows from Lemma 32.

► **Theorem 33.** *If PFA \mathcal{A} with m states is polynomially ambiguous with polynomial $p(n)$ then \mathcal{A} has polynomial leak complexity with polynomial $h(\ell) = p(m + ((\ell + 1)m)^m)$.*

Proof. Let \mathcal{A} be a PFA with m states. Fix an input word u and an integer ℓ . From Lemma 32, there is a word v such that $|v| \leq m + ((\ell + 1)m)^m$ and $|\text{accruns}(\mathcal{A}, u, \ell)| \leq |\text{accruns}(\mathcal{A}, v, \ell)|$. Now $\text{accruns}(\mathcal{A}, v, \ell)$ is a subset of the accepting runs of \mathcal{A} on input v . Since \mathcal{A} is polynomially ambiguous, we get $\text{accruns}(\mathcal{A}, v, \ell) \leq p(|v|) = p(m + ((\ell + 1)m)^m)$. ◀

Thanks to Theorem 33, we get that

► **Corollary 34.** *The class of polynomially ambiguous PFAs is effectively regular-approximable. The value of a polynomially ambiguous PFA can be approximated to any degree of precision and emptiness checked under isolation.*

6 Eventually Weakly Ergodic PFAs

Not all effectively regular-approximable PFAs are leak monotonic. We exhibit a class of PFAs from the literature that is effectively regular-approximable but not leak monotonic. Recall that a Markov Chain is ergodic if it is aperiodic and its underlying transition graph is strongly connected. Ergodicity in the context of PFAs have been studied in [29, 23, 19]. Intuitively, a PFA is weakly ergodic if any sequence of input symbols has only one terminal strongly connected component and this component is aperiodic. Weak ergodicity was generalized in [10] to define a new class of PFAs, called *eventually weakly ergodic* PFAs. Informally, a PFA \mathcal{A} is eventually weakly ergodic if its states can be partitioned into sets Q_T, Q_1, \dots, Q_r and there is an ℓ such that in the transition graph on any word of length ℓ , Q_1, \dots, Q_r are the only terminal strongly connected components, and in addition, they are aperiodic. (See Appendix F for the formal definition.) Every unary PFA turns out to be eventually weakly ergodic [10]. The problem of checking whether a PFA is eventually weakly ergodic is also decidable [10].

► **Example 35.** The PFA \mathcal{A}_3 in Figure 1c on page 7 is eventually weakly ergodic but not leak monotonic. This can be seen by taking $\ell = 1, Q_T = \emptyset, Q_1 = \{q_0, q_1\}$. On the other hand, the deterministic automaton \mathcal{A}_6 in Figure 3 is shown to be **not** eventually weakly ergodic in [10]. Thus, the class of leak monotonic automata and eventually weakly ergodic automata are not comparable.

Using the techniques of [10], we can show that the class of weakly ergodic PFAs is effectively regular-approximable. (See Appendix F for the proof.)

► **Theorem 36.** *The class of eventually weakly ergodic PFAs is effectively regular-approximable.*

Thus, we can approximate the value of eventually weakly ergodic PFAs and check emptiness under isolation for eventually weakly ergodic PFAs. Please note that the latter result is also given in [10].

7 Conclusions

In this paper, we addressed the problem of regular-approximability of PFAs. We showed that regular-approximability problem is undecidable. We also showed that if a PFA is regular-approximable then its value can be computed with arbitrary precision. We also showed that emptiness problem is decidable for regular-approximable PFAs when the given cut-point is isolated. We defined a class of PFAs, called leak monotonic PFAs and showed them to be regular-approximable. For PFAs belonging to this class, we gave an effective procedure for computing a deterministic automaton that approximates the language accepted by the given PFA with a given minimum probability threshold. We showed that PFAs with polynomial ambiguity as well as all HPAs are leak monotonic. We also introduced leak complexity and showed that PFAs with sub-exponential leak complexity are leak monotonic. We also solved an open problem showing that the emptiness problem is undecidable for PFAs with linear ambiguity. Finally, we showed that eventually weakly ergodic PFAs are also regular-approximable. As part of future work, it will be interesting to investigate algorithms to decide if a given PFA has sub-exponential leak complexity. The decidability of determining whether a given PFA is leak monotonic and checking emptiness under isolation for general PFAs are some other open problems.

References

- 1 C. Baier and M. Größer. Recognizing ω -regular languages with probabilistic automata. In *20th IEEE Symposium on Logic in Computer Science*, pages 137–146, 2005.
- 2 Y. Ben and A. P. Sistla. Model checking failure-prone open systems using probabilistic automata. In *13th International Symposium on Automated Technology for Verification and Analysis*, volume 9364 of *Lecture Notes in Computer Science*, pages 148–165. Springer, 2015.
- 3 A. Bertoni. The solution of problems relative to probabilistic automata in the frame of the formal languages theory. In *GI Jahrestagung*, pages 107–112, 1974.
- 4 A. Bertoni. Mathematical methods of the theory of stochastic automata. In *3rd Symposium of Mathematical Foundations of Computer Science*, volume 28 of *Lecture Notes in Computer Science*, pages 9–22. Springer, 1975.
- 5 R. Chadha, A. P. Sistla, and M. Viswanathan. Probabilistic Büchi automata with non-extremal acceptance thresholds. In *11th International Conference on Verification, Model checking and Abstract Interpretation*, pages 103–117, 2010.
- 6 R. Chadha, A. P. Sistla, and M. Viswanathan. Power of randomization in automata on infinite strings. *Logical Methods in Computer Science*, 7(3):1–22, 2011.
- 7 R. Chadha, A. P. Sistla, M. Viswanathan, and Y. Ben. Decidable and expressive classes of probabilistic automata. In *18th International Conference on Foundations of Software Science and Computation Structures*, volume 9034 of *Lecture Notes in Computer Science*, pages 200–214. Springer, 2015.
- 8 R. Chadha, A. Prasad Sistla, and M. Viswanathan. Emptiness under isolation and the value problem for hierarchical probabilistic automata. In *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017*, volume 10203 of *Lecture Notes in Computer Science*, pages 231–247, 2017.

- 9 R. Chadha, A.P. Sistla, and M. Viswanathan. Power of randomization in automata on infinite strings. In *20th International Conference on Concurrency Theory*, pages 229–243, 2009.
- 10 R. Chadha, A.P. Sistla, and M. Viswanathan. Probabilistic automata with isolated cut-points. In *38th International Symposium on Mathematical Foundation of Computer Science*, pages 254–265, 2013.
- 11 A. Condon and R. J. Lipton. On the complexity of space bounded interactive proofs (extended abstract). In *30th Annual Symposium on Foundations of Computer Science*, pages 462–467, 1989.
- 12 W. Czerwinski and S. Lasota. Regular separability of one counter automata. In *32nd IEEE Symposium on Logic in Computer Science*, pages 1–12, 2017.
- 13 L. Daviaud, M. Jurdzinski, R. Lazic, F. Mazowiecki, G. A. Pérez, and James Worrell. When is containment decidable for probabilistic automata? In *45th International Colloquium on Automata, Languages, and Programming*, 2018. To appear.
- 14 N. Fijalkow, H. Gimbert, E. Kelmendi, and Youssouf Oualhadj. Deciding the value 1 problem for probabilistic leaktight automata. *Logical Methods in Computer Science*, 11(2), 2015.
- 15 N. Fijalkow, H. Gimbert, and Y. Oualhadj. Deciding the value 1 problem for probabilistic leaktight automata. In *27th IEEE Symposium on Logic in Computer Science*, pages 295–304, 2012.
- 16 N. Fijalkow, C. Riveros, and J. Worrell. Probabilistic automata of bounded ambiguity. In *the International Conference on Concurrency Theory*, pages 19:1–19:14, 2017.
- 17 N. Fijalkow and M. Skrzypczak. Irregular behaviours for probabilistic automata. In *Reachability Problems*, pages 33–36, 2015.
- 18 H. Gimbert and Y. Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In *37th International Colloquium on Automata, Languages and Programming*, pages 527–538, 2010.
- 19 J. Hajnal and M. S. Bartlett. Weak ergodicity in non-homogeneous markov chains. *Mathematical proceedings of the Cambridge Philosophical Society*, 54(02):233–246, 1958.
- 20 O. H. Ibarra and B. Ravikumar. On sparseness, ambiguity and other decision problems for acceptors and transducers. In *3rd Annual Symposium on Theoretical Aspects of Computer Science*, pages 171–179, 1986.
- 21 G. Jacob. Un algorithme calculant le cardinal, fini ou infini, des demi-groupes de matrices. *Theoretical Computer Science*, 5(2):183–204, 1977.
- 22 A. Mandel and I. Simon. On finite semigroups of matrices. *Theoretical Computer Science*, 5(2):101–111, 1977.
- 23 A. Paz. Definite and quasidefinite sets of stochastic matrices. *Proceedings of the American Mathematical Society*, 16(4):634–641, 1965. URL: <http://www.jstor.org/stable/2033893>.
- 24 A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.
- 25 T. Place and M. Separation for dot-depth two. In *32nd IEEE Symposium on Logic in Computer Science*, pages 1–12, 2017.
- 26 M. O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.
- 27 C. Reutenauer. Propriétés arithmétiques et topologiques de séries rationnelles en variables non commutatives, 1997. Thèse troisième cycle, Université Paris VI.
- 28 A. Weber and H. Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991.
- 29 J. Wolfowitz. Products of indecomposable, aperiodic, stochastic matrices. *Proceedings of the American Mathematical Society*, 14(5):733–737, 1963.

A Proof of Theorem 9

Proof. Let `SomeApprox` be the set of all PFAs \mathcal{A} such that there is a non-trivial rational pair (x, y) such that $(\mathcal{A}, x, y) \in \text{Approx}$. We show that `SomeApprox` is Σ_2^0 -hard where Σ_2^0 is the second level in the arithmetical hierarchy. This automatically implies that `Approx` is not even recursively enumerable; for if it were recursively enumerable this would imply that `SomeApprox` is also recursively enumerable which will be a contradiction.

Let `ValueNot1` = $\{\mathcal{A} \mid \mathcal{A} \text{ is a PFA and } \text{value}(\mathcal{A}) < 1\}$. It has been shown in [10] that `ValueNot1` is Σ_2^0 -complete. We prove that `SomeApprox` is Σ_2^0 -hard by reducing `ValueNot1` to `SomeApprox`. Our reduction, given a PFA \mathcal{A} over Σ , constructs a PFA \mathcal{B} such that $\text{value}(\mathcal{A}) < 1$ iff $\mathcal{B} \in \text{SomeApprox}$. Let $\mathcal{A} = (Q, q_s, \delta, Q_f)$ be any PFA over some alphabet Σ . Now, we define \mathcal{B} as follows. If $\exists u \in \Sigma^*$ such that $P_{\mathcal{A}}(u) = 1$ then \mathcal{B} is simply the PFA \mathcal{A}_2 given in Figure 1b on page 7; observe that in this case, $\mathcal{A} \notin \text{ValueNot1}$, and $\mathcal{B} \notin \text{SomeApprox}$ as shown by Theorem 8. Note that the above condition can be checked effectively thanks to Proposition 4. If there is no such a string u , then we define \mathcal{B} to be a PFA over the alphabet $\Sigma' = \Sigma \cup \{\#\}$ defined as follows. $\mathcal{B} = (Q', q_s, \delta', Q_f)$ where $Q' = Q \cup \{q_r\}$ where $q_r \notin Q$ and δ' defined as follows: $\delta'(q, a, q') = \delta(q, a, q')$ for $q, q' \in Q$ and $a \in \Sigma$; $\delta'(q, \#, q_s) = 1$ for $q \in Q_f$; $\delta'(q, \#, q_r) = 1$ for $q \notin Q_f$; $\delta'(q_r, a, q_r) = 1$ for all $a \in \Sigma'$. Now, we make the following observations. For any $u \in \Sigma^*$, the acceptance probabilities of u by \mathcal{A} and \mathcal{B} are the same. Now consider any string v of the form $u_1\#u_2\#\dots\#u_k\#$ where each $u_i \in \Sigma^*$, for $1 \leq i \leq k$. It is easy to see that $P_{\mathcal{B}}(v) = \prod_{1 \leq i \leq k} P_{\mathcal{A}}(u_i)$. Also, $\text{value}(\mathcal{B}) = \text{value}(\mathcal{A})$.

Now, we show that $\mathcal{A} \in \text{ValueNot1}$ iff $\mathcal{B} \in \text{SomeApprox}$. Suppose $\mathcal{A} \in \text{ValueNot1}$. In this case, take any $x, y \in (0, 1)$ such that $\text{value}(\mathcal{A}) < x < x + y < 1$. Clearly such x, y exist, since $\text{value}(\mathcal{A}) < 1$. Since $\text{value}(\mathcal{B}) = \text{value}(\mathcal{A})$, we have $\text{value}(\mathcal{B}) < x < x + y < 1$. Clearly $L_{>x}(\mathcal{B}) = L_{\geq x+y}(\mathcal{B}) = \emptyset$. Since the empty set is a regular set, we see that \mathcal{B} is approximable with respect to (x, y) and hence $\mathcal{B} \in \text{SomeApprox}$. Now, assume $\mathcal{A} \notin \text{ValueNot1}$. This means $\text{value}(\mathcal{A}) = 1$. Now, we have two cases. In the first case, $\exists u \in \Sigma^*$ such that $P_{\mathcal{A}}(u) = 1$. In this case, by construction, \mathcal{B} is the automaton \mathcal{A}_2 which is not in `SomeApprox`. The second case is when there is no such string u . This means, for each $i > 0$, $\exists u_i \in \Sigma^*$ such that $P_{\mathcal{A}}(u_i) > (1 - \frac{1}{2^i})$. Since $P_{\mathcal{B}}(u_i) = P_{\mathcal{A}}(u_i)$, we have $P_{\mathcal{B}}(u_i) > (1 - \frac{1}{2^i})$. We show that $\mathcal{B} \notin \text{SomeApprox}$ by contradiction. Suppose $\mathcal{B} \in \text{SomeApprox}$. This means $\exists x, y$ and a regular language over $L \subseteq (\Sigma')^*$ such that $0 < x < x + y < 1$ and $L_{\geq x+y}(\mathcal{B}) \subseteq L \subseteq L_{>x}(\mathcal{B})$. Since L is a regular language, there exists an integer $N > 0$ satisfying Lemma 2. Now, let $z_1 = \max\{P_{\mathcal{A}}(u') \mid u' \in \Sigma^*, |u'| \leq N\}$. Fix an integer $k > 0$ such that $(z_1)^k \leq x$. Now, let $v \in \Sigma^*$ be any string such that $v = u_i$ for some $i > 0$ such that $(P_{\mathcal{A}}(v))^k \geq x + y$. Clearly such a string v exists. Now consider the string $w = (v\#)^k$ in $(\Sigma')^*$. Now, we have $P_{\mathcal{B}}(w) = (P_{\mathcal{A}}(v))^k \geq x + y$. Hence $w \in L$. Now applying Lemma 2, we see that there exists a string $w' = w_1\#w_2\#\dots\#w_k\#$ such that $w_i \in \Sigma^*, |w_j| \leq N$, for $1 \leq j \leq k$ and $w' \in L$. Clearly, $P_{\mathcal{A}}(w_i) \leq z_1$, for each i , $1 \leq i \leq k$. Now, $P_{\mathcal{B}}(w) = \prod_{1 \leq i \leq k} P_{\mathcal{A}}(w_i) \leq (z_1)^k$. Since $(z_1)^k \leq x$, we see that $P_{\mathcal{B}}(w) \leq x$ which contradicts our assumption that $L \subseteq L_{>x}(\mathcal{B})$. \blacktriangleleft

B Proof of Theorem 18

Proof. Let `SomeLeakMon` be the set of all PFAs \mathcal{A} such that there is an ϵ such that $(\mathcal{A}, \epsilon) \in \text{LeakMon}$. We show that `SomeLeakMon` is Σ_2^0 -hard where Σ_2^0 is the second level in the arithmetical hierarchy, which implies that `LeakMon` is not even recursively enumerable. As in the proof of Theorem 9, `ValueNot1` = $\{\mathcal{A} \mid \mathcal{A} \text{ is a PFA and } \text{value}(\mathcal{A}) < 1\}$ which is a Σ_2^0 -hard problem. We can conclude the theorem by reducing `ValueNot1` to `SomeLeakMon`.

Our reduction, given a PFA $\mathcal{A} = (Q, q_s, \delta, Q_f)$ over Σ , constructs a PFA \mathcal{B} such that $\text{value}(\mathcal{A}) < 1$ iff $\mathcal{B} \in \text{SomeLeakMon}$. Let $\mathcal{A} = (Q, q_s, \delta, Q_f)$ be any PFA over some alphabet Σ . Now, we define \mathcal{B} as follows. If $\exists u \in \Sigma^*$ such that $P_{\mathcal{A}}(u) = 1$ then \mathcal{B} is simply the PFA \mathcal{A}_3 given in Figure 1c on page 7; observe that in this case, $\mathcal{A} \notin \text{ValueNot1}$, and $\mathcal{B} \notin \text{SomeLeakMon}$.

If there is no such a string u , then we define \mathcal{B} to be a PFA over the alphabet Σ as follows. $\mathcal{B} = (Q \times \{1, 2\}, (q_s, 1), \delta', Q_f \times \{1, 2\})$ where $\delta'((q, i), a, (q', j)) = \frac{1}{2}\delta(q, a, q')$ for $q, q' \in Q, a \in \Sigma$ and $i, j \in \{1, 2\}$.

Now, we make the following observations. For any $u \in \Sigma^*$, the acceptance probabilities of u by \mathcal{A} and \mathcal{B} are the same. Thus, $\text{value}(\mathcal{B}) = \text{value}(\mathcal{A})$. Furthermore, every accepting run of \mathcal{B} on u has $|u|$ leaks. Using these observations, we shall show that $\mathcal{A} \in \text{ValueNot1}$ iff $\mathcal{B} \in \text{SomeLeakMon}$.

Suppose $\mathcal{A} \in \text{ValueNot1}$. Then there must exist ϵ_0 such that $\text{value}(\mathcal{B}) = \text{value}(\mathcal{A}) < \epsilon_0 < 1$. As no word is accepted by \mathcal{B} with probability $\geq \epsilon_0$, \mathcal{B} is ϵ_0 -leak monotonic with horizon $N_{\epsilon_0} = 0$.

Suppose $\mathcal{A} \notin \text{ValueNot1}$. Then $\text{value}(\mathcal{A}) = 1$. As there is no word accepted by \mathcal{A} with probability 1 and Σ is finite, we get that there must be an infinite sequence of non-empty words u_1, u_2, \dots such that for each i , $|u_i| < |u_{i+1}|$ and $P_{\mathcal{A}}(u_i) > 1 - \frac{1}{i}$. Suppose, for contradiction, $\mathcal{B} \in \text{SomeLeakMon}$. This means that there must exist $\epsilon_0 \in (0, 1)$ and N_{ϵ_0} such that \mathcal{B} is ϵ_0 -leak monotonic with horizon N_{ϵ_0} . Please note that as $\epsilon_0 < 1$, there must exist a j_0 such that $1 - \frac{1}{i} > \epsilon_0$ for all $i \geq j_0$. Fix $k = \max(N_{\epsilon_0}, j_0)$. Consider the word u_k . We have that $|u_k| \geq k \geq N_{\epsilon_0}$ and every run of \mathcal{B} on u_k has exactly $|k|$ leaks. As N_{ϵ_0} is a horizon of ϵ_0 -leak monotonicity we must have $P_{\mathcal{A}}(u_k) < \epsilon_0$. This contradicts the fact that $P_{\mathcal{A}}(u_k) = 1 - \frac{1}{k} > \epsilon_0$. \blacktriangleleft

C Proof of Theorem 21

Proof. For $i > 0, q \in Q_i$, let $\mathcal{A}_{i,q}$ be the restriction of \mathcal{A} to the set Q_i of states with starting state q . For any $\epsilon \in (0, 1)$, let $N_\epsilon > 0$ be a constant such that, for each $i > 0, q \in Q_i$ and each $u \in \Sigma^*$, the measure of the set of accepting runs of $\mathcal{A}_{i,q}$ on u , having at least N_ϵ leaks, is less than ϵ . Such a constant N_ϵ exists since each $\mathcal{A}_{i,q}$ is leak monotonic. Now let p be the minimum of the probabilities of reaching a state in $Q \setminus Q_0$, from any state in Q_0 , on any input string of length exactly m , where m is the constant specified in the theorem. Clearly $p > 0$. Now, fix an $\epsilon \in (0, 1)$. We specify a constant M_ϵ such that on every $u \in \Sigma^*$, the measure of the set of accepting runs of \mathcal{A} on u , having at least M_ϵ leaks, is less than ϵ . Let n' be the smallest integer such that $(1-p)^{n'} < \frac{\epsilon}{2}$ and let $L_{\frac{\epsilon}{2}} = m \cdot n'$. Observe that for any $u \in \Sigma^*$ of length at least $L_{\frac{\epsilon}{2}}$, $\delta_u(q_s, Q_0) < \frac{\epsilon}{2}$, i.e., the probability that \mathcal{A} is in some state in Q_0 after u is $< \frac{\epsilon}{2}$.

Now, let $M_\epsilon = L_{\frac{\epsilon}{2}} + N_{\frac{\epsilon}{2}}$. We show that M_ϵ satisfies the desired property. Now, consider any input string $u \in \Sigma^*$. If $|u| < M_\epsilon$ then the measure of the set of all runs of \mathcal{A} on u having at least M_ϵ leaks is zero. So, assume that $|u| \geq M_\epsilon$. Let u_1 be the prefix of u of length $L_{\frac{\epsilon}{2}}$ and $u_2 \in \Sigma^*$ be the suffix of u following u_1 , i.e., $u = u_1 u_2$. For any $i > 0, q \in Q_i$, let p_q be the probability measure of the set of all runs of $\mathcal{A}_{i,q}$, on input u_2 , having at least $N_{\frac{\epsilon}{2}}$ leaks. Observe that $p_q < \frac{\epsilon}{2}$. Now, we see that the probability measure of the set of all accepting runs of \mathcal{A} on u , having at least M_ϵ leaks, is bounded by $\frac{\epsilon}{2} + \sum_{q \in Q \setminus Q_0} \delta_{u_1}(q_s, q) \cdot p_q$. In the above expression, the first term in the sum bounds the probability of all such runs that remain entirely within Q_0 and the second term bounds the probability of all such runs that end in a state outside Q_0 . Since $p_q < \frac{\epsilon}{2}$ for $q \in Q \setminus Q_0$ and since $\sum_{q \in Q \setminus Q_0} \delta_{u_1}(q_s, q) \leq 1$, we see that the probability measure of the set of all accepting runs of \mathcal{A} on u , having at least M_ϵ leaks, is less than ϵ . \blacktriangleleft

D Proof of Theorem 26

Proof. The theorem is an easy consequence of Theorem 25, Theorem 16 and the following claim:

► **Claim.** *Every k -HPA \mathcal{A} with n -states and $k > 0$, has leak complexity at most $n^k \ell^{k-1}$.*

Proof. We prove this claim by induction on k . The base case is when $k = 1$. In this case, any accepting run that has ℓ leaks, either completely stays at level 0 or goes from a level 0 state to a higher level state making a non-leaky transition, or it goes to a level 1 state exactly after the ℓ^{th} leak (this is so because there can not be any leaks from level 1 states). Clearly, there can be at most m such runs that end in a level 1 accepting state, where m is the number of level 1 states. Thus, the total number of such runs can be at most $1 + m \leq n$, which is a constant independent of ℓ .

Now, assume that the claim is true for any $k > 0$. We show that that claim holds for $(k + 1)$ -HPA as well. Consider a $(k + 1)$ -HPA \mathcal{A} on an input alphabet Σ . Let m be the total number of states at levels 1 and higher. Consider an input $u \in \Sigma^*$. Let X be the set of accepting runs of \mathcal{A} on an input u , having $\ell > 0$ leaks. Let ℓ' be the maximum of the number of leaks from a level 0 state in any of the runs in X . Observe that $\ell' \leq \ell$. The set X can be partitioned into $\ell' + 1$ disjoint sets $X_b, X_1, \dots, X_{\ell'}$, where X_b is a singleton consisting of the run that stays at level 0 or transitions from a level 0 state to a higher level state using a non-leaky transition, and X_i are the set of runs that made a transition from a level 0 state to a higher level state on the i^{th} leak, for $1 \leq i \leq \ell'$. For each i , $1 \leq i \leq \ell'$, let u_i be the prefix of the input after which the i^{th} leak occurred, and v_i be the suffix of u following u_i . All runs in X_i have the same prefix, say ρ_i , until the level 0 state from which the i^{th} leak occurred and they transition to one or more of the m higher level states after this leak. Thus, we can partition X_i into $m_i \leq m$ disjoint sets $X_{i,1}, \dots, X_{i,m_i}$ such that all runs in $X_{i,j}$ transition to the same higher level state, say $q_{i,j}$, after the i^{th} leak, which is immediately after ρ_i . Now $X_{i,j}$ is simply the set of runs having prefix ρ_i followed by the set $X'_{i,j}$ of all accepting runs of \mathcal{A} starting from the state $q_{i,j}$ on the input v_i and having $\ell - i$ leaks. Since $q_{i,j}$ is a higher level state, the restriction of \mathcal{A} having $q_{i,j}$ as a start state is a k' -HPA for some $k' \leq k$. Now by the induction hypothesis, we see that the number of runs in $X'_{i,j}$ and hence those in $X_{i,j}$ is bounded by $n^{k'} \cdot (\ell - i)^{k'-1}$. From this we see that the number of runs in X_i is bounded by $m \cdot n^{k'} \cdot (\ell - i)^{k'-1}$. From this we see that $|X| \leq 1 + \sum_{1 \leq i \leq \ell'} m \cdot n^{k'} \cdot (\ell - i)^{k'-1}$. Since $\ell' \leq \ell$, we get $|X| \leq 1 + m \cdot n^k \cdot \ell^k \leq n^{k+1} \ell^k$. ◀

The Theorem follows. ◀

E Proof of Lemma 32

Proof. Fix u and ℓ . Let v be the word of the shortest length such that $|\text{accruns}(\mathcal{A}, v, \ell)| \geq |\text{accruns}(\mathcal{A}, u, \ell)|$. We will show that length of v is $\leq m + ((\ell + 1)m)^m$. Observe that the set of finite non-empty prefixes of $\text{accruns}(\mathcal{A}, v, \ell)$ can be arranged as a tree \mathbb{T} as follows. The initial state q_s is the root of the tree. If ρq is a prefix of some run in $\text{accruns}(\mathcal{A}, v, \ell)$ then ρq is a child of ρ . Attach to each node ρ of \mathbb{T} , two labels: a state label $st(\rho)$ which is the last state of ρ and a leak label $lk(\rho)$ which is the number of leaks in ρ . For each depth i , let c_i be the set of nodes at depth i . We say that a leak occurs at node ρ if there is a state q' such that $\rho q'$ is in the tree \mathbb{T} and $lk(\rho q') = lk(\rho) + 1$. Observe that if there is a leak at a node ρ at depth i with state label ρ then there is a leak at every node ρ' at depth i with state label q . We say that a leak occurs at depth i if a leak occurs at some node $\rho \in c_i$. We show that leaks in \mathbb{T} cannot be too far apart.

► **Claim.** *Let $i, j \leq |v|$ be such that $j - i > m^m$ then there is a $i \leq k \leq j$ such that a leak occurs at depth k .*

Proof. We proceed by contradiction. Assume that there are i and j with $j - i > m^m$ with no leak occurring at any depth k between i and j . Consider any node $\rho \in c_i$. By our assumption, for each $i \leq k \leq j$, there is a *unique* descendant of ρ_k of ρ . The leak label of ρ_k is exactly the leak label of ρ . Furthermore, for any two nodes ρ and ρ' of c_i with the same state labels, the state labels of ρ_k and ρ'_k are exactly the same. From this, it is easy to see that there are k_1 and k_2 with $i \leq k_1 < k_2 \leq j$ such that for each node ρ of c_i , the state and leak labels of ρ_{k_1} and ρ_{k_2} are exactly the same. Let w be the string obtained from u by deleting the subword $u[k_1 + 1 : k_2]$ from v . It is easy to see that $\text{accruns}(\mathcal{A}, w, \ell) \geq \text{accruns}(\mathcal{A}, v, \ell)$ contradicting the minimality of v . ◀

A similar argument shows that there must be an $i \leq m$ such that there is a leak at depth i . Thus, we can conclude the Lemma if we can show that there are at most $(\ell + 1)^m$ depths at which a leak can occur; this is so due to the fact that the first depth at which a leak occurs is in the first m input symbols, and there are at most $(\ell + 1)^m$ depths at which leaks can occur and there are at most m^m input symbols between two successive such depths.

► **Claim.** *There are at most $(\ell + 1)^m$ depths at which a leak can occur.*

Proof. For each depth i , we define a function $\text{sml}_i : Q \rightarrow \{\perp, 1, 2, \dots, \ell\}$ as follows

$$\text{sml}_i(q) = \begin{cases} \perp & \text{if } \{\rho \mid \rho \in c_i, st(\rho) = q, lk(\rho) > 0\} = \emptyset \\ n & \text{if } n = \min\{j > 0 \mid \exists \rho \in c_i, st(\rho) = q \text{ and } lk(\rho) = j\} \end{cases}.$$

Since there are only $(\ell + 1)^m$ possible functions sml_i , it suffices to show that for any two depths $i < j$ such that there is a leak at some depth $i \leq k < j$, we have that $\text{sml}_i \neq \text{sml}_j$. Observe that if there is no leak up-to depth i , then the latter is trivially true. So, we consider the case when there has been at least one leak before depth i .

To each depth j such that there is a leak before depth j , we associate an integer $1 \leq \text{level}_j \leq \ell + 1$. If there is no leak at depth j , $\text{level}_j = \ell + 1$. Otherwise level_j is the smallest integer $1 \leq r \leq \ell + 1$ such there is a leak at node ρ of c_j with leak label r .

Fix j such that there is a leak before depth j . We make the following two observations:

- (a) For each $r < \text{level}_j$, we have that $|\text{sml}_j^{-1}(\{1, 2, \dots, r\})| \geq |\text{sml}_{j+1}^{-1}(\{1, 2, \dots, r\})|$. This follows from the fact that there is a surjection g from the set $\text{sml}_j^{-1}(\{1, 2, \dots, r\})$ to the set $\text{sml}_{j+1}^{-1}(\{1, 2, \dots, r\})$ defined as follows. Let $q \in \text{sml}_j^{-1}(\{1, 2, \dots, r\})$. The definition of sml implies that there is a unique state q' such that $\delta(q, v[j], q') = 1$. Let $g(q) = q'$. The function g is easily seen to be a surjection.
- (b) If there is a leak at depth j then $|\text{sml}_j^{-1}(\{1, 2, \dots, \text{level}_j\})| > |\text{sml}_{j+1}^{-1}(\{1, 2, \dots, \text{level}_j\})|$. This can be concluded as follows. Let $A \subseteq \text{sml}_j^{-1}(\{1, 2, \dots, \text{level}_j\})$ be the set of states q such that there is no leak at any node $\rho \in c_j$ with state label q . Clearly A is a proper subset of $\text{sml}_j^{-1}(\{1, 2, \dots, \text{level}_j\})$. We can again define a surjection g from A onto $|\text{sml}_{j+1}^{-1}(\{1, 2, \dots, \text{level}_j\})|$ as in (a) above.

Now, let $i < j$ be such that such that there is a leak at some depth $i \leq k < j$. Let $r = \min(\text{level}_t \mid i \leq t < j)$. Observations (a) and (b) above imply that $|\text{sml}_i^{-1}(\{1, 2, \dots, r\})| > |\text{sml}_j^{-1}(\{1, 2, \dots, r\})|$. Thus, $\text{sml}_i \neq \text{sml}_j$. ◀

This concludes the proof of the Lemma. ◀

F

 Eventually weakly ergodic PFAs are regular-approximable

We recall the formal definition of eventually weakly ergodic PFAs.

► **Definition 37.** A PFA $\mathcal{A} = (Q, \delta, q_s, Q_f)$ is said to be *eventually weakly ergodic* if there is a partition Q_T, Q_1, \dots, Q_r of Q and a natural number $\ell > 0$ such that the following conditions hold:

- For each word u of length ℓ , each $1 \leq i \leq r$ and state $q_i \in Q_i$, $\text{post}(q_i, u) \subseteq Q_i$.
- For each word u of length ℓ and each $1 \leq i \leq r$, there exists a state $q_i^u \in Q_i$ such that $q_i^u \in \text{post}(q_i, u)$ for each $q_i \in Q_i$.
- For each word u of length ℓ and each state $q \in Q_T$, $\text{post}(q, u) \cap (\cup_{1 \leq j \leq r} Q_j) \neq \emptyset$.

It is shown in [10] that the acceptance probability of each word u can be approximated by a short word v . In order to describe this result, we recall the following definition from [10]:

► **Definition 38.** Given an alphabet Σ and natural numbers $\ell, \ell' > 0$ such that ℓ' divides ℓ , let $c_{(\ell, \ell')} : \Sigma^* \rightarrow \Sigma^*$ be defined as follows.

$$c_{(\ell, \ell')}(u) = \begin{cases} u & \text{if } |u| < \ell' + 2\ell; \\ u_0 u_1 v_1 & \text{if } u = u_0 u_1 w v_1, |u_0| < \ell', |u_1| = \ell, w \in (\Sigma^{\ell'})^+ \text{ and } |v_1| = \ell \end{cases}$$

► **Remark.** Observe that $c_{(\ell, \ell')}(\cdot)$ is well defined. If $|u| \geq \ell' + 2\ell$ then there are unique u_0, u_1, w, v_1 such that $u = u_0 u_1 w v_1, |u_0| < \ell', |u_1| = \ell, w \in (\Sigma^{\ell'})^+, |v_1| = \ell$.

The following is shown in [10].

► **Lemma 39.** Given an eventually weakly ergodic PFA $\mathcal{A} = (Q, \delta, q_s, Q_f)$ and $y > 0$, there are $\ell > 0$ and $\ell' > 0$ s.t. ℓ' divides ℓ and

$$\forall u \in \Sigma^*. |P_{\mathcal{A}}(u) - P_{\mathcal{A}}(c_{(\ell, \ell')}(u))| < \frac{y}{2}.$$

Furthermore, if y is rational then ℓ, ℓ' can be computed from \mathcal{A} and y .

Given x, y , Lemma 39 can be used to show that an eventually weakly ergodic PFA \mathcal{A} is regular-approximable with respect to (x, y) . The proof proceeds as follows. First, we compute ℓ', ℓ as given in Lemma 39. Next, we construct a regular language L that approximates $L_{>x}(\mathcal{A})$ as follows. L is the union of two regular languages L_{short} and L_{long} . $L_{short} = \{u \in \Sigma^* \mid |u| < \ell' + 2\ell, P_{\mathcal{A}}(u) > x\}$. It is easy to see that L_{short} is finite and hence regular.

We construct L_{long} by constructing a NFA \mathcal{B} that recognizes L_{long} . The set of states of \mathcal{B} is a union of four sets Q_0, Q_1, Q_2, Q_3 defined as follows:

- $Q_0 = \{u_0 \in \Sigma^* \mid |u_0| < \ell'\}$.
- $Q_1 = \{(u_0, u_1) \in \Sigma^* \mid |u_0| < \ell', |u_1| \leq \ell\}$.
- $Q_2 = \emptyset$ if $\ell' = 1$ else $Q_2 = \{(u_0, u_1, i) \in \Sigma^* \mid |u_0| < \ell', |u_1| = \ell, 1 \leq i \leq \ell' - 1\}$.
- $Q_3 = \{(u_0, u_1, v_1) \in \Sigma^* \mid |u_0| < \ell', |u_1| = \ell, |v_1| \leq \ell\}$.

The transition relation of \mathcal{B} is defined as follows. For each input symbol a :

- For each $u_0 \in Q_0$, there is a transition from u_0 to $(u_0, a) \in Q_1$ on a . Furthermore, there is also a transition from u_0 to $u_0 a$ if $|u_0 a| < \ell'$.
- For each $(u_0, u_1) \in Q_1$ such that $|u_1| < \ell$, there is a transition from (u_0, u_1) to $(u_0, u_1 a)$ on a .
- For each $(u_0, u_1) \in Q_1$ such that $|u_1| = \ell$, there are two transitions on input a :
 1. There is a transition to $(u_0, u_1, a) \in Q_3$ on a .
 2. If $\ell' = 1$ then there is a transition from (u_0, u_1) to itself on a . If $\ell' > 1$ then there is a transition from (u_0, u_1) to $(u_0, u_1, 1) \in Q_2$ on a .

- For each $(u_0, u_1, i) \in Q_2$ such that $i < \ell' - 1$, there is a transition to $(u_0, u_1, i + 1) \in Q_2$ on input a .
- For each $(u_0, u_1, \ell' - 1) \in Q_2$, there is a transition to $(u_0, u_1) \in Q_1$ on input a .
- For each $(u_0, u_1, v_1) \in Q_3$ such that $|v_1| < \ell$, there is a transition to $(u_0, u_1, v_1 a) \in Q_3$ on input a .
- There are no other transitions of \mathcal{B} .

The initial state of \mathcal{B} is the empty string λ . The set of final states of \mathcal{B} is the set:

$$\{(u_0, u_1, v_1) \in Q_3 \mid |v_1| = \ell, P_{\mathcal{A}}(u_0 u_1 v_1) \geq x + \frac{y}{2}\}.$$

Thanks to Lemma 39, it is easy to see that $L_{\geq x+y}(\mathcal{A}) \subseteq L \subseteq L_{>x}(\mathcal{A})$.


An Application of Parallel Cut Elimination in Unit-Free Multiplicative Linear Logic to the Taylor Expansion of Proof Nets

Jules Chouquet

IRIF UMR 8243, Université Paris Diderot, Sorbonne Paris Cité, CNRS, France
Jules.Chouquet@irif.fr

Lionel Vaux Auclair

Aix-Marseille Univ, CNRS, Centrale Marseille, I2M, Marseille, France
lionel.vaux@univ-amu.fr

 <https://orcid.org/0000-0001-9466-418X>

Abstract

We examine some combinatorial properties of parallel cut elimination in multiplicative linear logic (MLL) proof nets. We show that, provided we impose some constraint on switching paths, we can bound the size of all the nets satisfying this constraint and reducing to a fixed resultant net. This result gives a sufficient condition for an infinite weighted sum of nets to reduce into another sum of nets, while keeping coefficients finite. We moreover show that our constraints are stable under reduction.

Our approach is motivated by the quantitative semantics of linear logic: many models have been proposed, whose structure reflect the Taylor expansion of multiplicative exponential linear logic (MELL) proof nets into infinite sums of differential nets. In order to simulate one cut elimination step in MELL, it is necessary to reduce an arbitrary number of cuts in the differential nets of its Taylor expansion. It turns out our results apply to differential nets, because their cut elimination is essentially multiplicative. We moreover show that the set of differential nets that occur in the Taylor expansion of an MELL net automatically satisfy our constraints.

In the present work, we stick to the unit-free and weakening-free fragment of linear logic, which is rich enough to showcase our techniques, while allowing for a very simple kind of constraint: a bound on the number of cuts that are crossed by any switching path.

2012 ACM Subject Classification Theory of computation → Linear logic

Keywords and phrases linear logic, proof nets, cut elimination, differential linear logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.15

Funding This work was supported by the French-Italian *Groupement de Recherche International* on Linear Logic.

1 Introduction

1.1 Context: quantitative semantics and Taylor expansion

Linear logic takes its roots in the denotational semantics of λ -calculus: it is often presented, by Girard himself [15], as the result of a careful investigation of the model of coherence spaces. Since its early days, linear logic has thus generated a rich ecosystem of denotational models, among which we distinguish the family of *quantitative semantics*. Indeed, the first ideas behind linear logic were exposed even before coherence spaces, in the model of normal



© Jules Chouquet and Lionel Vaux Auclair;
licensed under Creative Commons License CC-BY
27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 15; pp. 15:1–15:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

functors [16], in which Girard proposed to consider analyticity, instead of mere continuity, as the key property of the interpretation of λ -terms: in this setting, terms denote power series, representing analytic maps between modules.

This quantitative interpretation reflects precise operational properties of programs: the degree of a monomial in a power series is closely related to the number of times a function uses its argument. Following this framework, various models were considered – among which we shall include the multiset relational model as a degenerate, boolean-valued instance. These models allowed to represent and characterize quantitative properties such as the execution time [5], including best and worst case analysis for non-deterministic programs [18], or the probability of reaching a value [2]. It is notable that this whole approach gained momentum in the early 2000's, after the introduction by Ehrhard of models [7, 8] in which the notion of analytic maps interpreting λ -terms took its usual sense, while Girard's original model involved set-valued formal power series. Indeed, the keystone in the success of this line of work is an analogue of the Taylor expansion formula, that can be established both for λ -terms and for linear logic proofs.

Mimicking this denotational structure, Ehrhard and Regnier introduced the differential λ -calculus [12] and differential linear logic [13], which allow to formulate a syntactic version of Taylor expansion: to a λ -term (resp. to a linear logic proof), we associate an infinite linear combination of approximants [14, 11]. In particular, the dynamics (i.e. β -reduction or cut elimination) of those systems is dictated by the identities of quantitative semantics. In turn, Taylor expansion has become a useful device to design and study new models of linear logic, in which morphisms admit a matrix representation: the Taylor expansion formula allows to describe the interpretation of promotion – the operation by which a linear resource becomes freely duplicable – in an explicit, systematic manner. It is in fact possible to show that any model of differential linear logic without promotion gives rise to a model of full linear logic in this way [4]: in some sense, one can simulate cut elimination through Taylor expansion.

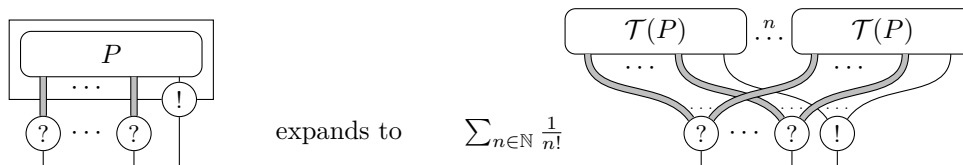
1.2 Motivation: reduction in Taylor expansion

There is a difficulty, however: Taylor expansion generates infinite sums and, *a priori*, there is no guarantee that the coefficients in these sums will remain finite under reduction. In previous works [4, 18], it was thus required for coefficients to be taken in a complete semiring: all sums should converge. In order to illustrate this requirement, let us first consider the case of λ -calculus.

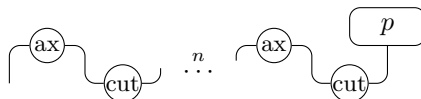
The linear fragment of differential λ -calculus, called *resource λ -calculus*, is the target of the syntactical Taylor expansion of λ -terms. In this calculus, the application of a term to another is replaced with a multilinear variant: $\langle s \rangle [t_1, \dots, t_n]$ denotes the n -linear symmetric application of resource term s to the multiset of resource terms $[t_1, \dots, t_n]$. Then, if x_1, \dots, x_k denote the occurrences of x in s , the redex $\langle \lambda x.s \rangle [t_1, \dots, t_n]$ reduces to the sum $\sum_{f: \{1, \dots, k\} \xrightarrow{\sim} \{1, \dots, n\}} s[t_{f(1)}/x_1, \dots, t_{f(k)}/x_k]$: here f ranges over all bijections $\{1, \dots, k\} \xrightarrow{\sim} \{1, \dots, n\}$ so this sum is zero if $n \neq k$. As sums are generated by reduction, it should be noted that all the syntactic constructs are linear, both in the sense that they commute to sums, and in the sense that, in the elimination of a redex, no subterm is copied nor erased. The key case of Taylor expansion is that of application:

$$\mathcal{T}(MN) = \sum_{n \in \mathbb{N}} \frac{1}{n!} \langle \mathcal{T}(M) \rangle \mathcal{T}(N)^n \quad (1)$$

where $\mathcal{T}(N)^n$ is the multiset made of n copies of $\mathcal{T}(N)$ – by n -linearity, $\mathcal{T}(N)^n$ is itself an infinite linear combination of multisets of resource terms appearing in $\mathcal{T}(N)$. Admitting that



■ **Figure 1** Taylor expansion of a promotion box (thick wires denote an arbitrary number of wires).



■ **Figure 2** Example of a family of nets, all reducing to a single net.

$\langle M \rangle [N_1, \dots, N_n]$ represents the n -th derivative of M , computed at 0, and n -linearly applied to N_1, \dots, N_n , one immediately recognizes the usual Taylor expansion formula.

From (1), it is immediately clear that, to simulate one reduction step occurring in N , it is necessary to reduce in parallel in an unbounded number of subterms of each component of the expansion. Unrestricted parallel reduction, however, is ill defined in this setting. Consider the sum $\sum_{n \in \mathbb{N}} \langle \lambda x x \rangle [\dots \langle \lambda x x \rangle [y] \dots]$ where each summand consists of n successive linear applications of the identity to the variable y : then by simultaneous reduction of all redexes in each component, each summand yields y , so the result should be $\sum_{n \in \mathbb{N}} y$ which is not defined unless the semiring of coefficients is complete in some sense.

Those considerations apply to linear logic as well as to λ -calculus. We will use proof nets [15] as the syntax for proofs of multiplicative exponential linear logic (MELL). The target of Taylor expansion is then in promotion-free differential nets [13], which we call *resource nets* in the following, by analogy with resource λ -calculus: these form the multilinear fragment of differential linear logic.

In linear logic, Taylor expansion consists in replacing duplicable subnets, embodied by promotion boxes, with explicit copies, as in Fig. 1: if we take n copies of the box, the main port of the box is replaced with an n -ary ! link, while the ? links at the border of the box collect all copies of the corresponding auxiliary ports. Again, to follow a single cut elimination step in P , it is necessary to reduce an arbitrary number of copies. And unrestricted parallel cut elimination in an infinite sum of resource nets is broken, as one can easily construct an infinite family of nets, all reducing to the same resource net p in a single step of parallel cut elimination: see Fig. 2.

1.3 Our approach: taming the combinatorial explosion of antireduction

The problem of convergence of series of linear approximants under reduction was first tackled by Ehrhard and Regnier, for the normalization of Taylor expansion of ordinary λ -terms [14]. Their argument relies on a uniformity property, specific to the pure λ -calculus: the support of the Taylor expansion of a λ -term forms a clique in some fixed coherence space of resource terms. This method cannot be adapted to proof nets: there is no coherence relation on differential nets such that all supports of Taylor expansions are cliques [22, section V.4.1].

An alternative method to ensure convergence without any uniformity hypothesis was first developed by Ehrhard for typed terms in a λ -calculus extended with linear combinations of terms [9]: there, the presence of sums also forbade the existence of a suitable coherence relation. This method can be generalized to strongly normalizable [20], or even weakly normalizable [23] terms. One striking feature of this approach is that it concentrates on

the support (i.e. the set of terms having non-zero coefficients) of the Taylor expansion. In each case, one shows that, given a normal resource term t and a λ -term M , there are finitely many terms s , such that:

- the coefficient of s in $\mathcal{T}(M)$ is non zero; and
- the coefficient of t in the normal form of s is non zero.

This allows to normalize the Taylor expansion: simply normalize in each component, then compute the sum, which is component-wise finite.

The second author then remarked that the same could be done for β -reduction [23], even without any uniformity, typing or normalizability requirement. Indeed, writing $s \rightrightarrows t$ if s and t are resource terms such that t appears in the support of a parallel reduct of s , the size of s is bounded by a function of the size of t and the height of s . So, given that if s appears in $\mathcal{T}(M)$ then its height is bounded by that of M , it follows that, for a fixed resource term t there are finitely many terms s in the support of $\mathcal{T}(M)$ such that $s \rightrightarrows t$: in short, parallel reduction is always well-defined on the Taylor expansion of a λ -term.

Our purpose in the present paper is to develop a similar technique for MELL proof nets: we show that one can bound the size of a resource net p by a function of the size of any of its parallel reducts, and of an additional quantity on p , yet to be defined. The main challenge is indeed to circumvent the lack of inductive structure in proof nets: in such a graphical syntax, there is no structural notion of height.

We claim that a side condition on switching paths, i.e. paths in the sense of Danos–Regnier’s correctness criterion [3], is an appropriate replacement. Backing this claim, there are first some intuitions:

- the culprits for the unbounded loss of size in reduction are the chains of consecutive cuts, as in Fig. 2;
- we want the validity of our side condition to be stable under reduction so, rather than chains of cuts, we should consider cuts in switching paths;
- indeed, if p reduces to q via cut elimination, then the switching paths of q are somehow related with those of p ;
- and the switching paths of a resource net in $\mathcal{T}(P)$ are somehow related with those of P .

In the following, we establish this claim up to some technical restrictions, which will allow us to simplify the exposition:

- we use generalized n -ary exponential links rather than separate (co)dereliction and (co)contraction, as this allows to reduce the dynamics of resource nets to that of multiplicative linear logic (MLL) proof nets;¹
- we limit our study to a *strict* fragment of linear logic, i.e. we do not consider multiplicative units, nor the 0-ary exponential links – weakening and coweakening – as dealing with them would require us to introduce much more machinery.

1.4 Outline

In Section 2, we first introduce proof nets formally, in the term-based syntax of Ehrhard [10]. We define the parallel cut elimination relation \rightrightarrows in this setting, that we decompose into multiplicative reduction \rightrightarrows_m and axiom-cut reduction \rightrightarrows_{ax} . We also present the notion of switching path for this syntax, and introduce the quantity that will be our main object of study in the following: the maximum number $\mathbf{cc}(p)$ of cuts that are crossed by any switching path in the net p . Let us mention that typing plays absolutely no role in our approach, so we do not even consider formulas of linear logic: we will rely only on the acyclicity of nets.

¹ In other words, we adhere to a version of linear logic proof nets and resource nets which is sometimes called *nouvelle syntaxe*, although it dates back to Regnier’s PhD thesis [21]. See also the discussion in our conclusion (Section 6).

Section 3 is dedicated to the proof that we can bound $\mathbf{cc}(q)$ by a function of $\mathbf{cc}(p)$, whenever $p \Rightarrow q$: the main case is the multiplicative reduction, as this may create new switching paths in q that we must relate with those in p . In this task, we concentrate on the notion of *slipknot*: a pair of residuals of a cut of p occurring in a path of q . Slipknots are essential in understanding how switching paths are structured after cut elimination.

We show in Section 4 that, if $p \Rightarrow q$ then the size of p is bounded by a function of $\mathbf{cc}(p)$ and the size of q . Although, as explained in our introduction, this result is motivated by the study of quantitative semantics, it is essentially a theorem about MLL.

We establish the applicability of our approach to the Taylor expansion of MELL proof nets in Section 5: we show that if p is a resource net of $\mathcal{T}(P)$, then the length of switching paths in p is bounded by a function of the size of P – hence so is $\mathbf{cc}(p)$.

Finally, we discuss further work in the concluding Section 6.

2 Definitions

We provide here the minimal definitions necessary for us to work with MLL proof nets. We use a term-based syntax, following Ehrhard [10].

As stated before, let us stress the fact that the choice of MLL is not decisive for the development of Sections 2 to 4. The reader can check that we rely on two ingredients only:

- the definition of switching paths;
- the fact that multiplicative reduction amounts to plug bijectively the premises of a \otimes link with those of \wp link.

The results of those sections are thus directly applicable to resource nets, thanks to our choice of generalized exponential links: this will be done in Section 6.

2.1 Structures

Our nets are finite families of trees and cuts; trees are inductively defined as MLL connectives connecting trees, where the leaves are elements of a countable set of variables V . The duality of two conclusions of an axiom is given by an involution $x \mapsto \bar{x}$ over this set.

Formally, the set T of *raw trees* (denoted by s, t , etc.) is generated as follows:

$$t ::= x \mid \otimes(t_1, \dots, t_n) \mid \wp(t_1, \dots, t_n)$$

where x ranges over a fixed countable set of variables V , endowed with a fixpoint-free involution $x \mapsto \bar{x}$.

We also define the subtrees of a given tree t , written $\mathbf{T}(t)$, in the natural way : if $t \in V$, then $\mathbf{T}(t) = \{t\}$. If $t = \alpha(t_1, \dots, t_n)$, then $\mathbf{T}(t) = \{t\} \cup \bigcup_{i \in \{1, \dots, n\}} \mathbf{T}(t_i)$, for $\alpha \in \{\otimes, \wp\}$. In particular, we write $\mathbf{V}(t)$ for $\mathbf{T}(t) \cap V$. A *tree* is a raw tree t such that if $\alpha(t_1, \dots, t_n) \in \mathbf{T}(t)$ (with $\alpha = \otimes$ or \wp), then the sets $\mathbf{V}(t_i)$ for $1 \leq i \leq n$ are pairwise disjoint: in other words, each variable x occurs at most once in t . A tree t is *strict* if $\{\otimes(), \wp()\} \cap \mathbf{T}(t) = \emptyset$.

From now on, we will consider strict trees only, i.e. we rule out the multiplicative units. This restriction will play a crucial rôle in expressing and establishing the bounds of Sections 3 and 4. It is possible to generalize our results in presence of units: we postpone the discussion on this subject to Section 6.²

² An additional consequence is the fact that, given a (strict) tree t , any other tree u occurs at most once as a subtree of t : e.g., in $\wp^2(t_1, t_2)$, $\mathbf{V}(t_1)$ and $\mathbf{V}(t_2)$ are both non empty and disjoint, so that $t_1 \neq t_2$. In other words, we can identify $\mathbf{T}(t)$ with the positions of subtrees in t , that play the rôle of

A *cut* is an unordered pair $c = \langle t|s \rangle$ of trees such that $\mathbf{V}(t) \cap \mathbf{V}(s) = \emptyset$, and then we set $\mathbf{T}(c) = \mathbf{T}(t) \cup \mathbf{T}(s)$. A *reducible cut* is a cut $\langle t|s \rangle$ such that t is a variable and $\bar{t} \notin \mathbf{V}(s)$, or such that we can write $t = \otimes(t_1, \dots, t_n)$ and $s = \wp(s_1, \dots, s_n)$, or *vice versa*. Note that, in the absence of typing, we do not require all cuts to be reducible, as this would not be stable under cut elimination.

Given a set A , we denote by \vec{a} any finite family of elements of A . In general, we abusively identify \vec{a} with any enumeration $(a_1, \dots, a_n) \in A^n$ of its elements, and write \vec{a}, \vec{b} for the union of disjoint families \vec{a} and \vec{b} . If $\vec{\gamma}$ is a family of trees or cuts, we write $\mathbf{V}(\vec{\gamma}) = \bigcup_{\gamma \in \vec{\gamma}} \mathbf{V}(\gamma)$ and $\mathbf{T}(\vec{\gamma}) = \bigcup_{\gamma \in \vec{\gamma}} \mathbf{T}(\gamma)$. An MLL *proof net* is a pair $p = (\vec{c}; \vec{t})$ of a finite family \vec{c} of cuts and a finite family \vec{t} of trees, such that for all cuts or trees $\gamma, \gamma' \in \vec{c}, \vec{t}$, $\mathbf{V}(\gamma) \cap \mathbf{V}(\gamma') = \emptyset$, and such that for any $x \in \mathbf{V}(p) = \mathbf{V}(\vec{c}) \cup \mathbf{V}(\vec{t})$, we have $\bar{x} \in \mathbf{V}(p)$ too. We then write $\mathbf{C}(p) = \vec{c}$.

2.2 Cut elimination

The *substitution* $\gamma[t/x]$ of a tree t for a variable x in a tree (or cut, or net) γ is defined in the usual way. By the definition of trees, we notice that this substitution is essentially linear, since each variable x appears at most once in a tree.

There are two basic cut elimination steps, one for each kind of reducible cut:

- the elimination of a connective cut yields a family of cuts: we write $\langle \otimes(t_1, \dots, t_n) | \wp(s_1, \dots, s_n) \rangle \rightarrow_m \langle (t_i | s_i)_{i \in \{1, \dots, n\}} \rangle$ that we extend to nets by setting $(c, \vec{c}; \vec{t}) \rightarrow_m (\vec{c}', \vec{c}; \vec{t})$ whenever $c \rightarrow_m \vec{c}'$;
- the elimination of an axiom cut generates a substitution: we write $(\langle x|t \rangle, \vec{c}; \vec{t}) \rightarrow_{ax} (\vec{c}; \vec{t})[t/\bar{x}]$ whenever $\bar{x} \notin \mathbf{V}(t)$.

We are in fact interested in the simultaneous elimination of any number of reducible cuts, that we describe as follows: we write $p \rightrightarrows p'$ if $p = (\langle x_1|t_1 \rangle, \dots, \langle x_n|t_n \rangle, c_1, \dots, c_k, \vec{c}; \vec{t})$ and $p' = (\vec{c}'_1, \dots, \vec{c}'_k, \vec{c}; \vec{t})[t_1/\bar{x}_1] \cdots [t_n/\bar{x}_n]$, with $c_i \rightarrow_m \vec{c}'_i$ for $1 \leq i \leq k$, and $\bar{x}_i \notin \mathbf{V}(t_j)$ for $1 \leq i \leq j \leq n$. We moreover write $p \rightrightarrows_m p'$ (resp. $p \rightrightarrows_{ax} p'$) in case $n = 0$ (resp. $k = 0$). It is a simple exercise to check that if $p \rightrightarrows p'$ then there exists q such that $p \rightrightarrows_m q \rightrightarrows_{ax} p'$: the converse does not hold, though, as the elimination of connective cuts may generate new axiom cuts.

2.3 Paths

In order to control the effect of parallel reduction on the size of proof nets, we rely on a side condition involving the number of cuts crossed by switching paths, i.e. paths in the sense of Danos–Regnier’s correctness criterion [3].

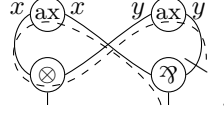
In our setting, a *switching* of a net p is a partial map $I : \mathbf{T}(p) \rightarrow \mathbf{T}(p)$ such that, for each $t = \wp(t_1, \dots, t_n) \in \mathbf{T}(p)$, $I(t) \in \{t_1, \dots, t_n\}$. Given a net p and a switching I of p , we define adjacency relations between the elements of $\mathbf{T}(p)$, written $\sim_{t,s}$ for $t, s \in \mathbf{T}(p)$ and \sim_c for $c \in \mathbf{C}(p)$, as the least symmetric relations such that:

- for any $x \in \mathbf{V}(p)$, $x \sim_{x, \bar{x}} \bar{x}$;
- for any $t = \otimes(t_1, \dots, t_n) \in \mathbf{T}(p)$, $t \sim_{t, t_i} t_i$ for each $i \in \{1, \dots, n\}$;
- for any $t = \wp(t_1, \dots, t_n) \in \mathbf{T}(p)$, $t \sim_{t, I(t)} I(t)$;
- for any $c = \langle t|s \rangle \in \mathbf{C}(p)$, $t \sim_c s$.

vertices when considering t as a graphical structure. This will allow us to keep notations concise in our treatment of paths. This trick is of course inessential for our results.

Whenever necessary, we may write, e.g., $\sim_{t,s}^p$ or $\sim_{t,s}^{p,I}$ for $\sim_{t,s}$ to make the underlying net and switching explicit. Let l and $m \in (\mathbf{T}(p) \times \mathbf{T}(p)) \cup \mathbf{C}(p)$ be two adjacency labels: we write $l \equiv m$ if $l = m$ or $m = (x, \bar{x})$ and $l = (\bar{x}, x)$ for some $x \in V$.

Given a switching I in p , an I -path is a sequence of trees t_0, \dots, t_n of $\mathbf{T}(p)$ such that there exists a sequence of pairwise \neq labels l_1, \dots, l_n with, for each $i \in \{1, \dots, n\}$, $t_{i-1} \sim_{l_i}^{p,I} t_i$.³ For instance, if $p = (; \otimes(x, y), \mathfrak{A}(\bar{y}, \bar{x}))$ and $I(\mathfrak{A}(\bar{y}, \bar{x})) = \bar{x}$, then the chain of adjacencies $\mathfrak{A}(\bar{x}, \bar{y}) \sim_{\mathfrak{A}(\bar{x}, \bar{y}), \bar{x}} \bar{x} \sim_{x, \bar{x}} x \sim_{\otimes(x, y), x} \otimes(x, y) \sim_{\otimes(x, y), y} y \sim_{y, \bar{y}} \bar{y}$ defines an I -path in p , which can be depicted as the dashed line in the following graphical representation of p :



We call *path* in p any I -path for I a switching of p , and we write $\mathbf{P}(p)$ for the set of all paths in p . We write $t \rightsquigarrow s$ or $t \rightsquigarrow_p s$ whenever there exists a path from t to s in p . Given $\chi = t_0, \dots, t_n \in \mathbf{P}(p)$, we call *subpaths* of χ the subsequences of χ : a subpath is either the empty sequence ϵ or a path of p . We moreover write $\bar{\chi}$ for the reverse path: $\bar{\chi} = t_n, \dots, t_0 \in \mathbf{P}(p)$. We say a net p is *acyclic* if for all $\chi \in \mathbf{P}(p)$ and $t \in \mathbf{T}(p)$, t occurs at most once in χ : in other words, there is no *cycle* t, χ, t . From now on, we consider acyclic nets only: it is well known that if p is acyclic and $p \rightrightarrows q$ then q is acyclic too.

If $c = \langle t|s \rangle \in \mathbf{C}(p)$, we may write χ_1, c, χ_2 for either χ_1, s, t, χ_2 or χ_1, t, s, χ_2 : by acyclicity, this notation is unambiguous, unless $\chi_1 = \chi_2 = \epsilon$.

For all $\chi \in \mathbf{P}(p)$, we write $\mathbf{cc}_p(\chi)$, or simply $\mathbf{cc}(\chi)$, for the number of cuts *crossed* by χ : $\mathbf{cc}_p(\chi) = \#\{\langle t|s \rangle \in \mathbf{C}(p) \mid t \in \chi\}$ (recall that cuts are unordered). Observe that, by acyclicity, a path χ crosses each cut $c = \langle t|s \rangle$ at most once: either $\chi = \chi_1, c, \chi_2$, or $\chi = \chi_1, t, \chi_2$, or $\chi = \chi_1, s, \chi_2$, with neither t nor s occurring in χ_1, χ_2 . Finally, we write $\mathbf{cc}(p) = \max\{\mathbf{cc}(\chi) \mid \chi \in \mathbf{P}(p)\}$: in the following, we show that the maximal number of cuts crossed by any switching path is a good parameter to limit the decrease in size induced by parallel reduction.

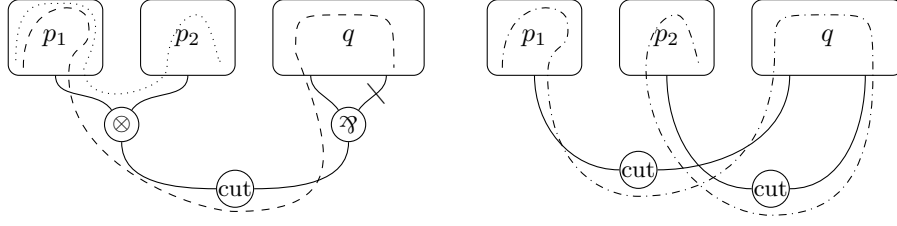
3 Variations of $\mathbf{cc}(p)$ under reduction

Here we establish that the possible increase of $\mathbf{cc}(p)$ under reduction is bounded. It should be clear that if $p \rightrightarrows_{ax} q$ then $\mathbf{cc}(q) \leq \mathbf{cc}(p)$: intuitively, the only effect of \rightrightarrows_{ax} is to straighten some paths, thus decreasing the number of crossed cuts. In the case of connective cuts however, cuts are duplicated and new paths are created.

Consider for instance a net r , as in Fig. 3, obtained from three nets p_1, p_2 and q , by forming the cut $\langle \otimes(t_1, t_2) | \mathfrak{A}(s_1, s_2) \rangle$ where $t_1 \in \mathbf{T}(p_1)$, $t_2 \in \mathbf{T}(p_2)$ and $s_1, s_2 \in \mathbf{T}(q)$. Observe that, in the reduct r' obtained by forming two cuts $\langle t_1 | s_1 \rangle$ and $\langle t_2 | s_2 \rangle$, we may very well form a path that travels from p_1 to q then p_2 ; while in p , this is forbidden by any switching of $\mathfrak{A}(s_1, s_2)$. For instance, if we consider $I(\mathfrak{A}(s_1, s_2)) = s_1$, we may only form a path between p_1 and p_2 through $\otimes(t_1, t_2)$, or a path between q and one of the p_i 's, through s_1 and the cut.

In the remainder of this section, we fix a reduction step $p \rightrightarrows_m q$, and we show that the previous example describes a general mechanism: if a new path is created in this step $p \rightrightarrows_m q$, it must involve a path ξ between two premises of a \mathfrak{A} involved in a cut c of p , *unfolded* into a path between the residuals of this cut. We call such an intermediate path ξ a *slipknot*.

³ In standard terminology of graph theory, an I -path in p is a trail in the unoriented graph with vertices in $\mathbf{T}(p)$ and edges given by the sum of adjacency relations defined by I (identifying $\sim_{x, \bar{x}}$ with $\sim_{\bar{x}, x}$). The only purpose of our choice of labels for adjacency relations and the definition of \equiv is indeed to capture this notion of path in the unoriented graph of subtrees induced by a switching in a net.



■ **Figure 3** A cut, the resulting slipknot, and examples of paths before and after reduction.

3.1 Residual cuts and slipknots

Notice that $\mathbf{T}(q) \subseteq \mathbf{T}(p)$. Observe that, given a switching J of q , it is always possible to extend J into a switching I of p , so that, for all $t, s \in \mathbf{T}(q)$:

- if $t \sim_{t,s}^{q,J} s$ then $t \sim_{t,s}^{p,I} s$, and
- if $c \in \mathbf{C}(p)$ and $t \sim_c^{q,J} s$ then $t \sim_c^{p,I} s$.

To determine I uniquely, it remains only to select a premise for each \mathfrak{A} involved in an eliminated cut. Consider $c = \langle \otimes(t_1, \dots, t_n) | \mathfrak{A}(s_1, \dots, s_n) \rangle \in \mathbf{C}(p)$ and assume c is eliminated in the reduction $p \Rightarrow_m q$. Then the *residuals* of c in q are the cuts $\langle t_i | s_i \rangle \in \mathbf{C}(q)$ for $1 \leq i \leq n$.

If $\xi \in \mathbf{P}(q)$, a *slipknot* of ξ is any pair (d, d') of (necessarily distinct) residuals in q of a cut in p , such that we can write $\xi = \chi_1, d, \chi_2, d', \chi_3$. We now show that a path in q is necessarily obtained by alternating paths in p and paths between slipknots, that recursively consist of such alternations. This will allow us to bound $\mathbf{cc}(q)$ depending on $\mathbf{cc}(p)$, by reasoning inductively on these paths. The main tool is the following lemma:

► **Lemma 1.** *If $\xi \in \mathbf{P}(q)$ then there exists a path $\xi^- \in \mathbf{P}(p)$ with the same endpoints as ξ .*

Proof. Assuming ξ is a J -path of q , we construct an I -path ξ^- in p with the same endpoints as ξ for an extension I of J as above. The definition is by induction on the number of residuals occurring as subpaths of ξ . In the process, we must ensure that the constraints we impose on I in each induction step can be satisfied globally: the trick is that we fix the value of $I(\mathfrak{A}(\vec{s}))$ only in case exactly one residual of the cut involving $\mathfrak{A}(\vec{s})$ occurs in ξ .

First consider the case of $\xi = \chi_1, d, \chi_2, d', \chi_3$, for a slipknot (d, d') , where d and d' are residuals of $c \in \mathbf{C}(p)$. We can assume, w.l.o.g., that: (i) no other residual of c occurs in χ_1 , nor in χ_3 ; (ii) no residual of a cut $c' \neq c$ occurs in both χ_1 and χ_3 . By the definition of residuals, we can write $c = \langle \otimes(\vec{t}) | \mathfrak{A}(\vec{s}) \rangle \in \mathbf{C}(p)$, $d = \langle t | s \rangle$ and $d' = \langle t' | s' \rangle$ with $t, t' \in \vec{t}$ and $s, s' \in \vec{s}$. It is then sufficient to prove that $\xi = \chi_1, t, s, \chi_2, s', t', \chi_3$, in which case we can set $\xi^- = \chi_1^-, t, \otimes(\vec{t}), t', \chi_3^-$, where χ_1^- and χ_3^- are obtained from the induction hypothesis (or by setting $\epsilon^- = \epsilon$ for empty subpaths): by condition (ii), the constraints we impose on I by forming χ_1^- and χ_3^- are independent.

Let us rule out the other three orderings of d and d' : (a) $\xi = \chi_1, s, t, \chi_2, t', s', \chi_3$, (b) $\xi = \chi_1, s, t, \chi_2, s', t', \chi_3$ or (c) $\xi = \chi_1, t, s, \chi_2, t', s', \chi_3$. First observe that χ_2 is not empty. Indeed, if $t \sim_l^q t'$ (or $t \sim_l^q s'$, or $s \sim_l^q t'$) then: l cannot be a cut of q because $\langle t | s \rangle$ and $\langle t' | s' \rangle \in \mathbf{C}(q)$; l cannot be of the form $\langle \alpha(t_1, \dots, t_n), t_n \rangle$ because the trees t, t', s, s' are pairwise disjoint; so l must be an axiom and we obtain a cycle in q .

Let u and v be the endpoints of χ_2 , and consider $\chi_2^- \in \mathbf{P}(p)$ with the same endpoints, obtained by induction hypothesis. Necessarily, we have $t \sim_l^{q,J} u$ in cases (a) and (b), $s \sim_l^{q,J} u$ in case (c), $t' \sim_m^{q,J} v$ in cases (a) and (c), and $s' \sim_m^{q,J} v$ in case (b), where $l \neq m$, and nor l nor m is a cut: it follows that the same adjacencies hold in p for any extension I of J . Observe that $\otimes(\vec{t}) \notin \chi_2^-$: otherwise, we would obtain a path $t \rightsquigarrow_p \otimes(\vec{t})$ (or $\otimes(\vec{t}) \rightsquigarrow_p t'$) that we

could extend into a cycle. Then in case (a), we obtain a cycle in p directly: $t, \chi_2^-, t', \otimes(\vec{t}), t$. In cases (b) and (c), we deduce that $\mathfrak{A}(\vec{s}) \notin \chi_2^-$, and we obtain a cycle, e.g. in case (b): $t, \chi_2^-, s', \mathfrak{A}(\vec{s}), \otimes(\vec{t}), t'$, for any I such that $I(\mathfrak{A}(\vec{s})) = s'$.

We can now assume that each cut of p has at most one residual occurring as a subpath of ξ . If no residual occurs in ξ , then we can set $\xi^- = \xi$. Now fix $c = \langle \otimes(\vec{t}) | \mathfrak{A}(\vec{s}) \rangle \in \mathbf{C}(p)$ and assume, w.l.o.g (otherwise, consider $\bar{\xi}$), that $\xi = \chi_1, t, s, \chi_2$ with $t \in \vec{t}$ and $s \in \vec{s}$. Then we set $I(\mathfrak{A}(\vec{s})) = s$ and $\xi^- = \chi_1^-, t, c, s, \chi_2^- \in \mathbf{P}(p)$: this is the only case in which we impose a value for I to construct ξ^- , so this choice, and the choices we make to form χ_1^- and χ_2^- are all independent. \blacktriangleleft

► **Lemma 2.** *If $\xi \in \mathbf{P}(q)$ and $c = \langle \otimes(\vec{t}) | \mathfrak{A}(\vec{s}) \rangle \in \mathbf{C}(p)$, then at most two residuals of c occur as subpaths of ξ , and then we can write $\xi = \chi_1, t, s, \chi_2, s', t', \chi_3$ with $t, t' \in \vec{t}$ and $s, s' \in \vec{s}$.*

Proof. Assume $\xi = \chi_1, d, \chi_2, d', \chi_3$ and $d = \langle t | s \rangle$ and $d' = \langle t' | s' \rangle$ with $t, t' \in \vec{t}$ and $s, s' \in \vec{s}$. Using Lemma 1, we establish that $\xi = \chi_1, t, s, \chi_2, s', t', \chi_3$: we can exclude the other cases exactly as in the proof of Lemma 1. Then, as soon as three residuals of c occur in ξ , a contradiction follows. \blacktriangleleft

► **Lemma 3.** *Slipknots are well-bracketed in the following sense: there is no path $\xi = d_1, \chi_1, d_2, \chi_2, d'_1, \chi_3, d'_2 \in \mathbf{P}(q)$ such that both (d_1, d'_1) and (d_2, d'_2) are slipknots.*

Proof. Assume $c_1 = \langle \otimes(\vec{t}_1) | \mathfrak{A}(\vec{s}_1) \rangle$, $c_2 = \langle \otimes(\vec{t}_2) | \mathfrak{A}(\vec{s}_2) \rangle$, and, for $1 \leq i \leq 2$, $d_i = (t_i, s_i)$ and $d'_i = (t'_i, s'_i)$, with $t_i, t'_i \in \vec{t}_i$ and $s_i, s'_i \in \vec{s}_i$. By the previous lemma, we must have $\xi = t_1, s_1, \chi_1, t_2, s_2, \chi_2, s'_1, t'_1, \chi_3, s'_2, t'_2$. Observe that nor χ_1^- nor χ_3^- can cross c_1 or c_2 : otherwise, we obtain a cycle in p . Then $s_1, \chi_1^-, t_2, c_1, s'_2, \chi_3^-, t'_1, c_2, s_1$ is a cycle in p . \blacktriangleleft

► **Corollary 4.** *Any path of q is of the form $\zeta_1, c_1, \chi_1, c'_1, \zeta_2, \dots, \zeta_n, c_n, \chi_n, c'_n, \zeta_{n+1}$ where each subpath ζ_i is without slipknot, and each (c_i, c'_i) is a slipknot.*

The previous result describes precisely how paths in q are related with those in p : it will be crucial in the following.

3.2 Bounding the growth of \mathbf{cc}

Now we show that we can bound $\mathbf{cc}(q)$ depending only on $\mathbf{cc}(p)$. For each $\xi \in \mathbf{P}(q)$, we define the *width* $w_p(\xi)$ (or just $w(\xi)$): $w_p(\xi) = \max\{\mathbf{cc}_p(\chi^-) | \chi \text{ subpath of } \xi\}$. We have:

► **Lemma 5.** *For any path $\zeta \in \mathbf{P}(q)$, $\mathbf{cc}_p(\zeta^-) \leq w_p(\zeta) \leq \mathbf{cc}(p)$ and $w_p(\zeta) \leq \mathbf{cc}_q(\zeta)$. If moreover ζ has no slipknot, then $w_p(\zeta) = \mathbf{cc}_q(\zeta) = \mathbf{cc}_p(\zeta^-)$.*

Defining $\varphi : \mathbb{N} \rightarrow \mathbb{N}$ by $\varphi(0) = 0$ and $\varphi(n+1) = 2(n+1) + (n+1)(\varphi(n))$, we obtain:

► **Lemma 6.** *If $\xi \in \mathbf{P}(q)$ then $\mathbf{cc}(\xi) \leq \varphi(w_p(\xi))$.*

Proof. The proof is by induction on $w(\xi)$. If $w(\xi) = 0$, then we can easily check that $\mathbf{cc}(\xi) = 0$. Otherwise assume $w(\xi) = n+1$. Then we set $\xi = \zeta_1, c_1, \chi_1, c'_1, \zeta_2, \dots, \zeta_k, c_k, \chi_k, c'_k, \zeta_{k+1}$ as in Corollary 4.

First observe that for all $i \in \{1, \dots, k\}$, $w(\chi_i) \leq w(\xi) - 1$. Indeed, c_i, χ_i is a subpath of ξ and $w(c_i, \chi_i) = w(\chi_i) + 1$ by the definition of width. So, by induction hypothesis, $\mathbf{cc}(\chi_i) \leq \varphi(n)$. We also have that $\sum_{i=1}^{k+1} \mathbf{cc}(\zeta_i) \leq w(\xi) - k$. Observe indeed that $\mathbf{cc}(\xi^-) = \sum_{i=1}^{k+1} \mathbf{cc}(\zeta_i) + k$, because of Lemma 5 applied to ζ_i , and because of the construction of ξ^- that contracts the slipknots c_i, χ_i, c'_i ; also recall that $\mathbf{cc}(\xi^-) \leq w(\xi)$.

We obtain:

$$\mathbf{cc}(\xi) = \sum_{1 \leq i \leq k} \mathbf{cc}(\chi_i) + \sum_{1 \leq j \leq k+1} \mathbf{cc}(\zeta_j) + 2k \leq k\varphi(n) + w(\xi) - k + 2k$$

and, since $k \leq \mathbf{cc}(\xi^-) \leq w(\xi) = n+1$, we obtain $\mathbf{cc}(\xi) \leq (n+1)\varphi(n) + 2(n+1) = \varphi(n+1)$. ◀

Using Lemma 5 again, we obtain:

► **Corollary 7.** *Let $p \rightrightarrows_m q$. Then, $\mathbf{cc}(q) \leq \varphi(\mathbf{cc}(p))$.*

► **Remark.** It is in fact possible to show that $\mathbf{cc}(q) \leq 2n!\mathbf{cc}(p)$, which is a better bound and closer to the graphical intuition, but the proof is much longer, and we are only interested in the existence of a bound.

4 Bounding the size of antireducts

For any tree, cut or net γ , we define the *size* of γ as $\#\gamma = \mathbf{card}(\mathbf{T}(\gamma))$: graphically, $\#p$ is nothing but the number of wires in p . In this section, we show that the loss of size during parallel reduction is directly controlled by $\mathbf{cc}(p)$ and $\#q$: more precisely, we show that the ratio $\frac{\#p}{\#q}$ is bounded by a function of $\mathbf{cc}(p)$.

First observe that the elimination of multiplicative cuts cannot decrease the size by more than a half:

► **Lemma 8.** *If $p \rightrightarrows_m q$ then $\#p \leq 2\#q$.*

Proof. It is sufficient to observe that if $c \rightarrow_m \vec{c}$ then $\#c = 2 + \#\vec{c} \leq 2\#\vec{c}$.⁴ ◀

4.1 Elimination of axiom cuts

Observe that:

- if $x \in \mathbf{V}(\gamma)$ then $\#\gamma[t/x] = \#\gamma + \#t - 1$;
- if $x \notin \mathbf{V}(\gamma)$ then $\#\gamma[t/x] = \#\gamma$.

It follows that, in the elimination of a single axiom cut $p \rightarrow_{ax} q$, we have $\#p = \#q + 1$. But we cannot reproduce the proof of Lemma 8 for \rightrightarrows_{ax} : as stated in our introduction, chains of axiom cuts reducing into a single wire are the source of the collapse of size. We can bound the length of those chains by $\mathbf{cc}(p)$, however, and this allows us to bound the loss of size during reduction.

► **Lemma 9.** *If $p \rightrightarrows_{ax} q$ then $\#p \leq (2\mathbf{cc}(p) + 1)\#q$.*

Proof. Assume $p = (\langle x_1|t_1 \rangle, \dots, \langle x_n|t_n \rangle, \vec{c}; \vec{s})$ and $q = (\vec{c}; \vec{s})[t_1/\bar{x}_1] \cdots [t_n/\bar{x}_n]$ with $\bar{x}_i \notin \mathbf{V}(t_j)$ for $1 \leq i \leq j \leq n$. In case $\mathbf{cc}(p) = 0$, we have $n = 0$ and $p = q$ so the result is obvious. We thus assume $\mathbf{cc}(p) > 0$: to establish the result in this case, we make the chains of eliminated axiom cuts explicit.

Due to the condition on free variables, there exists a (necessarily unique) permutation of $\langle x_1|t_1 \rangle, \dots, \langle x_n|t_n \rangle$ yielding a family of the form $\vec{c}_1, \dots, \vec{c}_k$ such that:

⁴ This is due to the fact that all the trees are strict, so \vec{c} is not empty and $\#\vec{c} \geq 1$. Without the strictness condition, we would have to deal with annihilating reductions $(\otimes()) \vartheta () \rightarrow_m \epsilon$: this will be discussed in the conclusion.

- for $1 \leq i \leq k$, we can write $\vec{c}_i = \langle x_0^i | \bar{x}_1^i \rangle, \dots, \langle x_{n_i-1}^i | \bar{x}_{n_i}^i \rangle, \langle x_{n_i}^i | t^i \rangle$;
- each \vec{c}_i is maximal with this shape, i.e. $\bar{x}_0^i \notin \{x_1, \dots, x_n, t_1, \dots, t_n\}$ and, in case t^i is a variable, $\bar{t}^i \notin \{x_1, \dots, x_n, t_1, \dots, t_n\}$;
- if $i < j$, then the cut $\langle x_{n_i}^i | t^i \rangle$ occurs before $\langle x_{n_j}^j | t^j \rangle$ in $\langle x_1 | t_1 \rangle, \dots, \langle x_n | t_n \rangle$.

It follows that if $\bar{x}_0^i \in \mathbf{V}(t_j)$ then $j < i$, and then $q = (\vec{c}; \vec{s})[t^1/\bar{x}_0^1] \dots [t^k/\bar{x}_0^k]$, by applying the same permutation to the substitutions as we did to cuts: we can do so because, by a standard argument, if $x \neq y$, $x \notin \mathbf{V}(u)$ and $y \notin \mathbf{V}(u)$ then $\gamma[u/x][v/y] = \gamma[v/y][u/x]$.

For $1 \leq i \leq k$, since \vec{c}_i is a chain of $n_i + 1$ cuts, it follows that $n_i \leq \mathbf{cc}(p) - 1$. So $\#p = \#\vec{c} + \#\vec{s} + \sum_{i=1}^k (\#t^i + 2n_i + 1) \leq \#\vec{c} + \#\vec{s} + \sum_{i=1}^k \#t^i + k(2\mathbf{cc}(p) - 1)$. Moreover $\#q = \#\vec{c} + \#\vec{s} + \sum_{i=1}^k \#t^i - k$. It follows that $\#p \leq \#q + 2k\mathbf{cc}(p)$ and, to conclude, it will be sufficient to prove that $\#q \geq k$.

For $1 \leq i \leq k$, let $A_i = \{j > i \mid \bar{x}_0^j \in \mathbf{V}(t^i)\}$, and then let $A_0 = \{i \mid \bar{x}_0^i \in \mathbf{V}(\vec{c}, \vec{s})\}$. It follows from the construction that $\{A_0, \dots, A_{k-1}\}$ is a partition (possibly including empty sets) of $\{1, \dots, k\}$. By construction, $\#t^i > \mathbf{card}(A_i)$. Now consider $q_i = (\vec{c}; \vec{s})[t^1/\bar{x}_0^1] \dots [t^i/\bar{x}_0^i]$ for $0 \leq i \leq k$ so that $q = q_k$. For $1 \leq i \leq k$, we obtain $\#q_i = \#q_{i-1} + \#t^i - 1 \geq \#q_{i-1} + \mathbf{card}(A_i)$. Also observe that $\#q_0 = \#(\vec{c}; \vec{s}) \geq \mathbf{card}(A_i)$. We can then conclude: $\#q = \#q_k \geq \sum_{i=0}^k \mathbf{card}(A_i) = k$. ◀

4.2 General case

Recall that any parallel cut elimination step $p \Rightarrow q$ can be decomposed into a multiplicative-then-axiom pair of reductions: $p \Rightarrow_m q' \Rightarrow_{ax} q$. This allows us to bound the loss of size in the reduction $p \Rightarrow q$, using the previous results:

► **Theorem 10.** *If $p \Rightarrow q$ then $\#p \leq 4(\varphi(\mathbf{cc}(p)) + 1)\#q$.*

Proof. Consider first q' such that $p \Rightarrow_m q'$ and $q' \Rightarrow_{ax} q$. By Lemma 8, $\#p \leq 2\#q'$. Lemma 9 states that $\#q' \leq (2\mathbf{cc}(q') + 1)\#q$. Finally, Corollary 7, entails that $\mathbf{cc}(q') \leq \varphi(\mathbf{cc}(p))$, and we can conclude: $\#p \leq 2(\varphi(\mathbf{cc}(p)) + 1)\#q \leq 4(\varphi(\mathbf{cc}(p)) + 1)\#q$. ◀

► **Corollary 11.** *If q is an MLL net and $n \in \mathbb{N}$, then $\{p \mid p \Rightarrow q \text{ and } \mathbf{cc}(p) \leq n\}$ is finite.*

To be precise, due to our term syntax, the previous corollary holds only up to renaming variables in axioms: we keep this precision implicit in the following.

It follows that, given an infinite linear combination of $\sum_{i \in I} a_i.p_i$, such that $\{\mathbf{cc}(p_i) \mid i \in I\}$ is finite, we can always consider an arbitrary family of reductions $p_i \Rightarrow q_i$ for $i \in I$ and form the sum $\sum_{i \in I} a_i.q_i$: this is always well defined.

5 Taylor expansion

We now show how the previous results apply to Taylor expansion. For that purpose, we must extend our syntax to MELL proof nets. Our presentation departs from Ehrhard's [11] in our treatment of promotion boxes: instead of introducing boxes as tree constructors labelled by nets, with auxiliary ports as inputs, we consider box ports as 0-ary trees, that are related with each other in a *box context*, associating each box with its contents. This is in accordance with the usual presentation of promotion as a black box, and has two motivations:

- In Ehrhard's syntax, the promotion is not a net but an open tree, for which the trees associated with auxiliary ports must be mentioned explicitly: this would complicate the expression of Taylor expansion.
- The *nouvelle syntaxe* imposes constraints on auxiliary ports, that are easier to express when these ports are directly represented in the syntax.

Then we show that if p is a resource net in the support of the Taylor expansion of an MELL proof net P , then $\text{cc}(p)$ (and in fact the length of any path in p) is bounded by a function of P .

Observe that we need only consider the support of Taylor expansion, so we do not formalize the expansion of MELL nets into infinite linear combinations of resource nets: rather, we introduce $\mathcal{T}(P)$ as a set of approximants. Also, as we limit our study to *strict* nets, we will restrict $\mathcal{T}(P)$ to those approximants that take at least one copy of each box of P : this is enough to cover the case of weakening-free MELL.

5.1 MELL nets

In addition to the set of variables, we fix a denumerable set \mathcal{A} of *box ports*: we assume given an enumeration $\mathcal{A} = \{a_i^b \mid i, b \in \mathbb{N}\}$. We call *principal ports* the ports a_0^b and *auxiliary ports* the other ports. In the so-called *nouvelle syntaxe* of MELL, contractions and derelictions are merged together in a generalized contraction cell, and auxiliary ports must be premises of such generalized contractions.

We introduce the corresponding term syntax, as follows. Raw pre-trees (S°, T° , etc.) and raw trees (S, T , etc.) are defined by mutual induction as follows:

$$T ::= x \mid a_0^b \mid \otimes(T_1, \dots, T_n) \mid \wp(T_1, \dots, T_n) \mid ?(T_1^\circ, \dots, T_n^\circ) \quad \text{and} \quad T^\circ ::= T \mid a_{i+1}^b$$

requiring that each \otimes, \wp and $?$ is of arity at least 1. We write $\mathbf{V}(S)$ (resp. $\mathbf{B}(S)$) for the set of variables (resp. of principal and auxiliary ports) occurring in S . A *tree* (resp. a *pre-tree*) is a raw tree (resp. raw pre-tree) in which each variable and port occurs at most once. A *cut* is an unordered pair of trees $C = \langle T \mid S \rangle$ with disjoint sets of variables and ports.

We now define *box contexts* and *pre-nets* by mutual induction as follows. A box context Θ is the data of a finite set $\mathcal{B}_\Theta \subset \mathbb{N}$, and, for each $b \in \mathcal{B}_\Theta$, a closed pre-net $\Theta(b)$, of the form $(\Theta_b; \vec{C}_b; T_b, \vec{S}_b^\circ)$. Then we write $\vec{S}_b^\circ = S_{b,1}^\circ, \dots, S_{b,n_b}^\circ$. A pre-net is a triple $P^\circ = (\Theta; \vec{C}; \vec{S}^\circ)$ where Θ is a box context, each variable and port occurs at most once in \vec{C}, \vec{S}° , and moreover, if $a_i^b \in \mathbf{B}(\vec{C}; \vec{S}^\circ)$ then $b \in \mathcal{B}_\Theta$ and $i \leq n_b$. A closed pre-net is a pre-net $P^\circ = (\Theta; \vec{C}; \vec{S}^\circ)$ such that x occurs iff \bar{x} occurs, and moreover, if $b \in \mathcal{B}_\Theta$ then each a_i^b with $0 \leq i \leq n_b$ occurs. Then a *net* is a closed pre-net of the form $P = (\Theta; \vec{C}; \vec{S})$.

We write $\mathbf{T}(\gamma)$ for the set of sub-pre-trees of a pre-tree, or cut, or pre-net γ : the definition extends that for subtrees in MLL nets, moreover setting $\mathbf{T}(a) = \{a\}$ for any $a \in \mathcal{A}$ (so we do not look into the content of boxes). As for MLL, we set $\#\gamma = \mathbf{card}(\mathbf{T}(\gamma))$. We write $\mathbf{depth}(P^\circ)$ for the maximum level of nesting of boxes in P° , i.e. the inductive depth in the previous definition. Also, the size of MELL pre-nets includes that of their boxes: we set $\mathbf{size}(P^\circ) = \#P^\circ + \sum_{b \in \mathcal{B}_\Theta} \mathbf{size}(\Theta(b))$.

We extend the switching functions of MLL to $?$ links: for each $T = ?(T_1, \dots, T_n)$, $I(T) \in \{T_1, \dots, T_n\}$, which induces a new adjacency relation $T \sim_{T, I(T)} I(T)$. We also consider adjacency relations \sim_b for $b \in \mathcal{B}_\Theta$, setting $a_i^b \sim_b a_j^b$ whenever $0 \leq i < j \leq n_b$: w.r.t. paths, a box behaves like an $(n_b + 1)$ -ary axiom link and the contents is not considered. We write $\mathbf{P}(P^\circ)$ for the set of paths in P° . We say a pre-net P° is *acyclic* if there is no cycle in $\mathbf{P}(P^\circ)$ and, inductively, each $\Theta(b)$ is acyclic. From now on, we consider acyclic pre-nets only.

5.2 Resource nets and Taylor expansion

The Taylor expansion of a net P will be a set of *resource nets*: these are the same as the multiplicative nets introduced before, except we have two new connectives $!$ and $?$. Raw trees are given as follows:

$$t ::= x \mid \otimes(t_1, \dots, t_n) \mid \wp(t_1, \dots, t_n) \mid !(t_1, \dots, t_n) \mid ?(t_1, \dots, t_n).$$

Again, we will consider strict trees only: each \otimes , \wp , $!$ and $?$ is of arity at least 1. In resource nets, we extend switchings to $?$ links as in MELL nets, and for each $t = ?(t_1, \dots, t_n)$, we set $t \sim_{t, I(t)} I(t)$. Moreover, for each $t = !(t_1, \dots, t_n)$, we set $t \sim_{t, t_i} t_i$ for $1 \leq i \leq n$.

We are now ready to introduce the expansion of MELL nets. During the construction, we need to track the conclusions of copies of boxes, in order to collect copies of auxiliary ports in the external $?$ links: this is the rôle of the intermediate notion of pre-Taylor expansion.

► **Definition 12.** Taylor expansion is defined by induction on depth as follows. Given a closed pre-net $P^\circ = (\Theta; \vec{C}; \vec{S}^\circ)$, a *pre-Taylor expansion* of P° is any pair (p, f) of a resource net $p = (\vec{c}; \vec{t})$, together with a function $f : \vec{t} \rightarrow \vec{S}^\circ$ such that $f^{-1}(T)$ is a singleton whenever $T \in \vec{S}^\circ$ is a tree, obtained as follows:

- for each $b \in \mathcal{B}_\Theta$, fix a number $k_b > 0$ of copies;
- for $1 \leq j \leq k_b$, fix a pre-Taylor expansion (p_j^b, f_j^b) of $\Theta(b)$, and write $p_j^b = (\vec{c}_j^b; t_j^b, \vec{s}_j^b)$ so that $f_j^b(t_j^b) = T_b$;
- up to renaming the variables of the p_j^b 's, ensure that the sets $\mathbf{V}(p_j^b)$ are pairwise disjoint, and also disjoint from $\mathbf{V}(\vec{C}) \cup \mathbf{V}(\vec{S}^\circ)$;
- $(\vec{c}; \vec{t})$ is obtained from $(\vec{C}; \vec{S}^\circ)$ by replacing each a_0^b with $!(t_1^b, \dots, t_{k_b}^b)$ and each a_{i+1}^b with an enumeration of $\bigcup_{j=1}^{k_b} (f_j^b)^{-1}(S_{b, i+1}^\circ)$ – thus increasing the arity of the $?$ -connective having a_{i+1}^b as a premise, or increasing the number of trees in \vec{t} if $a_{i+1}^b \in \vec{S}^\circ$ – and then concatenating \vec{c}_j^b for $b \in \mathcal{B}_\Theta$ and $1 \leq j \leq k_b$;
- for $t \in \vec{t}$, set $f(t) = a_{i+1}^b$ if $f_j^b(t) = S_{b, i+1}^\circ$ for some j , otherwise let $f(t)$ be the only pre-tree of \vec{S}° such that t is obtained from $f(t)$ by the previous substitution.

The *Taylor expansion*⁵ of a net P is then $\mathcal{T}(P) = \{p \mid (p, f) \text{ is a pre-Taylor expansion of } P\}$.

5.3 Paths in Taylor expansion

In the following, we fix a pre-Taylor expansion (p, f) of $P^\circ = (\Theta; \vec{C}; \vec{S}^\circ)$, and we describe the structure of paths in p . Observe that if $t \in \mathbf{T}(p)$ then:

- either t is at top level, i.e. t is obtained from some $T \in \mathbf{T}(P^\circ) \setminus \mathcal{A}$ by substituting box ports with trees from resource nets, and then we say t is *outer* and write $t^* = T$;
- or t is in a copy of a box, i.e. $t \in \mathbf{T}(p_j^b)$ for some $b \in \mathcal{B}_\Theta$ and $1 \leq j \leq k_b$, and then we say t is *inner* and write $\beta(t) = b$ and $\iota(t) = (b, j)$;
- or t is a *cocontraction*, i.e. $t = !(t_1^b, \dots, t_{k_b}^b)$ for some $b \in \mathcal{B}_\Theta$, and then we write $\beta(t) = b$ and $t = !_b$.

We moreover distinguish the *boundaries*, i.e. the cocontractions of p , together with all the elements of the families \vec{s}_j^b of Definition 12: we write $!_b = a_0^b$ and $!_s = f(s)$ if $s \in \vec{s}_j^b$.

We say a subpath $\xi = t_1, \dots, t_n$ of $\chi \in \mathbf{P}(p)$ is an *inner subpath* (resp. an *outer subpath*) if each t_i is inner (resp. outer), and ξ is a *box subpath* if each t_i is inner or a cocontraction.

► **Lemma 13.** *If $\xi = t_0, \dots, t_n$ is an inner path of p then $\iota(t_i) = \iota(t_j)$ for all i and j . We then write $\beta(\xi) = b$ and $\iota(\xi) = (b, j)$.*

Proof. If $t \sim s$ and t and s are both inner then $\iota(t) = \iota(s)$. ◀

⁵ More extensive presentations of Taylor expansion of MELL nets exist in the literature, in various styles [19, 17, 6]. Our only purpose here is to introduce sufficient notations to present our analysis of the length of paths in $\mathcal{T}(P)$ by a function of the size of P .

► **Lemma 14.** *If ξ is a box path of p then ξ is an inner path or there is $b \in \mathcal{B}_\Theta$ such that $\xi = \chi_1, !_b, \chi_2$ with χ_1 and χ_2 inner subpaths. In the latter case: if $\chi_1 \neq \epsilon$ then $\beta(\chi_1) = b$; if $\chi_2 \neq \epsilon$ then $\beta(\chi_2) = b$; and $\iota(\chi_1) \neq \iota(\chi_2)$ in case both subpaths are non empty.*

Proof. If $t \sim s$ and t and s are both inner then $\iota(t) = \iota(s)$; if $t \sim !_b$ and t is inner then $\beta(t) = b$; and no other adjacency relation can hold between the elements of a box path. ◀

► **Lemma 15.** *If $\xi = t_0, \dots, t_n$ is outer then $\xi^* = t_0^*, \dots, t_n^* \in \mathbf{P}(P^\circ)$.*

Proof. If t and s are outer, then $t \sim_I^{p,I} s$ iff $t^* \sim_{I^*}^{P^\circ, I^*} s^*$, where I^* is obtained by restricting I to outer trees and then composing with $-^*$. Moreover, $-^*$ is injective. ◀

► **Lemma 16.** *Assume $\xi = \xi_0, \chi_1, \xi_1, \dots, \chi_n, \xi_n \in \mathbf{P}(p)$ where each χ_i is a box path and each ξ_i is outer. Then we can write $\chi_i = u_i, \chi'_i, v_i$ where u_i and v_i are boundaries. Moreover, $\beta(\chi_i) \neq \beta(\chi_j)$ when $i \neq j$, and we obtain $\xi^* = \xi_0^*, [u_1], [v_1], \xi_1^*, \dots, [u_n], [v_n], \xi_n^* \in \mathbf{P}(P^\circ)$.*

Proof. The proof is by induction on n . If $n = 0$, i.e. ξ is outer, then we conclude by the previous lemma. We can thus assume $n > 0$.

The endpoints of χ_i are boundaries, because χ_i is a box path and the endpoints of ξ_{i-1} and ξ_i are outer. Since each boundary is adjacent to at most one outer tree, of which it is an immediate subtree or against which it is cut, χ_i is not reduced to a single boundary. For $1 \leq i \leq n$, write $\chi_i = (u_i, \chi'_i, v_i)$.

Write $b_i = \beta(\chi_i)$. Observe that, up to $-^*$, the only new adjacency relations in ξ^* are the $[u_i] \sim_{b_i} [v_i]$ for $1 \leq i \leq n$. Hence, to conclude that ξ^* is indeed a path, it will be sufficient to prove that $b_i \neq b_j$ when $i \neq j$. If $i < j$ then, by applying the induction hypothesis, we obtain $\zeta = \xi_i^*, \dots, [u_{j-1}], [v_{j-1}], \xi_{j-1}^* \in \mathbf{P}(P^\circ)$. Then, if we had $b_i = b_j$, we would obtain a cycle $[v_i], \zeta, [u_j], [v_i]$ in P° , which is a contradiction. ◀

From Lemma 16, we can derive that p is acyclic as soon as P° is. Indeed, if ξ is a cycle in p :

- either there is a tree at top level in ξ and we can apply Lemma 16 to obtain a cycle in P° ;
- or ξ is an inner path, and we proceed inductively in $\Theta(\beta(\xi))$.

Our final result is a quantitative version of this corollary: not only there is no cycle in p but the length of paths in p is bounded by a function of P° . If $\xi = t_1, \dots, t_n$, we write $|\xi| = n$ for the *length* of ξ .

► **Theorem 17.** *If $p \in \mathcal{T}(P^\circ)$ and $\xi \in \mathbf{P}(p)$ then $|\xi| \leq 2^{\text{depth}(P^\circ)} \text{size}(P^\circ)$.*

Proof. Write $\xi = \xi_0, \chi_1, \xi_1, \dots, \chi_n, \xi_n \in \mathbf{P}(p)$ where each χ_i is a box path and each ξ_i is an outer path.

Write $b_i = \beta(\chi_i)$. By Lemma 14, χ_i is either an inner path or of the form $\zeta_i, !_{b_i}, \zeta'_i$ with ζ_i and ζ'_i inner subpaths in b_i . By induction hypothesis applied to those inner subpaths, we obtain $|\chi_i| \leq 1 + 2 \times 2^{\text{depth}(\Theta(b_i))} \text{size}(\Theta(b_i))$.

Let ξ^* be as in Lemma 16: we have $|\xi^*| = 2n + \sum_{i=0}^n |\xi_i^*| \leq \#(P^\circ)$. It follows that $\sum_{i=0}^n |\xi_i| \leq \#(P^\circ) - 2n$.

We obtain: $|\xi| = \sum_{i=0}^n |\xi_i| + \sum_{i=1}^n |\chi_i| \leq \#(P^\circ) - 2n + \sum_{i=1}^n (1 + 2^{\text{depth}(\Theta(b_i)+1)} \text{size}(\Theta(b_i)))$ hence $|\xi| \leq \#(P^\circ) + \sum_{i=1}^n 2^{\text{depth}(\Theta(b_i)+1)} \text{size}(\Theta(b_i))$ and, since $\text{depth}(\Theta(b_i)) < \text{depth}(P^\circ)$, $|\xi| \leq 2^{\text{depth}(P^\circ)} (\#(P^\circ) + \sum_{i=1}^n \text{size}(\Theta(b_i)))$. We conclude recalling that $\text{size}(P^\circ) = \#(P^\circ) + \sum_{b \in \mathcal{B}_\Theta} \text{size}(\Theta(b))$. ◀

In particular, we obtain $\text{cc}(p) \leq 2^{\text{depth}(P^\circ)} \text{size}(P^\circ)$.

5.4 Cut elimination in Taylor expansion

In resource nets, the elimination of the cut $\langle?(t_1, \dots, t_n)!(s_1, \dots, s_m)\rangle$ yields the finite sum $\sum_{\sigma:\{1, \dots, n\} \xrightarrow{\sim} \{1, \dots, m\}} \langle t_1|_{s_{\sigma(1)}} \rangle, \dots, \langle t_n|_{s_{\sigma(n)}} \rangle$. It turns out that the results of Sections 3 and 4 apply directly to resource nets: setting $\langle?(t_1, \dots, t_n)!(s_1, \dots, s_n)\rangle \rightarrow \langle t_1|_{s_{\sigma(1)}} \rangle, \dots, \langle t_n|_{s_{\sigma(n)}} \rangle$ for each permutation σ , we obtain an instance of multiplicative reduction, as the order of premises is irrelevant from a combinatorial point of view – this is all the more obvious because no typing constraint was involved in our argument. In other words, Corollary 11 also applies to the parallel reduction of resource nets. With Theorem 17, we obtain:

► **Corollary 18.** *If q is a resource net and P is an MELL net, $\{p \in \mathcal{T}(P); p \Rightarrow q\}$ is finite.*

6 Conclusion

Recall that our original motivation was the definition of a reduction relation on infinite linear combinations of resource nets, simulating cut elimination in MELL through Taylor expansion. We claim that a suitable notion is as follows:

► **Definition 19.** Write $\sum_{i \in I} a_i p_i \Rightarrow \sum_{i \in I} a_i q_i$ as soon as:

- for each $i \in I$, the resource net p_i reduces to q_i (which may be a finite sum);
- for any resource net q , there are finitely many $i \in I$ such that q is a summand of q_i .

In particular, if $\sum_{i \in I} a_i p_i$ is a Taylor expansion, then Theorem 18 ensures that the second condition of the definition of \Rightarrow is automatically valid. The details of the simulation in a quantitative setting remain to be worked out, but the main stumbling block is now over: the necessary equations on coefficients are well established, as they have been extensively studied in the various denotational models; it only remained to be able to form the associated sums directly in the syntax.

Let us mention that another important incentive to publish our results is the *normalization-by-evaluation* programme that we develop with Guerrieri, Pellissier and Tortora de Falco [1] – which is limited to strict nets for independent reasons. Indeed, if P is cut-free, the elements of the semantics of P are in one-to-one correspondence with $\mathcal{T}(P)$. Then, given a sequence P_1, \dots, P_n of MELL nets such that P_i reduces to P_{i+1} by cut elimination and P_n is normal, from $p_n \in \mathcal{T}(P_n)$ we can construct a sequence p_1, \dots, p_{n-1} of resource nets, such that each $p_i \in \mathcal{T}(P_i)$ and $p_i \Rightarrow p_{i+1}$. Then our results ensure that $\#p_1$ is bounded by a function of n , $\text{size}(P_1)$ and $\#p_n$, which is a crucial step of our construction.

We finish the paper by reviewing the restrictions that we imposed on our framework. Strictness is not an essential condition for the main results to hold. It is possible to deal with units and weakenings (0-ary \mathfrak{A} , \otimes and $?$ nodes), and then with complete Taylor expansion, including 0-ary developments of boxes (generating weakenings and coweakenings). In this case, we need to introduce additional structure – jumps from weakenings, that can be part of switching paths – and some other constraint – a bound on the number of weakenings that can jump to a given tree. The proof is naturally longer, and the bounds much greater, but the finiteness property still holds. We leave a formal treatment of this extension for further work.

The other notable constraint is the use of the *nouvelle syntaxe*, with generalized exponential links. It is also possible to deal with a standard representation, including separate derelictions and coderelictions, with a finer grained cut elimination procedure. This introduces additional complexity in the formalism but, by contrast with lifting the strictness condition, it essentially requires no new concept or technique: the difficulty in parallel reduction is to control the chains of cuts to be simultaneously eliminated, and decomposing cut elimination into finer reduction steps can only decrease the length of such chains.

References

- 1 Jules Chouquet, Giulio Guerrieri, Luc Pellissier, and Lionel Vaux. Normalization by evaluation in linear logic. In Stefano Guerrini, editor, *Preproceedings of the International Workshop on Trends in Linear Logic and Applications, TLLA*, 2017.
- 2 Vincent Danos and Thomas Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Inf. Comput.*, 209(6):966–991, 2011. doi:10.1016/j.ic.2011.02.001.
- 3 Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Arch. Math. Log.*, 28(3):181–203, 1989.
- 4 Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. PhD thesis, Université d’Aix-Marseille II, Marseille, France, 2007.
- 5 Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *CoRR*, abs/0905.4251, 2009. arXiv:0905.4251.
- 6 Daniel de Carvalho. The relational model is injective for multiplicative exponential linear logic. 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 41:1–41:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.CSL.2016.41.
- 7 Thomas Ehrhard. On köthe sequence spaces and linear logic. *Mathematical Structures in Computer Science*, 12(5):579–623, 2002. doi:10.1017/S0960129502003729.
- 8 Thomas Ehrhard. Finiteness spaces. *Mathematical Structures in Computer Science*, 15(4):615–646, 2005. doi:10.1017/S0960129504004645.
- 9 Thomas Ehrhard. A finiteness structure on resource terms. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 402–410, 2010.
- 10 Thomas Ehrhard. A new correctness criterion for MLL proof nets. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 38:1–38:10, 2014.
- 11 Thomas Ehrhard. An introduction to differential linear logic: proof-nets, models and antiderivatives. *CoRR*, abs/1606.01642, 2016.
- 12 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 309(1-3):1–41, 2003.
- 13 Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Electr. Notes Theor. Comput. Sci.*, 123:35–74, 2005.
- 14 Thomas Ehrhard and Laurent Regnier. Uniformity and the taylor expansion of ordinary lambda-terms. *Theor. Comput. Sci.*, 403(2-3):347–372, 2008.
- 15 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
- 16 Jean-Yves Girard. Normal functors, power series and lambda-calculus. *Annals of Pure and Applied Logic*, 37(2):129, 1988.
- 17 Giulio Guerrieri, Luc Pellissier, and Lorenzo Tortora de Falco. Computing connected proof(-structure)s from their taylor expansion. In *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, pages 20:1–20:18, 2016.
- 18 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, New Orleans, LA, USA, June 25-28, 2013*, pages 301–310. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.36.

- 19 Michele Pagani and Christine Tasson. The inverse taylor expansion problem in linear logic. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 222–231, 2009.
- 20 Michele Pagani, Christine Tasson, and Lionel Vaux. Strong normalizability as a finiteness structure via the taylor expansion of lambda-terms. In *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 408–423, 2016.
- 21 Laurent Regnier. *Lambda-calcul et réseaux*. PhD thesis, Université Paris 7, Paris, France, 1992.
- 22 Christine Tasson. *Sémantiques et syntaxes vectorielles de la logique linéaire*. PhD thesis, Université Paris Diderot, Paris, France, Dec 2009.
- 23 Lionel Vaux. Taylor expansion, β -reduction and normalization. In *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, pages 39:1–39:16, 2017.

Fully Abstract Models of the Probabilistic λ -calculus

Pierre Clairambault

Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, France
pierre.clairambault@ens-lyon.fr

Hugo Paquet

Department of Computer Science and Technology, University of Cambridge, UK
hugo.paquet@cl.cam.ac.uk

Abstract

We compare three models of the probabilistic λ -calculus: the *probabilistic Böhm trees* of Leventis, the *probabilistic concurrent games* of Winskel et al., and the *weighted relational model* of Ehrhard et al. Probabilistic Böhm trees and probabilistic strategies are shown to be related by a precise *correspondence theorem*, in the spirit of existing work for the pure λ -calculus. Using Leventis' theorem (probabilistic Böhm trees characterise observational equivalence), we derive a full abstraction result for the games model. Then, we relate probabilistic strategies to the weighted relational model, using an *interpretation-preserving functor* from the former to the latter. We obtain that the relational model is also fully abstract.

2012 ACM Subject Classification Theory of computation \rightarrow Denotational semantics, Theory of computation \rightarrow Probabilistic computation

Keywords and phrases Game Semantics, Lambda-calculus, Probabilistic programming, Relational model, Full abstraction

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.16

Acknowledgements This work was performed within the framework of the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program “Investissements d’Avenir” (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

1 Introduction

The interest in probabilistic programs in recent years, driven in particular by applications in machine learning and statistical modelling, has triggered the need for theoretical foundations, going beyond the pioneering work of Kozen [14] and Saheb-Djahromi [21]. Although a variety of approaches exist, we focus on the *probabilistic λ -calculus* Λ^+ , which extends the pure (untyped) λ -calculus with a probabilistic choice operator. The extension is natural and applications are quick to arise – see for instance [3]. But in order for Λ^+ to become a useful formal model for probabilistic computation, the extensive classical theory of the λ -calculus must be readapted.

Among the existing research in this direction, we are especially interested in the work of Ehrhard, Pagani and Tasson [11], and of Leventis [16, 17]. In [11], the authors define an operational semantics for Λ^+ and study a model in the category of *probabilistic coherence spaces*, an existing model [9] of Probabilistic PCF. They prove an adequacy theorem for Λ^+ , and this result also applies to the *weighted relational model*, of which probabilistic coherence spaces are a refinement.



© Pierre Clairambault and Hugo Paquet;
licensed under Creative Commons License CC-BY
27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 16; pp. 16:1–16:17



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

More recently, the PhD thesis of Leventis [16] offers a thorough exploration of the syntactical aspects of the calculus. In particular the author defines a notion of *probabilistic Böhm tree*, and redevelops the Böhm theory of the λ -calculus in a probabilistic setting. This includes Böhm’s separation theorem: probabilistic Böhm trees, in their *infinitely extensional* form, precisely characterise observational equivalence in Λ^+ .

In this paper, we propose an alternative model in the framework of concurrent games, integrating ideas from our earlier work on a concurrent games model of probabilistic PCF [5] and from Ker, Ong and Nickau’s fully abstract semantics of the pure untyped λ -calculus [13].

In [13], an exact correspondence is proved between strategies and infinitely extensional Böhm trees. Drawing inspiration from that work, we relate probabilistic strategies and probabilistic Böhm trees, but unlike [13], the correspondence is not bijective, because of the additional branching information contained in probabilistic strategies. By quotienting out this information, we derive from Leventis’ theorem a full abstraction result for the games model.

Finally, we study a functor from the probabilistic games model to the weighted relational model. This functor is a *time-forgetting* operation on strategies, in the spirit of [1]. Note that proving the functoriality of such operations is usually challenging even without probabilities, see for example Melliès’ work [19] – here, we address this by leveraging a “deadlock-free lemma” proved for concurrent strategies in [5]. We show that this functor preserves the interpretation of Λ^+ , with significant consequences: Ehrhard et al.’s adequacy result can be lifted to strategies, and the full abstraction result obtained for games via probabilistic Böhm trees can be shown to hold also for the weighted relational model, so far only known to be adequate¹.

In Section 2, we present Λ^+ and its operational semantics; we also recall Leventis’ work on probabilistic Böhm trees and define concurrent probabilistic strategies, hinting at the correspondence between the two. In Section 3, we outline the construction of a category of concurrent games and probabilistic strategies, and the reflexive object that it contains. We then study, in Section 4, the correspondence between probabilistic strategies and probabilistic Böhm trees, and prove full abstraction for the games model. Finally, in Section 5, we collapse probabilistic strategies down to weighted relations, thus showing full abstraction for the relational model.

2 The Probabilistic λ -calculus

2.1 Syntax

The set Λ^+ of terms of the probabilistic λ -calculus is defined by the following grammar, where p ranges over the interval $[0, 1]$ and x over an infinite set Var :

$$M, N ::= x \mid \lambda x.M \mid MN \mid M +_p N.$$

Write Λ_0^+ for the set of **closed terms**, i.e. those with no free variables.

The operator $+_p$ represents probabilistic choice, so that a term of the form $M +_p N$ has two possible reduction steps: to M , with probability p , and to N , with probability $1 - p$. Accordingly, the reduction relation we consider is a Markov process over the set Λ^+ , and corresponds to a probabilistic variant of the standard **head-reduction**. It is defined inductively:

¹ Independently and using a different method, Leventis and Pagani have obtained an alternative proof of full abstraction, but this work is so far unpublished.

$$\frac{}{(\lambda x.M)N \xrightarrow{1} M[N/x]} \quad \frac{}{M +_p N \xrightarrow{p} M} \quad \frac{}{M +_p N \xrightarrow{1-p} N}$$

$$\frac{M \xrightarrow{p} M'}{\lambda x.M \xrightarrow{p} \lambda x.M'} \quad \frac{M \xrightarrow{p} M' \quad M \neq \lambda x.P}{MN \xrightarrow{p} M'N}$$

For $M, N \in \Lambda^+$, there may be many reduction paths from M to N . The **weight** of a path $\pi : M \xrightarrow{p_1} \dots \xrightarrow{p_n} N$ is the product of the transition probabilities: $w(\pi) = \prod_{i=1}^n p_i$. The **probability of M reducing to N** is then defined as $\Pr(M \rightarrow N) = \sum_{\pi: M \rightarrow^* N} w(\pi)$.

The normal forms for this reduction are terms of the form $\lambda x_0 \dots x_{n-1}. y M_0 \dots M_{k-1}$, where $n, k \in \mathbb{N}$ and $M_i \in \Lambda^+$ for all i . Such terms are called **head-normal forms** (hnfs). A pure λ -term has at most one hnf called – if it exists – *its* hnf, though of course, this does not hold in the presence of probabilities.

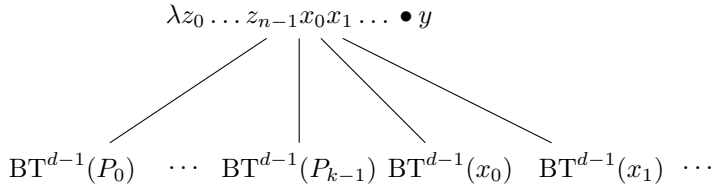
Given a set \mathcal{H} of hnfs, we set $\Pr(M \rightarrow \mathcal{H}) = \sum_{H \in \mathcal{H}} \Pr(M \rightarrow H)$. The **probability of convergence** of a term M , denoted $\Pr_{\Downarrow}(M)$, is the probability of M reducing to some hnf: $\Pr_{\Downarrow}(M) = \Pr(M \rightarrow \{H \in \Lambda^+ \mid H \text{ hnf}\})$. Finally we say that two terms M and N are **observationally equivalent**, written $M =_{\text{obs}} N$, if for all contexts $C[\]$, $\Pr_{\Downarrow}(C[M]) = \Pr_{\Downarrow}(C[N])$.

2.2 Probabilistic Böhm trees

Infinitely extensional Böhm trees for pure λ -terms

There are several notions of infinite normal forms for pure λ -terms, including *e.g.* the **Böhm trees** [2] and the **Lévy-Longo trees**, among others. The normal forms for the probabilistic λ -terms considered in this paper build on the **infinitely extensional Böhm trees** (also called **Nakajima trees**), which provide a notion of infinitely η -expanded normal form.

The infinitely extensional Böhm tree of M is in general an infinite tree, which can be defined as the limit of a sequence of finite-depth approximants. In fact those approximants will suffice for the purposes of this paper: given a λ -term M and $d \in \mathbb{N}$, the tree $\text{BT}^d(M)$ is \perp if $d = 0$ or if M has no head-normal form, and



if $d > 0$ and M has hnf $\lambda z_0 \dots z_{n-1}. y P_0 \dots P_{k-1}$.

In order to deal with issues of α -renaming, we adopt the same convention as Leventis [16], whereby the infinite sequence of abstracted variables at the root of a tree of depth $d > 0$ is labelled x_0^d, x_1^d, \dots so that any tree is determined by the pair $(y, (T_n)_{n \in \mathbb{N}})$ of its head variable and sequence of subtrees.

Leventis' probabilistic trees

Infinitely extensional Böhm trees for the λ -calculus have striking properties: they characterise observational equivalence of terms, and as a model they yield the *maximal consistent sensible* λ -theory (see [2] for details). In his PhD thesis, Leventis [16] proposes a notion of *probabilistic* Böhm tree which plays the same role for Λ^+ . Intuitively, because a term of

the form $\lambda x_0 \dots x_{n-1}.z P_0 \dots P_{k-1} +_p \lambda y_0 \dots y_{m-1}.w Q_0 \dots Q_{l-1}$ has two hnf's, it may be represented by a probability distribution over trees of the form of that above. Accordingly, two different kinds of trees are considered: **value trees**, representing head-normal forms (without probability distribution at top-level), and **probabilistic Böhm trees**, representing general terms:

► **Definition 1.** For each $d \in \mathbb{N}$, the sets \mathcal{PT}^d of **probabilistic Böhm trees of depth d** and \mathcal{VT}^d of **value trees of depth d** are defined as:

$$\begin{aligned} \mathcal{VT}^0 &= \emptyset, \\ \mathcal{VT}^{d+1} &= \left\{ (y, (T_n)_{n \in \mathbb{N}}) \mid y \in \text{Var} \text{ and } \forall n \in \mathbb{N}, T_n \in \mathcal{PT}^d \right\} \text{ and} \\ \mathcal{PT}^d &= \left\{ T : \mathcal{VT}^d \rightarrow [0, 1] \mid \sum_{t \in \mathcal{VT}^d} T(t) \leq 1 \right\}. \end{aligned}$$

We can then assign trees to individual terms:

► **Definition 2.** Given $M \in \Lambda^+$ and $d \in \mathbb{N}$, its **probabilistic Böhm tree of depth d** is the tree $\text{PT}^d(M) \in \mathcal{PT}^d$ defined as follows:

$$\begin{aligned} \text{PT}^d(M) : \mathcal{VT}^d &\longrightarrow [0, 1] \\ t &\longmapsto \Pr(M \rightarrow \{H \text{ hnf} \mid \text{VT}^d(H) = t\}) \end{aligned}$$

where for any hnf $H = \lambda z_0 \dots z_{n-1}.y P_0 \dots P_{k-1}$, the **value tree of depth d of H** is defined as

$$\text{VT}^d(H) = \left(y, \left(\text{PT}^{d-1}(P_0), \dots, \text{PT}^{d-1}(P_{k-1}), \text{PT}^{d-1}(x_n^d), \dots \right) \right).$$

Consider for example the term $M_1 = \lambda xy.x (y +_{\frac{1}{3}} (\lambda z.z))$, a head-normal form. Figure 1a outlines the first steps in the construction of its value tree of depth d , for some fixed $d \geq 2$; note that we use the symbol δ_t to denote the distribution in which t has probability 1, and all other trees 0.

Infinitely extensional probabilistic Böhm trees precisely characterise observational equivalence in Λ^+ ; writing $M =_{\text{PT}} N$ if for every $d \in \mathbb{N}$, $\text{PT}^d(M) = \text{PT}^d(N)$, we have:

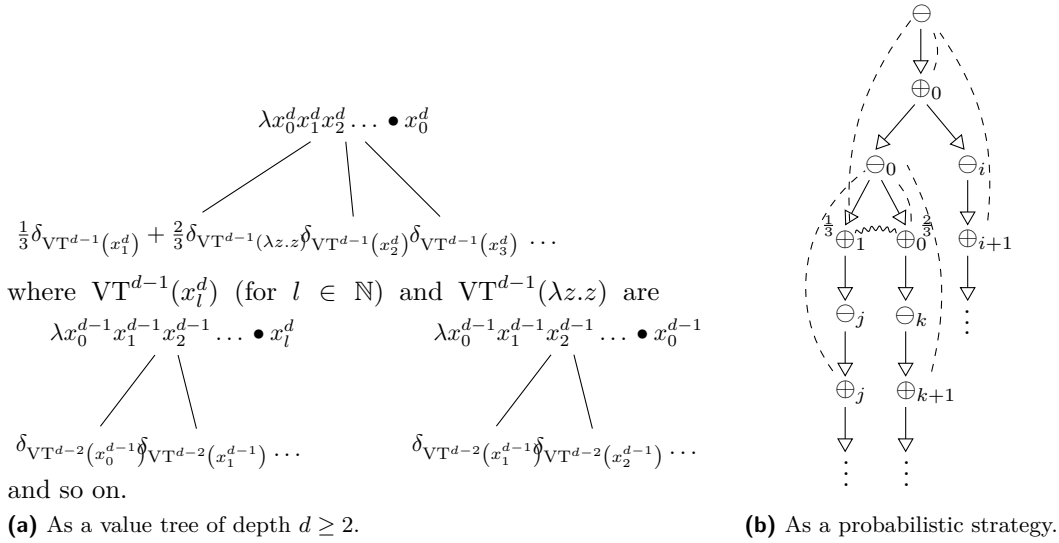
► **Theorem 3** (Leventis [16]). *For any $M, N \in \Lambda^+$, $M =_{\text{obs}} N$ if and only if $M =_{\text{PT}} N$.*

So infinitely extensional probabilistic Böhm trees provide a *fully abstract* interpretation of the probabilistic λ -calculus. We will see now that similar trees arise as *probabilistic strategies* when interpreting λ -terms in a denotational games model.

2.3 Strategies and event structures

Moving towards our game semantics of Λ^+ , we will first introduce our probabilistic strategies as a more economical, syntax-free presentation of probabilistic Böhm trees. The usual correspondence between Böhm trees and innocent strategies [12, 13] is thus naturally extended to the probabilistic and nondeterministic case.

First, we notice that the precise name given to variables in *e.g.* Figure 1a does not matter. Techniques like De Bruijn levels or indices do not apply here since we abstract infinitely many variables at each level – however, a variable occurrence is uniquely identified by a *pointer* to the node where it was abstracted, along with a *number* n , expressing that the variable



■ **Figure 1** Two interpretations of the term $M_1 = \lambda xy.x (y + \frac{1}{3} (\lambda z.z))$.

was the $(n + 1)$ -th introduced at this node. For example, the variable x_0^d is expressed with a pointer to the initial node, along with number 0. As a consequence of this representation, we can omit the abstractions: at each node, there are always countably many variables being introduced, and their name does not matter as they will be referred to differently.

Next, we split each node of the Böhm tree into two: first a node intuitively carrying the abstractions (the target of pointers – we refer to these nodes as *negative*), and one carrying the variable occurrence (the source of pointers – we refer to those as *positive*). Besides bringing us closer to games, this allows us to easily distinguish the two kinds of branching of probabilistic Böhm trees. The different arguments of a variable node form a *negative* branching: each comes with its own (implicit) distinct set of fresh variables, and a subtree (by convention, we annotate by n the negative node corresponding to the n th argument). In contrast, for a probabilistic choice such as $\frac{1}{3} \delta_{VT^{d-1}}(x_1^d) + \frac{2}{3} \delta_{VT^{d-1}}(\lambda z.z)$ in Figure 1a, the two subtrees start by defining the same variables – so instead we represent this using a *positive* branching, where we further annotate the first node of each branch with its probability.

Altogether, and ignoring the wiggly line $\sim\sim\sim$ for now, the reader may check that the diagram of Figure 1b matches the Böhm tree of Figure 1a according to these conventions (the correspondence will be made formal in Section 4). Read from top to bottom, these diagrams have an interactive flavour: they describe the actions of a player \oplus depending on those of its opponent \ominus . Our formalisation in terms of *strategies* will follow this intuition.

Probabilistic Böhm trees as probabilistic event structures

Now, we formalise the representation introduced above as a *probabilistic strategy* in the sense of [24], *i.e.* a form of probabilistic event structure. In this section we only provide this as a static representation, and leave the mechanism to *compose* strategies for Section 3. Our strategies (such as the one of Figure 1b) involve a partial order: the *dependency relation* (going from top to bottom); a relation $\sim\sim\sim$ indicating conflict and generated by probabilistic choice; and an annotation for probabilities. These are naturally formalised as *probabilistic concurrent strategies* [24] (though for the purposes of this paper we will only make use of *sequential* such strategies). We first recall the definition of event structures.

► **Definition 4.** An **event structure** [22] is a tuple (E, \leq, Con) where E is a set of **events**, \leq a partial order indicating **causal dependency**, and Con a non-empty set of **consistent** finite subsets of E , such that

$$\begin{aligned} [e] &= \{e' \mid e' \leq e\} \text{ is finite for all } e \in E \\ \{e\} &\in \text{Con for all } e \in E \\ Y \subseteq X \in \text{Con} &\implies Y \in \text{Con} \\ X \in \text{Con and } e \leq e' \in X &\implies X \cup \{e\} \in \text{Con}. \end{aligned}$$

The event structures we consider additionally have a polarity function $\text{pol} : E \rightarrow \{+, -\}$ indicating for each event whether it is a move of Player (+) or Opponent (-). We call them **event structures with polarity** (esps).

We fix some notation. Write $e \rightarrow e'$ for **immediate causality**, i.e. $e < e'$ with no events in between. Write $\mathcal{C}(E)$ for the set of finite **configurations** of E , i.e. those finite $x \subseteq E$ such that $x \in \text{Con}$ and x is down-closed: if $e \leq e' \in x$ then $e \in x$. If E has polarity, we sometimes annotate an event e to specify its polarity, as in e^+, e^- . If $x, y \in \mathcal{C}(E)$, write $x \subseteq^+ y$ (resp. $x \subseteq^- y$) if $x \subseteq y$ and every event in $y \setminus x$ has positive (resp. negative) polarity.

Ignoring probabilities and pointers, the diagram of Figure 1b is an esp: \leq is the transitive reflexive closure of \rightarrow , and consistent sets are those finite sets whose down-closure do not contain two events related by the *immediate conflict* \rightsquigarrow . We now equip esps with probabilities, which comes in the form of a $[0, 1]$ -valued function called a *valuation*.

For the forest-like event structures required to represent probabilistic λ -terms, it suffices to fix, for each Opponent event, a probability distribution on the Player events that immediately follow, as in Figure 1b. But to compose them we apply the more general machinery of [24], where valuations assign a coefficient to each *configuration* and not simply to each event. For $x \in \mathcal{C}(E)$, the coefficient $v(x)$ is the probability that the configuration x will be *reached* in an execution, provided the Opponent moves in x occur. The following definition is from [24]:

► **Definition 5.** A **probabilistic event structure with polarity** consists of an esp $(E, \leq, \text{Con}, \text{pol})$ and a **valuation**, that is, a map $v : \mathcal{C}(E) \rightarrow [0, 1]$ satisfying

- $v(\emptyset) = 1$;
- if $x \subseteq^- y$, then $v(x) = v(y)$; and
- if $y \subseteq^+ x_1, \dots, x_n$, then

$$v(y) \geq \sum_I (-1)^{|I|+1} v\left(\bigcup_{i \in I} x_i\right)$$

where I ranges over non-empty subsets of $\{1, \dots, n\}$ such that $\bigcup_{i \in I} x_i$ is a configuration.

Leaving aside pointers the diagram of Figure 1b represents a probabilistic esp, setting the valuation of a configuration x to be $\frac{1}{3}$ (resp. $\frac{2}{3}$) if it contains the event annotated with $\frac{1}{3}$ (resp. $\frac{2}{3}$), and 1 otherwise – a configuration cannot contain both labelled events.

Probabilistic strategies are certain probabilistic esps, equipped with a *labelling map* into the *game* they play on. Games are themselves esps, with the following particular shape:

► **Definition 6.** An **arena** is an esp A which is

- *forest-shaped*: if $a, b, c \in A$ with $a \leq b$ and $c \leq b$ then $a \leq c$ or $c \leq a$; and
- *alternating*: if $a \rightarrow b$ then $\text{pol}(a) \neq \text{pol}(b)$.
- *race-free*: if $x \in \mathcal{C}(A)$ has $x \subseteq^- y \in \mathcal{C}(A)$ and $x \subseteq^+ z \in \mathcal{C}(A)$, then $y \cup z \in \mathcal{C}(A)$.

Usually in game semantics, arenas represent *types*. For our untyped language, strategies representing terms all play on a *universal arena* U , introduced soon. For now though, we define a *probabilistic strategy* playing on arbitrary arena A as an esp, labelled by A .

► **Definition 7.** A **probabilistic strategy** on A consists of a probabilistic esp S , and a labelling function $\sigma : S \rightarrow A$ on events, preserving polarity, and such that:

- (1) σ *preserves configurations*: for every $x \in \mathcal{C}(S)$, $\sigma x \in \mathcal{C}(A)$;
- (2) σ *is locally injective*: if $s, s' \in x \in \mathcal{C}(S)$ and $\sigma s = \sigma s'$, then $s = s'$;
- (3) σ **is receptive**: for $x \in \mathcal{C}(S)$, if $\sigma x \subseteq^- y \in \mathcal{C}(A)$, there is a unique $x \subseteq x' \in \mathcal{C}(S)$ such that $\sigma x' = y$;
- (4) σ **is courteous**: for $s, s' \in S$, if $s \rightarrow_S s'$ and if $\text{pol}(s) = +$ or $\text{pol}(s') = -$, then $\sigma s \rightarrow_A \sigma s'$.

Conditions (1) and (2) express that σ is a **map of event structures** from S to A . Conditions (3) and (4) are there to restrict the behaviour of Player: they prevent any further constraints from being put on Opponent events than those already specified by the game.

The diagram of Figure 1b presents a probabilistic strategy $\sigma : S \rightarrow A$ – or more precisely the diagram presents S , with the pointers being representations of the immediate dependency in A of positive moves (though we do not display A for lack of space).

Winskel [24], building on previous work [20], showed how to *compose* probabilistic strategies and organise them into a category. But his games are affine, and cannot deal with the replication of resources. In recent work [5], we have extended probabilistic strategies with *symmetry*, that augments the expressivity of esps by allowing interchangeable copies of the same event. In the next section we introduce probabilistic strategies with symmetry, and give the interpretation of Λ^+ . Because of this replication of resources the interpretation of the term M_1 of Figure 1 will be an *expansion* of Figure 1b, taking into account Opponent's replications – and in general, our correspondence theorem will associate a probabilistic Böhm tree with its expansion in that sense, formulated as a probabilistic strategy.

3 Game semantics for Λ^+

In this section we construct our game semantics for Λ^+ . The category of games we use is close to our earlier concurrent games model of probabilistic PCF [5], in which we introduce a universal arena inspired from [13].

3.1 Games and strategies with symmetry

Symmetry in event structures [23] can be presented via *isomorphism families*:

► **Definition 8.** An **isomorphism family** on an event structure E is a set \tilde{E} of bijections between configurations of E , such that:

- \tilde{E} contains all identity bijections, and is closed under composition and inverse of bijections.
- For every $\theta : x \cong y \in \tilde{E}$ and $x' \in \mathcal{C}(E)$ such that $x' \subseteq x$, then $\theta|_{x'} \in \tilde{E}$.
- For every $\theta : x \cong y \in \tilde{E}$ and every extension $x \subseteq x' \in \mathcal{C}(E)$, there exists a (non-necessarily unique) $y \subseteq y' \in \mathcal{C}(E)$ and an extension $\theta \subseteq \theta'$ such that $\theta' : x' \cong y' \in \tilde{E}$.

As usual [23], it follows from these axioms that any $\theta \in \tilde{E}$ is an order-isomorphism, *i.e.* preserves and reflects the order. An **event structure with symmetry** is a pair (E, \tilde{E}) , with \tilde{E} an isomorphism family on E . If E has polarity, then we ask that every $\theta \in \tilde{E}$ preserves it, and call (E, \tilde{E}) an **event structure with symmetry and polarity** (essp).

We illustrate this definition by presenting as an essp the *universal arena* – the game that Λ^+ strategies will play on. It is an infinitely deep tree, with at every level, ω available moves, corresponding to calls from one of the players to a variable in context. There are ω ‘symmetric’ copies of each move. Formally:

► **Definition 9.** The esp $(U, \leq, \text{Con}, \text{pol})$ is defined as having:

- *events*: $U = (\mathbb{N} \times \mathbb{N})^*$, finite sequences of ordered pairs;
- *causality*: $s \leq t$ if s is a prefix of t ;
- *consistency*: no conflicts, $\text{Con} = \mathcal{P}_{\text{fin}}(U)$;
- *polarity*: $\text{pol}(s) = -$ if $|s|$ is even, $+$ if it is odd.

In a pair $(m, n) \in \mathbb{N} \times \mathbb{N}$, m represents the variable address (the subscript in Figure 1b) and n is the copy index of the move (not displayed in Figure 1b).

We now add symmetry to U , following the intuition that different copies of the same move should be interchangeable. The isomorphism family \tilde{U} is generated by an equivalence relation \sim on events, defined as the smallest equivalence relation satisfying $s \sim s' \implies s \cdot (m, n) \sim s' \cdot (m, n')$ for any $s, s' \in U$ and $m, n, n' \in \mathbb{N}$. Then, a bijection $\theta : x \cong y$ between configurations of U is in \tilde{U} whenever for all $e \in x$, $e \sim \theta(e)$.

The elements of \tilde{U} are *reindexing bijections*, which may update the copy indices of moves in a configuration. In the sequel, we will identify strategies differing only by the choice of positive copy indices, hence we need to formally identify the bijections in \tilde{U} which do not affect Opponent’s copy indices. Because of the dual nature of games we must do the same for Player; thus we define \sim^+ and \sim^- to be the smallest equivalence relations on U satisfying:

$$\begin{aligned} s \sim^p s' &\implies s \cdot (m, n) \sim^p s' \cdot (m, n) \quad (\text{for } p \in \{+, -\}) \\ s \sim^+ s' \text{ and } |s| \text{ is even} &\implies s \cdot (m, n) \sim^+ s' \cdot (m, n') \\ s \sim^- s' \text{ and } |s| \text{ is odd} &\implies s \cdot (m, n) \sim^- s' \cdot (m, n') \end{aligned}$$

for any s, s', m, n, n' . Just like \sim generates \tilde{U} , the relations \sim^+ and \sim^- generate isomorphism families \tilde{U}_+ and \tilde{U}_- , respectively.

In general, the compositional mechanism will require all arenas to come with similar data:

► **Definition 10.** A \sim -arena is a tuple $\mathcal{A} = (A, \tilde{A}, \tilde{A}_-, \tilde{A}_+)$ with A an arena, and \tilde{A} , \tilde{A}_- , and \tilde{A}_+ isomorphism families on A , such that

- \tilde{A}_- and \tilde{A}_+ are subsets of \tilde{A} ;
- if $\theta \in \tilde{A}_- \cap \tilde{A}_+$ then θ is an identity bijection;
- if $\theta \in \tilde{A}_-$ and $\theta \subseteq^- \theta' \in \tilde{A}$ then $\theta' \in \tilde{A}_-$ (where the notation \subseteq^- makes sense since bijections preserve polarity);
- if $\theta \in \tilde{A}_+$ and $\theta \subseteq^+ \theta' \in \tilde{A}$ then $\theta' \in \tilde{A}_+$.

In particular, \sim -arenas are certain *thin concurrent games*, in the terminology of [7, 8].

► **Lemma 11.** $\mathcal{U} = (U, \tilde{U}, \tilde{U}_-, \tilde{U}_+)$ is a \sim -arena.

Strategies are in turn equipped with symmetry:

► **Definition 12.** A **probabilistic essp** is an essp \mathcal{S} with a valuation $v : \mathcal{C}(\mathcal{S}) \rightarrow [0, 1]$, such that for every $\theta : x \cong y$ in $\tilde{\mathcal{S}}$, $v(x) = v(y)$. A **probabilistic \sim -strategy** on a \sim -arena \mathcal{A} consists of a probabilistic essp \mathcal{S} , and a labelling $\sigma : \mathcal{S} \rightarrow \mathcal{A}$, such that:

- (1) the underlying map $\sigma : \mathcal{S} \rightarrow \mathcal{A}$ is a strategy;
- (2) σ preserves symmetry: if $\theta : x \cong y \in \tilde{\mathcal{S}}$ then $\sigma\theta : \sigma x \cong \sigma y$ defined as $\{(\sigma s, \sigma s') \mid (s, s') \in \theta\}$, is in $\tilde{\mathcal{A}}$ (that is, it is a **map of essps** $(\mathcal{S}, \tilde{\mathcal{S}}) \rightarrow (\mathcal{A}, \tilde{\mathcal{A}})$);

(3) σ is \sim -receptive: if $\theta \in \tilde{S}$ and $\sigma\theta \subseteq^- \psi \in \tilde{A}$, there is a unique $\theta' \subseteq \theta \in \tilde{S}$ s.t. $\sigma\theta' = \psi$.
(4) \mathcal{S} is **thin**: for $\theta : x \cong y$ in \tilde{S} with $x \subseteq^+ x \cup \{s\}$, there is a *unique* $t \in S$ s.t. $\theta \cup \{(s, t)\} \in \tilde{S}$.
Finally, before we define our category of games and strategies with symmetry, let us say what it means for strategies to be the same *up to Player copy indices*:

► **Definition 13.** Probabilistic \sim -strategies $\sigma : \mathcal{S} \rightarrow \mathcal{A}$ and $\tau : \mathcal{T} \rightarrow \mathcal{A}$ are **weakly isomorphic** if there is an isomorphism of essps $\varphi : \mathcal{S} \rightarrow \mathcal{T}$, such that for any $x \in \mathcal{C}(S)$, $v_S(x) = v_T(\varphi x)$, and moreover the diagram

$$\begin{array}{ccc} \mathcal{S} & \xrightarrow{\varphi} & \mathcal{T} \\ \sigma \downarrow & \swarrow \tau & \\ \mathcal{A} & & \end{array}$$

commutes *up to positive symmetry*, in the sense that for any $x \in \mathcal{C}(S)$, the set $\{(\sigma e, \tau(\varphi e)) \mid e \in x\}$ is (the graph of) a bijection in \tilde{A}_+ .

3.2 The category PG

We now define a category with objects the \sim -arenas, and morphisms probabilistic \sim -strategies.

Let us first define some constructions on games: if \mathcal{A} is a \sim -arena, its **dual** \mathcal{A}^\perp is the \sim -arena obtained by reversing the polarity of events in \mathcal{A} , and swapping the positive and negative isomorphism families. If \mathcal{A} and \mathcal{B} are \sim -arenas, their **parallel composition** $\mathcal{A} \parallel \mathcal{B}$ is the tuple $(A \parallel B, \tilde{A} \parallel \tilde{B}, \tilde{A}_- \parallel \tilde{B}_-, \tilde{A}_+ \parallel \tilde{B}_+)$, where $A \parallel B$ is the esp with events $A + B$ (the tagged disjoint union), componentwise causal dependency and polarity, and consistent sets those of the form $X_A \parallel X_B$ for $X_A \in \text{Con}_A$ and $X_B \in \text{Con}_B$; and where the parallel composition $\tilde{A} \parallel \tilde{B}$ of isomorphism families \tilde{A} and \tilde{B} comprises bijections of the form $\theta : x_A \parallel x_B \cong y_A \parallel y_B$, defined as $\theta(1, a) = (1, \theta_A(a))$ and $\theta(2, b) = (2, \theta_B(b))$ for some $\theta_A : x_A \cong y_A$ and $\theta_B : x_B \cong y_B$ in the component iso families. Note that we will often make use of the parallel composition $\|_{i \in I} \mathcal{A}_i$ of a family of \sim -arenas; it is defined analogously.

With that in place, a **probabilistic \sim -strategy from \mathcal{A} to \mathcal{B}** is a probabilistic \sim -strategy on the \sim -arena $\mathcal{A}^\perp \parallel \mathcal{B}$. Given $\sigma : \mathcal{S} \rightarrow \mathcal{A}^\perp \parallel \mathcal{B}$ and $\tau : \mathcal{T} \rightarrow \mathcal{B}^\perp \parallel \mathcal{C}$, we can form their **interaction** as the pullback

$$\begin{array}{ccccc} & & \mathcal{T} \otimes \mathcal{S} & & \\ & \swarrow \Pi_1 & \downarrow \vee & \searrow \Pi_2 & \\ \mathcal{S} \parallel \mathcal{C} & & & & \mathcal{A} \parallel \mathcal{T} \\ & \searrow \sigma \parallel \mathcal{C} & & \swarrow \mathcal{A} \parallel \tau & \\ & & \mathcal{A} \parallel \mathcal{B} \parallel \mathcal{C} & & \end{array}$$

in the category of event structures with symmetry (and *without* polarity). The interaction is *probabilistic*: for any configuration $x \in \mathcal{C}(T \otimes S)$, we set $v_{T \otimes S}(x) = v_S((\Pi_1 x)_S) \times v_T((\Pi_2 x)_T)$, where $(\Pi_1 x)_S$ is the S -component of $\Pi_1 x \in \mathcal{C}(S \parallel C)$, and likewise for $(\Pi_2 x)_T$. The resulting map $\tau \otimes \sigma : T \otimes S \rightarrow \mathcal{A} \parallel \mathcal{B} \parallel \mathcal{C}$ is not quite a probabilistic \sim -strategy, because σ and τ play on dual versions of \mathcal{B} , making ambiguous the polarity of some events.

So as in [20, 6], the **composition** of \mathcal{S} and \mathcal{T} is obtained after *hiding* those moves of the interaction which act as *synchronisation events* – the moves $e \in T \otimes S$ such that $(\tau \otimes \sigma)e = (2, b)$ for some $b \in B$. The remaining set of events (so-called *visible*) induces an event structure $T \odot S$ with all structure inherited from $T \otimes S$, and polarity induced from $\mathcal{A}^\perp \parallel \mathcal{C}$. Any configuration $x \in \mathcal{C}(T \odot S)$ has a **unique witness** $[x] \in \mathcal{C}(T \otimes S)$. The isomorphism family $\widetilde{T \odot S}$ comprises bijections $\theta : x \cong y$ such that there is $\theta' : [x] \cong [y]$ in $\widetilde{T \otimes S}$ with $\theta \subseteq \theta'$. We get a map $\tau \odot \sigma : T \odot S \rightarrow \mathcal{A}^\perp \parallel \mathcal{C}$ which satisfies all the conditions for a probabilistic \sim -strategy, with $v_{T \odot S}(x) = v_{T \otimes S}([x])$ for every $x \in \mathcal{C}(T \odot S)$.

Copycat

As usual in game semantics, the identity morphism on a \sim -arena \mathcal{A} will be a probabilistic \sim -strategy $\mathfrak{c}_{\mathcal{A}} : \mathbb{C}_{\mathcal{A}} \rightarrow \mathcal{A}^\perp \parallel \mathcal{A}$ called **copycat**, in which Player deterministically copies the behaviour of Opponent – so any Opponent move immediately triggers the corresponding Player move in the dual game, with probability 1. Formally, $\mathbb{C}_{\mathcal{A}}$ has the same events, polarity, and consistent subsets as $\mathcal{A}^\perp \parallel \mathcal{A}$ and the extra immediate causal dependencies $\{((1, a), (2, a)) \mid a \in A, \text{pol}_{\mathcal{A}^\perp}(a) = -\}$ and $\{((2, a), (1, a)) \mid a \in A, \text{pol}_{\mathcal{A}}(a) = -\}$ (from this $\leq_{\mathbb{C}_{\mathcal{A}}}$ is obtained by transitive closure). Copycat has an isomorphism family $\mathbb{C}_{\mathcal{A}}$ which we do not define here for lack of space (it can be found e.g. in [8]). Together with the valuation $v_{\mathbb{C}_{\mathcal{A}}}(x) = 1$ for all $x \in \mathcal{C}(\mathbb{C}_{\mathcal{A}})$, this turns copycat into a probabilistic \sim -strategy.

Recall that strategies are considered up to *weak isomorphism* (Definition 13). Doing so crucially relies on the thinness axiom on strategies, which implies [8] that weak isomorphism is stable under composition, so that we may perform a quotient and retain a well-defined notion of composition. Though identity and associativity laws for strategies only hold up to isomorphism, the quotient will turn them into strict equalities. So as in [5], we have:

► **Lemma 14.** *There is a category \mathbf{PG} having*

- *objects: \sim -arenas*
- *morphisms $\mathcal{A} \mapsto \mathcal{B}$: weak isomorphism classes of probabilistic \sim -strategies on $\mathcal{A}^\perp \parallel \mathcal{B}$.*

Categorical structure

\mathbf{PG} itself is a *compact closed category*, but we are interested in the subcategory \mathbf{PG}^- , where \sim -arenas and strategies are **negative** (that is, all initial moves are negative), and strategies are moreover **well-threaded** (meaning that events in \mathcal{S} depend on a *unique* initial move).

Let \mathcal{A} and \mathcal{B} be objects of \mathbf{PG}^- . Their **tensor product** $\mathcal{A} \otimes \mathcal{B}$ is simply defined as $\mathcal{A} \parallel \mathcal{B}$. The tensorial unit is the empty \sim -arena, and moreover the tensor is *closed*: the **function space** $\mathcal{A} \multimap \mathcal{B}$ has events those of $(\parallel_{\min(B)} \mathcal{A}^\perp) \parallel \mathcal{B}$ with same polarity. The causal dependency is induced, with extra causal links $\{((2, b), (1, (b, a))) \mid b \in \min(B), a \in A\}$. The function $\chi : (\mathcal{A} \multimap \mathcal{B}) \rightarrow \mathcal{A}^\perp \parallel \mathcal{B}$ defined as $(1, (b, a)) \mapsto (1, a)$ and $(2, b) \mapsto (2, b)$ allows us to characterise consistent sets and iso families concisely: $\text{Con}_{\mathcal{A} \multimap \mathcal{B}}$ is defined as the largest set making χ a map of *esps*, and an order-isomorphism θ between configurations of $\mathcal{A} \multimap \mathcal{B}$ is in $\widetilde{\text{Con}}_{\mathcal{A} \multimap \mathcal{B}}$ iff $\chi\theta \in \widetilde{\text{Con}}_{\mathcal{A}^\perp \parallel \mathcal{B}}$. \mathbf{PG}^- also has **cartesian products**, with $\mathcal{A} \& \mathcal{B}$ defined as $\mathcal{A} \parallel \mathcal{B}$, only with consistent sets restricted to those of $\mathcal{A} \parallel \emptyset$ and $\emptyset \parallel \mathcal{B}$. The rest of the structure, including symmetry, is induced from $\mathcal{A} \parallel \mathcal{B}$ by restriction.

Finally there is a **linear exponential comonad** [18] $!$ on \mathbf{PG}^- . Given $\mathcal{A} \in \mathbf{PG}^-$, the \sim -arena $!\mathcal{A}$ is an expanded version of \mathcal{A} with countably many copies of every move. Accordingly, the *esp* $!\mathcal{A}$ is simply $\parallel_{i \in \omega} \mathcal{A}$, and the bijections in $!\mathcal{A}$ are those $\theta : \parallel_{i \in I} x_i \cong \parallel_{j \in J} y_j$ such that there exists a permutation $\pi : I \cong J$ and bijections $\theta_i \in \widetilde{\mathcal{A}}$ with $\theta((i, a)) = (\pi i, \theta_i a)$ for all $(i, a) \in \parallel_{i \in I} x_i$. Recall that \mathcal{A} is negative, so the set $!\widetilde{\mathcal{A}}_+$ of *positive* bijections (those in which only Player moves are reindexed) comprises those $\theta \in \widetilde{!\mathcal{A}}$ for which $I = J$ and $\pi : I \rightarrow J$ is the identity function, and such that each $\theta_i \in \widetilde{\mathcal{A}}_+$. On the other hand, bijections in $!\widetilde{\mathcal{A}}_-$ can consist of any $\pi : I \cong J$, so long as $\theta_i \in \widetilde{\mathcal{A}}_-$ for all i .

We leave out all further details of the categorical structure of \mathbf{PG}^- , including the various constructions on morphisms. It can be shown that \mathbf{PG}^- , together with the data above, is a model of Intuitionistic Linear Logic. From here it is standard that the Kleisli category for $!$ is a ccc:

► **Lemma 15.** *There is a cartesian closed category \mathbf{PG}_1^- having*

- *objects: negative \sim -arenas*
- *morphisms $\mathcal{A} \mapsto \mathcal{B}$: (weak isomorphism classes of) negative and well-threaded probabilistic \sim -strategies on $!\mathcal{A}^\perp \parallel \mathcal{B}$.*

With a slight abuse of notation, we shall keep using \odot for composition in the Kleisli category \mathbf{PG}_1^- . We use the following notations for the cartesian closed structure: $\mathcal{A} \Rightarrow \mathcal{B}$ is the function space $!\mathcal{A} \multimap \mathcal{B}$, cur is the bijection $\mathbf{PG}_1^-(\mathcal{A} \& \mathcal{B}, \mathcal{C}) \cong \mathbf{PG}_1^-(\mathcal{A}, \mathcal{B} \Rightarrow \mathcal{C})$, and $\text{ev}_{\mathcal{A}, \mathcal{B}} : (\mathcal{A} \Rightarrow \mathcal{B}) \& \mathcal{A} \mapsto \mathcal{B}$ is the evaluation morphism.

3.3 Interpretation of Λ^+

We finally come to our interpretation of Λ^+ terms as probabilistic strategies. We start by imposing one key new condition on strategies: *sequential innocence*. The cut-down model will be closer to the language, allowing us to prove a correspondence result in Section 4. We assume from now on that all strategies are negative and well-threaded:

► **Definition 16.** A probabilistic \sim -strategy $\sigma : \mathcal{S} \rightarrow \mathcal{A}$ is **sequential innocent** if

- a subset $X \subseteq \mathcal{S}$ is a configuration *if and only if* it is an Opponent-branching tree (that is, causality is tree-shaped and if $a \rightarrow b$ and $a \rightarrow c$ in X then $\text{pol}(a) = +$) and $\sigma X \in \mathcal{C}(\mathcal{A})$;
- for every $x, y, z \in \mathcal{C}(\mathcal{S})$ such that $x = y \cap z$ and $y \cup z \in \mathcal{C}(\mathcal{S})$, either $v(x) = 0$ or

$$\frac{v(y \cup z)}{v(x)} = \frac{v(y) v(z)}{v(x) v(x)}.$$

Less formally, innocence forces the independence (causal and probabilistic) of Opponent-forking branches of the strategy. Sequential innocent probabilistic \sim -strategies are closed under composition, stable under weak isomorphism, and copycat verifies all conditions, so we can consider the subcategory $\mathbf{PG}_1^{\text{si}}$ of \mathbf{PG}_1 consisting of those strategies. It is easy to check that $\mathbf{PG}_1^{\text{si}}$ is still a ccc; it is the category we will use to interpret Λ^+ , and in what follows we refer to $\mathbf{PG}_1^{\text{si}}$ -strategies simply as Λ^+ -strategies.

A reflexive object

Recall the \sim -arena \mathcal{U} defined in 3.1. It is a **reflexive object**, meaning that there are maps $\lambda \in \mathbf{PG}_1^{\text{si}}(\mathcal{U} \Rightarrow \mathcal{U}, \mathcal{U})$ and $\text{app} \in \mathbf{PG}_1^{\text{si}}(\mathcal{U}, \mathcal{U} \Rightarrow \mathcal{U})$ such that $\text{app} \odot \lambda = \text{id}_{\mathcal{U} \Rightarrow \mathcal{U}}$. It is easy to see that there is an isomorphism of essps $\rho : \mathcal{U} \cong \mathcal{U} \Rightarrow \mathcal{U}$. To turn this into an isomorphism in $\mathbf{PG}_1^{\text{si}}$, we can lift it to a copycat-like strategy which “plays following ρ ”. Details of this lifting are omitted but can be found in [8].

Closed terms of the probabilistic λ -calculus are interpreted as probabilistic strategies on \mathcal{U} . Open terms M with free variables in Γ are interpreted as Λ^+ -strategies $\llbracket M \rrbracket^\Gamma : \mathcal{U}^\Gamma \mapsto \mathcal{U}$, where $\mathcal{U}^\Gamma = \&_{x \in \Gamma} \mathcal{U}$. The interpretation of the λ -calculus constructions is standard, using that \mathcal{U} is a reflexive object in a ccc:

$$\begin{aligned} \llbracket x \rrbracket^\Gamma &= \pi_x, \text{ the } x^{\text{th}} \text{ projection} \\ \llbracket \lambda x.M \rrbracket^\Gamma &= \lambda \odot \text{cur}(\llbracket M \rrbracket^{\Gamma, x}) \\ \llbracket MN \rrbracket^\Gamma &= \text{ev}_{\mathcal{U}, \mathcal{U}} \odot \langle \text{app} \odot \llbracket M \rrbracket^\Gamma, \llbracket N \rrbracket^\Gamma \rangle \end{aligned}$$

In order to give an interpretation to the probabilistic choice operator, we must define the sum of two strategies. Let $\sigma : \mathcal{S} \rightarrow (\mathcal{U}^\Gamma)^\perp \parallel \mathcal{U}$ and $\tau : \mathcal{T} \rightarrow (\mathcal{U}^\Gamma)^\perp \parallel \mathcal{U}$ be Λ^+ -strategies, and let $p \in [0, 1]$. The essp $\mathcal{S} +_p \mathcal{T}$ has a unique initial Opponent move (as do \mathcal{S} and \mathcal{T} – wlog call this move ε), and continues as either \mathcal{S} or \mathcal{T} non-deterministically. That is, it has events

$\{\varepsilon\} \uplus (S \setminus \{\varepsilon\}) \uplus (T \setminus \{\varepsilon\})$, and all structure induced from \mathcal{S} and \mathcal{T} , with $X \in \text{Con}_{S+pT}$ iff $X \in \text{Con}_S$ or $X \in \text{Con}_T$. We define $v_{S+pT}(x)$ to be 1 if $x = \emptyset, \{\varepsilon\}$, $pv_S(x)$ if $x \in \mathcal{C}(S)$, and $(1-p)v_T(x)$ if $x \in \mathcal{C}(T)$. The obvious map $\sigma +_p \tau : \mathcal{S} +_p \mathcal{T} \rightarrow (\mathcal{U}^\Gamma)^\perp \parallel \mathcal{U}$ is a Λ^+ -strategy, and the interpretation of the syntactic $+_p$ is simply $\llbracket M +_p N \rrbracket^\Gamma = \llbracket M \rrbracket^\Gamma +_p \llbracket N \rrbracket^\Gamma$. We have:

► **Theorem 17 (Adequacy).** *For any $M \in \Lambda_0^+$, writing $\sigma : \mathcal{S} \rightarrow \mathcal{U}$ for $\llbracket M \rrbracket$, we have*

$$\text{Pr}_\downarrow(M) = \sum_{\substack{x \in \mathcal{C}(S) \\ |x^+|=1}} v_S(x),$$

where x^+ is the set of positive events of x .

We only state the result at this point; it will follow directly from the interpretation-preserving functor of Section 5 and the adequacy of the weighted relational model for Λ^+ . A direct corollary of Theorem 17 is the following soundness result:

► **Lemma 18 (Soundness).** *For any $M, N \in \Lambda^+$ with free variables in Γ , if $\llbracket M \rrbracket^\Gamma = \llbracket N \rrbracket^\Gamma$ then $M =_{\text{obs}} N$.*

In fact we will prove in Section 5 that the converse, *full abstraction*, also holds modulo a mild (effective) quotient. It will also follow that the weighted relational model itself is also fully abstract, which was open. These facts rely on Leventis' result [16] along with the formal correspondence between strategies and Böhm trees, to which we now move on.

4 The Correspondence Theorem

In [13], the authors prove an *exact correspondence theorem* for the pure λ -calculus: infinitely extensional Böhm trees precisely correspond to deterministic innocent strategies on a universal arena. They work in a different games framework, but the analogous phenomenon occurs in ours (the main technical difference, if we were to conduct the proof in the deterministic case, would be the explicit duplication of moves: our strategies are *expanded*, in order to accommodate Opponent's choice of copy index for every move).

For Λ^+ however, the correspondence is not so exact: although terms M and $M +_p M$ have the same probabilistic Böhm tree, they have different interpretations in $\mathbf{PG}_1^{\text{si}}$, where each probabilistic choice is recorded as an explicit branching point.² In what follows, we identify a class of *Böhm tree-like* probabilistic strategies for which the exact correspondence does hold, and we show that any strategy can be reduced to a Böhm tree-like one. Two strategies can then be considered equivalent if they reduce to the same.

First, given a Λ^+ -strategy $\sigma : \mathcal{S} \rightarrow \mathcal{U}$, define a relation \approx on the events of S as the smallest equivalence relation such that if $s_1 \approx s'_1$, $s_1 \rightarrow s_2$, $s'_1 \rightarrow s'_2$ and there is an order-isomorphism $\varphi : \{s \in S \mid s_2 \leq s\} \cong \{s' \in S \mid s'_2 \leq s'\}$ such that $\sigma s \sim^+ (\sigma \circ \varphi) s$ for all $s \geq s_2$, then $s_2 \approx s'_2$. Informally, \approx identifies events coming from the same syntactic construct in two copies of a term in an idempotent probabilistic sum, as in $M +_p M$ (where Opponent has played the same copy indices).

► **Definition 19.** We say σ is **Böhm tree-like** if it satisfies

- (1) for every $x \in \mathcal{C}(S)$, $v_S(x) > 0$; and
- (2) for every $s, s' \in S$, if $s \approx s'$ then $s = s'$.

² In particular, $\mathbf{PG}_1^{\text{si}}$ does not yield a *probabilistic λ -theory* in the sense of Leventis [16].

In other words, a Böhm tree-like strategy is one with no redundant branches. Many Λ^+ -strategies do not satisfy this property, but all can be reduced to one that does:

► **Definition 20.** Given a Λ^+ -strategy $\sigma : \mathcal{S} \rightarrow \mathcal{U}$, let \mathcal{S}_{bt} be the set of \approx -equivalence classes containing at least one event s such that $v_{\mathcal{S}}([s]) > 0$ (where $[s]$ is the down-closure of s).

It is direct to turn \mathcal{S}_{bt} into an *essp* \mathcal{S}_{bt} with structure induced by \mathcal{S} . The (partial) quotient map $f : \mathcal{S} \rightarrow \mathcal{S}_{\text{bt}}$ is then used to *push-forward* the valuation, i.e.

$$v_{\mathcal{S}_{\text{bt}}}(x) = \sum_{\substack{y \in \mathcal{C}(\mathcal{S}) \\ f y = x}} v_{\mathcal{S}}(y).$$

Then, $\sigma_{\text{bt}} : \mathcal{S}_{\text{bt}} \rightarrow \mathcal{U}$ is a Böhm tree-like Λ^+ -strategy. Write $\sigma =_{\text{bt}} \tau$ when $\sigma_{\text{bt}} = \tau_{\text{bt}}$.

We can now make formal the connection between Λ^+ -strategies and probabilistic Böhm trees. To do so we define a bijective map from the set of Böhm tree-like Λ^+ -strategies of depth d on $(\mathcal{U}^{\Gamma})^{\perp} \parallel \mathcal{U}$, to the set \mathcal{PT}_d^{Γ} of probabilistic Böhm trees of depth d with free variables in Γ . Let us say first what we mean by the *depth* of a strategy:

► **Definition 21.** The **depth** of a Λ^+ -strategy $\sigma : \mathcal{S} \rightarrow \mathcal{U}$, $\text{depth}(\sigma)$, is the maximum number of Player moves in a chain $s_0 \rightarrow \dots \rightarrow s_n$ in \mathcal{S} , and ∞ if such chains have unbounded length.

We can show by induction on d :

► **Lemma 22.** *For every $d \in \mathbb{N}$ and every $\Gamma \subseteq_{\text{fn}} \text{Var}$ there is a bijection*

$$\Psi_{\Gamma}^d : \{\sigma_{\text{bt}} \mid \sigma \in \mathbf{PG}_{\Gamma}^{\text{si}}(\mathcal{U}^{\Gamma}, \mathcal{U}) \text{ and } \text{depth } \sigma \leq d\} \xrightarrow{\cong} \mathcal{PT}_{\Gamma}^d.$$

Proof (sketch). In Section 2.3, we motivated the definition of probabilistic strategies via a geometric correspondence with probabilistic Böhm trees, to be expected in the light of standard definability results in game semantics.

However, probabilistic strategies differ from the picture of Section 2.3 due to the necessity for Player to acknowledge Opponent's replications, spawning countably many symmetric copies of branches starting with an Opponent move. It follows however from the axioms of symmetry that events differing only by Opponent's choice of copy indices have isomorphic futures. One can, with no loss of information, focus on a sub-strategy where Opponent performs no duplication, and apply the correspondence explained in Section 2.3. ◀

We now show that this bijection preserves the interpretation of Λ^+ .

► **Theorem 23 (Correspondence theorem).** *For any $M \in \Lambda^+$ and $d \in \mathbb{N}$, $\Psi_{\Gamma}^d(\llbracket M \rrbracket^d)_{\text{bt}} = \mathcal{PT}^d(M)$, where $\llbracket M \rrbracket^d$ is the maximal sub-strategy of $\llbracket M \rrbracket$ with depth $\leq d$.*

Proof (sketch). The proof is by induction on d , and follows a similar argument as in the non-probabilistic case [13], with the additional difficulty of dealing with *infinite width*: a probabilistic Böhm tree may be a probability distribution with infinite support, and the first level of Player moves in a probabilistic strategy may be infinite. One must therefore consider finite-width approximations.

Probabilistic strategies are traditionally ordered using a probabilistic version of the prefix order: given $\sigma : \mathcal{S} \rightarrow \mathcal{A}$ and $\tau : \mathcal{T} \rightarrow \mathcal{A}$ we say $\sigma \sqsubseteq \tau$ if $\mathcal{S} \subseteq \mathcal{T}$ (i.e. $\mathcal{S} \subseteq \mathcal{T}$ and all data is inherited), and for all $x \in \mathcal{C}(\mathcal{S})$, $v_{\mathcal{S}}(x) \leq v_{\mathcal{T}}(x)$. However the naive restriction of this order to the set of Böhm tree-like strategies is not sensible, because $\sigma \sqsubseteq \tau$ does not imply $\sigma_{\text{bt}} \sqsubseteq \tau_{\text{bt}}$. An alternative is given by Leventis [16, p. 111], who defines an order \preceq on the set

\mathcal{PT}_Γ^d , characterised in this setting as follows: $t \preceq t'$ iff there exists a strategy σ such that $(\Psi_\Gamma^d)^{-1}(t) =_{\text{bt}} \sigma$ and $\sigma \sqsubseteq (\Psi_\Gamma^d)^{-1}(t')$. Intuitively, the branches of σ are those of $(\Psi_\Gamma^d)^{-1}(t)$, duplicated and assigned probability in such a way that they can be extended to those of $(\Psi_\Gamma^d)^{-1}(t')$ using the prefix order \sqsubseteq .

Under \preceq the set \mathcal{PT}_Γ^d is a cpo, and we also call \preceq the corresponding order on the set of Böhm tree-like strategies (this automatically makes Ψ_Γ^d a continuous bijection).

Leventis proves the crucial property that for every term M there is a chain t_0, t_1, \dots of finite-width trees satisfying $\text{PT}^d(M) = \bigvee t_i$. Replaying his argument in our game semantics, we show that the chain $(\Psi_\Gamma^d)^{-1}(t_i), i \in \mathbb{N}$ has lub $(\llbracket M \rrbracket^d)_{\text{bt}}$. We conclude, because $(\Psi_\Gamma^d)^{-1}(\text{PT}_\Gamma^d(M)) = (\Psi_\Gamma^d)^{-1}(\bigvee_{i \in \mathbb{N}} t_i) = \bigvee_{i \in \mathbb{N}} (\Psi_\Gamma^d)^{-1}(t_i) = (\llbracket M \rrbracket^d)_{\text{bt}}$. \blacktriangleleft

Using the correspondence it follows easily that:

► **Lemma 24.** *For any $M, N \in \Lambda^+$, $M =_{PT} N$ if and only if $\llbracket M \rrbracket =_{\text{bt}} \llbracket N \rrbracket$.*

► **Theorem 25** (Full abstraction). *The model $\mathbf{PG}_\Gamma^{\text{si}} / =_{\text{bt}}$ is fully abstract, i.e. $M =_{\text{obs}} N$ if and only if $\llbracket M \rrbracket =_{\text{bt}} \llbracket N \rrbracket$.*

5 Weighted Relational Semantics

In this final section, we consider the weighted relational model of Λ^+ . It lives in the category $\mathbf{PRel}_!$ whose objects are sets and whose morphisms are certain matrices with coefficients in the set $\overline{\mathbb{R}}_+ = \mathbb{R}_+ \cup \{\infty\}$. This interpretation of probabilistic λ -terms was first suggested in [11], where authors consider the category $\mathbf{PCoh}_!$ of **probabilistic coherence spaces**, a refinement (using *biorthogonality*) of the model $\mathbf{PRel}_!$ presented here. $\mathbf{PCoh}_!$ has desirable properties (notably, all coefficients are finite) but because there is a faithful functor $\mathbf{PCoh}_! \rightarrow \mathbf{PRel}_!$ preserving the interpretation of Λ^+ , all the results of [11] hold for the simpler model $\mathbf{PRel}_!$, which we focus on in this paper and proceed to define.

5.1 The weighted relational model of Λ^+

We use the notation $\mathbf{PRel}_!$ to indicate that the model is obtained as the Kleisli category for a comonad $!$, much like $\mathbf{PG}_!$. The underlying category \mathbf{PRel} is a well-known model of intuitionistic linear logic (see e.g. [15]), but we skip its construction and give a direct presentation of $\mathbf{PRel}_!$:

► **Definition 26.** The category $\mathbf{PRel}_!$ is defined as follows:

- *objects:* sets;
- *morphisms from X to Y :* maps $\varphi : \mathcal{M}_f(X) \times Y \rightarrow \overline{\mathbb{R}}_+$, where $\mathcal{M}_f(X)$ is the set of **finite multisets** of elements of X ;
- *composition:* for $\varphi \in \mathbf{PRel}_!(X, Y)$, $\psi \in \mathbf{PRel}_!(Y, Z)$, define $\psi \circ \varphi : \mathcal{M}_f(X) \times Z \rightarrow \overline{\mathbb{R}}_+$ as

$$(\psi \circ \varphi)(m, c) = \sum_{p \in \mathcal{M}_f(Y)} \psi_{p,c} \sum_{\substack{(m_b)_{b \in p} \\ \text{s.t. } m = \uplus m_b}} \prod_{b \in p} \varphi_{(m_b, b)}$$

for every $m \in \mathcal{M}_f(X)$ and $c \in Z$.

- *identity:* for any set X , and for any $m \in \mathcal{M}_f(X)$ and $a \in X$, define

$$\text{id}_X(m, a) = \begin{cases} 1 & \text{if } m = [a] \\ 0 & \text{otherwise.} \end{cases}$$

$\mathbf{PRel}_!$ is cartesian closed, with $X \& Y = X \uplus Y$ and $X \Rightarrow Y = \mathcal{M}_f(X) \times Y$. There is a reflexive object \mathcal{D} in $\mathbf{PRel}_!$, supporting the interpretation of Λ^+ , and defined as the least fixed point of the operation mapping X to the set $\mathcal{M}_f(X)^{(\omega)}$ of **quasi-finite** sequences of finite multisets over X , i.e. with all but finitely many elements equal to $[\]$. Concretely, \mathcal{D} is the lub of the chain D_0, D_1, \dots where $D_0 = \emptyset$ and $D_{i+1} = \mathcal{M}_f(D_i)^{(\omega)}$ for all i . It is the case that $\mathcal{D} \cong \mathcal{D} \Rightarrow \mathcal{D}$; the set-theoretical bijection and its lifting to a $\mathbf{PRel}_!$ isomorphism can be found in [11].

Terms of Λ^+ are interpreted in the standard way, with $\llbracket M +_p N \rrbracket_{\mathbf{PRel}_!}^\Gamma(d) = p \llbracket M \rrbracket_{\mathbf{PRel}_!}^\Gamma(d) + (1-p) \llbracket N \rrbracket_{\mathbf{PRel}_!}^\Gamma(d)$ for every $d \in \mathcal{D}$. We have:

► **Theorem 27** (Adequacy [11]). *For any $M \in \Lambda_0^+$, the map $\llbracket M \rrbracket_{\mathbf{PRel}_!} : \mathcal{D} \rightarrow \overline{\mathbb{R}}_+$ satisfies*

$$\text{Pr}_\downarrow(M) = \sum_{d \in \mathcal{D}_2} \llbracket M \rrbracket_{\mathbf{PRel}_!}(d).$$

5.2 Relational collapse

We now connect the two models *via* a functor $\downarrow : \mathbf{PG}_!^{\text{si}} \rightarrow \mathbf{PRel}_!$, which intuitively *forgets* the causal information in a strategy, only remembering the states reached during the execution.

If (E, \tilde{E}) is an event structure with symmetry, write \cong for the equivalence relation on $\mathcal{C}(E)$ defined as $x \cong y$ if and only if there is $\theta : x \cong y$ in \tilde{E} . For \mathcal{A} an arbitrary negative \sim -arena, the set $\downarrow \mathcal{A}$ is then defined as the quotient $\{x \in \mathcal{C}(\mathcal{A}) \mid x \text{ non-empty}\} / \cong$.

For any \mathcal{A}, \mathcal{B} , there is a bijection $\downarrow(\mathcal{A} \Rightarrow \mathcal{B}) \simeq \mathcal{M}_f(\downarrow \mathcal{A}) \times \downarrow \mathcal{B}$, enabling morphisms of $\mathbf{PG}_!^{\text{si}}$ to be mapped to those of $\mathbf{PRel}_!$: if $\sigma : \mathcal{S} \rightarrow !\mathcal{A} \Rightarrow \mathcal{B}$ is a Λ^+ -strategy and $\mathbf{x} \in \downarrow(\mathcal{A} \Rightarrow \mathcal{B})$ (so \mathbf{x} is an equivalence class of configurations), the set of **witnesses** of \mathbf{x} is defined as $\text{wit}_S(\mathbf{x}) = \{z \in \mathcal{C}(S) \mid \sigma z \in \mathbf{x} \text{ and the maximal moves of } z \text{ have polarity } +\} / \cong$. Because v_S is invariant under symmetry, we can transport σ to $\downarrow \sigma : \downarrow(\mathcal{A} \Rightarrow \mathcal{B}) \rightarrow \overline{\mathbb{R}}_+$ *via*

$$\downarrow \sigma(\mathbf{x}) = \sum_{\mathbf{z} \in \text{wit}_S(\mathbf{x})} v_S(\mathbf{z})$$

for each $\mathbf{x} \in \downarrow(\mathcal{A} \Rightarrow \mathcal{B})$. One can then easily deduce from the *deadlock-free lemma* of [5]:

► **Lemma 28.** *\downarrow is a functor $\mathbf{PG}_!^{\text{si}} \rightarrow \mathbf{PRel}_!$.*

Furthermore, \downarrow preserves the interpretation of Λ^+ terms and is well-defined on the quotiented model $\mathbf{PG}_!^{\text{si}} / =_{\text{bt}}$:

► **Lemma 29.** *$\downarrow \mathcal{U} \cong \mathcal{D}$ and up to this iso, for any $M \in \Lambda^+$ we have $\downarrow \llbracket M \rrbracket_{\mathbf{PG}_!^{\text{si}}} = \llbracket M \rrbracket_{\mathbf{PRel}_!}$.*

► **Lemma 30.** *If $\sigma =_{\text{bt}} \tau$ then $\downarrow \sigma = \downarrow \tau$.*

Combining the previous two lemmas and the soundness theorem, we finally get:

► **Theorem 31** (Full abstraction). *For any $M, N \in \Lambda^+$ with free variables in Γ , $M =_{\text{obs}} N$ if and only if $\llbracket M \rrbracket_{\mathbf{PRel}_!} = \llbracket N \rrbracket_{\mathbf{PRel}_!}$.*

6 Conclusion

An immediate corollary of Theorem 31 is that the probabilistic coherence space model of [11] is fully abstract, since $\mathbf{PCoh}_!$ and $\mathbf{PRel}_!$ induce the same equational theory on Λ^+ terms.

Interestingly, the results of this paper should further entail that the interpretation of Λ^+ in the simpler model of Danos and Harmer [10] is also fully abstract, since one can in

principle map our strategies functorially to theirs. Note however that since it is not known how to state a notion of probabilistic innocence in Danos and Harmer’s model, definability fails and the present work could not have been carried out there.

So using probabilistic concurrent games, we obtain probabilistic analogues of well-established results from the theory of the pure λ -calculus: the correspondence between Böhm trees and innocent strategies [13], and the full abstraction property of the relational model [4].

References

- 1 Patrick Baillot, Vincent Danos, Thomas Ehrhard, and Laurent Regnier. Timeless games. In *International Workshop on Computer Science Logic*, pages 56–77. Springer, 1997.
- 2 Hendrik Pieter Barendregt. *The lambda calculus*, volume 2. North-Holland Amsterdam, 1984.
- 3 Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. A Lambda-Calculus Foundation for Universal Probabilistic Programming. *CoRR*, abs/1512.08990, 2015. URL: <http://arxiv.org/abs/1512.08990>.
- 4 Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. Not enough points is enough. In *International Workshop on Computer Science Logic*, pages 298–312. Springer, 2007.
- 5 Simon Castellan, Pierre Clairambault, Hugo Paquet, and Glynn Winskel. The Concurrent Game Semantics of Probabilistic PCF. In *Logic in Computer Science (LICS), 2018 33rd Annual ACM/IEEE Symposium on*, page to appear. ACM/IEEE, 2018.
- 6 Simon Castellan, Pierre Clairambault, and Glynn Winskel. Symmetry in Concurrent Games. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 28. ACM, 2014.
- 7 Simon Castellan, Pierre Clairambault, and Glynn Winskel. The Parallel Intensionally Fully Abstract Games Model of PCF. In *Logic in Computer Science (LICS), 2015 30th Annual ACM/IEEE Symposium on*, pages 232–243. IEEE, 2015.
- 8 Simon Castellan, Pierre Clairambault, and Glynn Winskel. Concurrent hyland-ong games. *arXiv*, 2016. [arXiv:1409.7542](https://arxiv.org/abs/1409.7542).
- 9 Vincent Danos and Thomas Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Information and Computation*, 209(6):966–991, 2011.
- 10 Vincent Danos and Russell S Harmer. Probabilistic game semantics. *ACM Transactions on Computational Logic (TOCL)*, 3(3):359–382, 2002.
- 11 Thomas Ehrhard, Michele Pagani, and Christine Tasson. The computational meaning of probabilistic coherence spaces. In *Logic in Computer Science (LICS), 2011 26th Annual IEEE Symposium on*, pages 87–96. IEEE, 2011.
- 12 J Martin E Hyland and C-HL Ong. On full abstraction for PCF: I, II, and III. *Information and computation*, 163(2):285–408, 2000.
- 13 Andrew D Ker, Hanno Nickau, and C-H Luke Ong. Innocent game models of untyped λ -calculus. *Theoretical Computer Science*, 272(1-2):247–292, 2002.
- 14 Dexter Kozen. Semantics of probabilistic programs. In *Foundations of Computer Science, 1979., 20th Annual Symposium on*, pages 101–114. IEEE, 1979.
- 15 Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. Weighted relational models of typed lambda-calculi. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 301–310. IEEE Computer Society, 2013.
- 16 Thomas Leventis. *Probabilistic lambda-theories*. PhD thesis, Aix-Marseille Université, 2016.
- 17 Thomas Leventis. Probabilistic Böhm Trees and Probabilistic Separation. In *Logic in Computer Science (LICS), 2018 33rd Annual ACM/IEEE Symposium on*. ACM/IEEE, 2018.

- 18 Paul-André Mellies. Categorical semantics of linear logic. *Panoramas et synthèses*, 27:15–215, 2009.
- 19 Paul-André Mellies and Samuel Mimram. Asynchronous games: innocence without alternation. In *International Conference on Concurrency Theory*, pages 395–411. Springer, 2007.
- 20 Silvain Rideau and Glynn Winskel. Concurrent strategies. In *Logic in Computer Science (LICS), 2011 26th Annual IEEE Symposium on*, pages 409–418. IEEE, 2011.
- 21 Nasser Saheb-Djahromi. Cpo’s of measures for nondeterminism. *Theoretical Computer Science*, 12(1):19–37, 1980.
- 22 Glynn Winskel. *Events in Computation*. PhD thesis, University of Edinburgh, 1980.
- 23 Glynn Winskel. Event structures with symmetry. *Electronic Notes in Theoretical Computer Science*, 172:611–652, 2007.
- 24 Glynn Winskel. Distributed probabilistic and quantum strategies. *Electronic Notes in Theoretical Computer Science*, 298:403–425, 2013.


Uniform Inductive Reasoning in Transitive Closure Logic via Infinite Descent

Liron Cohen¹

Dept. of Computer Science, Cornell University, NY, USA
lironcohen@cornell.edu

Reuben N. S. Rowe²

School of Computing, University of Kent, Canterbury, UK
r.n.s.rowe@kent.ac.uk

 <https://orcid.org/0000-0002-4271-9078>

Abstract

Transitive closure logic is a known extension of first-order logic obtained by introducing a transitive closure operator. While other extensions of first-order logic with inductive definitions are a priori parametrized by a set of inductive definitions, the addition of the transitive closure operator uniformly captures all finitary inductive definitions. In this paper we present an *infinitary* proof system for transitive closure logic which is an *infinite descent*-style counterpart to the existing (explicit induction) proof system for the logic. We show that, as for similar systems for first-order logic with inductive definitions, our infinitary system is complete for the standard semantics and subsumes the explicit system. Moreover, the uniformity of the transitive closure operator allows semantically meaningful complete restrictions to be defined using simple syntactic criteria. Consequently, the restriction to regular infinitary (i.e. *cyclic*) proofs provides the basis for an effective system for automating inductive reasoning.

2012 ACM Subject Classification Theory of computation → Proof theory, Theory of computation → Automated reasoning

Keywords and phrases Induction, Transitive Closure, Infinitary Proof Systems, Cyclic Proof Systems, Soundness, Completeness, Standard Semantics, Henkin Semantics

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.17

Related Version An extended version of the paper is available at [12], <https://arxiv.org/abs/1802.00756>.

1 Introduction

A core technique in mathematical reasoning is that of *induction*. This is especially true in computer science, where it plays a central role in reasoning about recursive data and computations. Formal systems for mathematical reasoning usually capture the notion of inductive reasoning via one or more inference rules that express the general induction schemes, or principles, that hold for the elements being reasoned over.

Increasingly, we are concerned with not only being able to formalise as much mathematical reasoning as possible, but also with doing so in an effective way. In other words, we seek to be able to automate such reasoning. *Transitive closure* (TC) logic has been identified as a

¹ Supported by Fulbright Post-doctoral Scholar program, Weizmann Institute of Science National Postdoctoral Award program for Advancing Women in Science, and Eric and Wendy Schmidt Postdoctoral Award program for Women in Mathematical and Computing Sciences.

² Supported by EPSRC Grant No. EP/N028759/1.



potential candidate for a minimal, “most general” system for inductive reasoning, which is also very suitable for automation [1, 10, 11]. TC adds to first-order logic a single operator for forming binary relations: specifically, the transitive closures of arbitrary formulas (more precisely, the transitive closure of the binary relation induced by a formula with respect to two distinct variables). In this work, for simplicity, we use a reflexive form of the operator; however the two forms are equivalent in the presence of equality. This modest addition affords enormous expressive power: namely it provides a uniform way of capturing inductive principles. If an induction scheme is expressed by a formula φ , then the elements of the inductive collection it defines are those “reachable” from the base elements x via the iteration of the induction scheme. That is, those y ’s for which (x, y) is in the transitive closure of φ . Thus, bespoke induction principles do not need to be added to, or embedded within, the logic; instead, all induction schemes are available within a single, unified language. In this respect, the transitive closure operator resembles the W-type [22], which also provides a single type constructor from which one can uniformly define a variety of inductive types.

TC logic is intermediate between first- and second-order logic. Furthermore, since the TC operator is a particular instance of a least fixed point operator, TC logic is also subsumed by fixed-point logics such as the μ -calculus [19]. However, despite its minimality TC logic retains enough expressivity to capture inductive reasoning, as well as to subsume arithmetics (see Section 4.2.1). Moreover, from a proof theoretical perspective the conciseness of the logic makes it of particular interest. The use of only one constructor of course comes with a price: namely, formalizations (mostly of non-linear induction schemes) may be somewhat complex. However, they generally do not require as complex an encoding as in arithmetics, since the TC operator can be applied on any formula and thus (depending on the underlying signature) more naturally encode induction on sets more complex than the natural numbers.

Since its expressiveness entails that TC logic subsumes arithmetics, by Gödel’s result, any effective proof system for it must necessarily be incomplete for the standard semantics. Notwithstanding, a natural, effective proof system which is sound for TC logic was shown to be complete with respect to a generalized form of Henkin semantics [9]. In this paper, following similar developments in other formalizations for fixed point logics and inductive reasoning (see e.g. [4, 5, 6, 24, 27]), we present an *infinitary* proof theory for TC logic which, as far as we know, is the first system that is (cut-free) complete with respect to the standard semantics. More specifically, our system employs infinite-height, rather than infinite-width proofs (see Section 3.2). The soundness of such infinitary proof theories is underpinned by the principle of *infinite descent*: proofs are permitted to be infinite, non-well-founded trees, but subject to the restriction that every infinite path in the proof admits some infinite descent. The descent is witnessed by tracing terms or formulas for which we can give a correspondence with elements of a well-founded set. In particular, we can trace terms that denote elements of an inductively defined (well-founded) set. For this reason, such theories are considered systems of implicit induction, as opposed to those which employ explicit rules for applying induction principles. While a full infinitary proof theory is clearly not effective, in the aforementioned sense, such a system can be obtained by restricting consideration to only the *regular* infinite proofs. These are precisely those proofs that can be finitely represented as (possibly cyclic) graphs.

These infinitary proof theories generally subsume systems of explicit induction in expressive power, but also offer a number of advantages. Most notably, they can ameliorate the primary challenge for inductive reasoning: finding an induction *invariant*. In explicit induction systems, this must be provided *a priori*, and is often much stronger than the goal one is ultimately interested in proving. However, in implicit systems the inductive arguments and

hypotheses may be encoded in the cycles of a proof, so cyclic proof systems seem better for automation. The cyclic approach has also been used to provide an optimal cut-free complete proof system for Kleene algebra [15], providing further evidence of its utility for automation.

In the setting of TC logic, we observe some further benefits over more traditional formal systems of inductive definitions and their infinitary proof theories (cf. LKID [6, 21]). TC (with a pairing function) has all first-order definable finitary inductive definitions immediately “available” within the language of the logic: as with inductive hypotheses, one does not need to “know” in advance which induction schemes will be required. Moreover, the use of a single transitive closure operator provides a uniform treatment of all induction schemes. That is, instead of having a proof system parameterized by a set of inductive predicates and rules for them (as is the case in LKID), TC offers a single proof system with a single rule scheme for induction. This has immediate advantages for developing the metatheory: the proofs of completeness for standard semantics and adequacy (i.e. subsumption of explicit induction) for the infinitary system presented in this paper are simpler and more straightforward. Moreover, it permits a cyclic subsystem, which also subsumes explicit induction, to be defined via a simple syntactic criterion that we call *normality*. The smaller search space of possible proofs further enhances the potential for automation. TC logic seems more expressive in other ways, too. For instance, the transitive closure operator may be applied to arbitrarily complex formulas, not only to collections of atomic formulas (cf. Horn clauses), as in e.g. [4, 6].

We show that the explicit and cyclic TC systems are equivalent under arithmetic, as is the case for LKID [3, 26]. However, there are cases in which the cyclic system for LKID is strictly more expressive than the explicit induction system [2]. To obtain a similar result for TC, the fact that all induction schemes are available poses a serious challenge. For one, the counter-example used in [2] does not serve to show this result holds for TC. If this strong inequivalence indeed holds also for TC, it must be witnessed by a more subtle and complex counter-example. Conversely, it may be that the explicit and cyclic systems do coincide for TC. In either case, this points towards fundamental aspects that require further investigation.

The rest of the paper is organised as follows. In Section 2 we reprise the definition of transitive closure logic and both its standard and Henkin-style semantics. Section 3 presents the existing explicit induction proof system for TC logic, and also our new infinitary proof system. We prove the latter sound and complete for the standard semantics, and also derive cut-admissibility. In Section 4 we compare the expressive power of the infinitary system (and its cyclic subsystem) with the explicit system. Section 5 concludes and examines the remaining open questions for our system as well as future work. Due to lack of space, proofs are omitted but can be found in an extended version [12].

2 Transitive Closure Logic and its Semantics

In this section we review the language of transitive closure logic, and two possible semantics for it: a standard one, and a Henkin-style one. For simplicity of presentation we assume (as is standard practice) a designated equality symbol in the language. We denote by $v[x_1 := a_n, \dots, x_n := a_n]$ the variant of the assignment v which assigns a_i to x_i for each i , and by $\varphi \left\{ \frac{t_1}{x_1}, \dots, \frac{t_n}{x_n} \right\}$ the result of simultaneously substituting each t_i for the free occurrences of x_i in φ .

► **Definition 1** (The language \mathcal{L}_{RTC}). Let σ be a first-order signature with equality, whose terms are ranged over by s and t and predicates by P , and let x, y, z , etc. range over a countable set of variables. The language \mathcal{L}_{RTC} consists of the formulas defined by the grammar:

$$\varphi, \psi ::= s = t \mid P(t_1, \dots, t_n) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \forall x.\varphi \mid \exists x.\varphi \mid (\text{RTC}_{x,y} \varphi)(s, t)$$

As usual, $\forall x$ and $\exists x$ bind free occurrences of the variable x and we identify formulas up to renaming of bound variables, so that capturing of free variables during substitution does not occur. Note that in the formula $(RTC_{x,y} \varphi)(s, t)$ free occurrences of x and y in φ are also bound (but not those in s and t).

► **Definition 2** (Standard Semantics). Let $M = \langle D, I \rangle$ be a first-order structure (i.e. D is a non-empty domain and I an interpretation function), and v an assignment in M which we extend to terms in the obvious way. The satisfaction relation \models between model-valuation pairs $\langle M, v \rangle$ and formulas is defined inductively on the structure of formulas by:

- $M, v \models s = t$ if $v(s) = v(t)$;
- $M, v \models P(t_1, \dots, t_n)$ if $(v(t_1), \dots, v(t_n)) \in I(P)$;
- $M, v \models \neg \varphi$ if $M, v \not\models \varphi$;
- $M, v \models \varphi_1 \wedge \varphi_2$ if both $M, v \models \varphi_1$ and $M, v \models \varphi_2$;
- $M, v \models \varphi_1 \vee \varphi_2$ if either $M, v \models \varphi_1$ or $M, v \models \varphi_2$;
- $M, v \models \varphi_1 \rightarrow \varphi_2$ if $M, v \models \varphi_1$ implies $M, v \models \varphi_2$;
- $M, v \models \exists x. \varphi$ and $M, v \models \forall x. \varphi$ if $M, v[x := a] \models \varphi$ for some (respectively all) $a \in D$;
- $M, v \models (RTC_{x,y} \varphi)(s, t)$ if $v(s) = v(t)$, or there exist $a_0, \dots, a_n \in D$ ($n > 0$) s.t. $v(s) = a_0$, $v(t) = a_n$, and $M, v[x := a_i, y := a_{i+1}] \models \varphi$ for $0 \leq i < n$.

We say that a formula φ is valid with respect to the standard semantics when $M, v \models \varphi$ holds for all models M and valuations v .

We next recall the concepts of frames and Henkin structures (see, e.g., [18]). A frame is a first-order structure together with some subset of the powerset of its domain (called its set of admissible subsets).

► **Definition 3** (Frames). A frame M is a triple $\langle D, I, \mathcal{D} \rangle$, where $\langle D, I \rangle$ is a first-order structure, and $\mathcal{D} \subseteq \wp(D)$.

Note that if $\mathcal{D} = \wp(D)$, the frame is identified with a standard first-order structure.

► **Definition 4** (Frame Semantics). \mathcal{L}_{RTC} formulas are interpreted in frames as in Definition 2 above, except for:

- $M, v \models (RTC_{x,y} \varphi)(s, t)$ if for every $A \in \mathcal{D}$, if $v(s) \in A$ and for every $a, b \in D$: $a \in A$ and $M, v[x := a, y := b] \models \varphi$ implies $b \in A$, then $v(t) \in A$.

We now consider Henkin structures, which are frames whose set of admissible subsets is closed under parametric definability.

► **Definition 5** (Henkin structures). A Henkin structure is a frame $M = \langle D, I, \mathcal{D} \rangle$ such that $\{a \in D \mid M, v[x := a] \models \varphi\} \in \mathcal{D}$ for every φ , and v in M .

We refer to the semantics induced by quantifying over the (larger) class of Henkin structures as the Henkin semantics.

It is worth noting that the inclusion of equality in the basic language is merely for notational convenience. This is because the RTC operator allows us, under both the standard and Henkin semantics, to actually *define* equality $s = t$ on terms as $(RTC_{x,y} \perp)(s, t)$.

3 Proof Systems for \mathcal{L}_{RTC}

In this section, we define two proof systems for \mathcal{L}_{RTC} . The first is a finitary proof system with an explicit induction rule for RTC formulas. The second is an infinitary proof system, in which RTC formulas are simply unfolded, and inductive arguments are represented via

$$\begin{array}{c}
\text{(Axiom): } \frac{}{\varphi \Rightarrow \varphi} \quad \text{(WL): } \frac{\Gamma \Rightarrow \Delta}{\Gamma, \varphi \Rightarrow \Delta} \quad \text{(WR): } \frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \varphi} \\
\\
\text{(=L}_1\text{): } \frac{\Gamma \Rightarrow \varphi \left\{ \frac{s}{x} \right\}, \Delta}{\Gamma, s = t \Rightarrow \varphi \left\{ \frac{t}{x} \right\}, \Delta} \quad \text{(=L}_2\text{): } \frac{\Gamma \Rightarrow \varphi \left\{ \frac{t}{x} \right\}, \Delta}{\Gamma, s = t \Rightarrow \varphi \left\{ \frac{s}{x} \right\}, \Delta} \quad \text{(=R): } \frac{}{\Rightarrow t = t} \\
\\
\text{(}\forall\text{L): } \frac{\Gamma \Rightarrow \varphi, \Delta \quad \Gamma, \psi \Rightarrow \Delta}{\Gamma, \varphi \vee \psi \Rightarrow \Delta} \quad \text{(}\wedge\text{L): } \frac{\Gamma, \varphi, \psi \Rightarrow \Delta}{\Gamma, \varphi \wedge \psi \Rightarrow \Delta} \quad \text{(}\rightarrow\text{L): } \frac{\Gamma \Rightarrow \varphi, \Delta \quad \Gamma, \psi \Rightarrow \Delta}{\Gamma, \varphi \rightarrow \psi \Rightarrow \Delta} \quad \text{(}\neg\text{L): } \frac{\Gamma \Rightarrow \varphi, \Delta}{\Gamma, \neg \varphi \Rightarrow \Delta} \\
\\
\text{(}\forall\text{R): } \frac{\Gamma \Rightarrow \varphi, \psi, \Delta}{\Gamma \Rightarrow \varphi \vee \psi, \Delta} \quad \text{(}\wedge\text{R): } \frac{\Gamma \Rightarrow \varphi, \Delta \quad \Gamma \Rightarrow \psi, \Delta}{\Gamma \Rightarrow \varphi \wedge \psi, \Delta} \quad \text{(}\rightarrow\text{R): } \frac{\Gamma, \varphi \Rightarrow \psi, \Delta}{\Gamma \Rightarrow \varphi \rightarrow \psi, \Delta} \quad \text{(}\neg\text{R): } \frac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma \Rightarrow \neg \varphi, \Delta} \\
\\
\text{(}\exists\text{L): } \frac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma, \exists x. \varphi \Rightarrow \Delta} \quad x \notin \text{fv}(\Gamma, \Delta) \quad \text{(}\forall\text{L): } \frac{\Gamma, \varphi \left\{ \frac{t}{x} \right\} \Rightarrow \Delta}{\Gamma, \forall x. \varphi \Rightarrow \Delta} \quad \text{(Cut): } \frac{\Gamma \Rightarrow \varphi, \Delta \quad \Sigma, \varphi \Rightarrow \Pi}{\Gamma, \Sigma \Rightarrow \Delta, \Pi} \\
\\
\text{(}\exists\text{R): } \frac{\Gamma \Rightarrow \varphi \left\{ \frac{t}{x} \right\}, \Delta}{\Gamma \Rightarrow \exists x. \varphi, \Delta} \quad \text{(}\forall\text{R): } \frac{\Gamma \Rightarrow \varphi, \Delta}{\Gamma \Rightarrow \forall x. \varphi, \Delta} \quad x \notin \text{fv}(\Gamma, \Delta) \quad \text{(Subst): } \frac{\Gamma \Rightarrow \Delta}{\Gamma \left\{ \frac{t_1}{x_1}, \dots, \frac{t_n}{x_n} \right\} \Rightarrow \Delta \left\{ \frac{t_1}{x_1}, \dots, \frac{t_n}{x_n} \right\}}
\end{array}$$

■ **Figure 1** Proof rules for the sequent calculus $\mathcal{LK}_=$ with substitution.

infinite descent-style constructions. We show the soundness and completeness of these proof systems, and also compare their provability relations.

Our systems for \mathcal{L}_{RTC} are extensions of $\mathcal{LK}_=$, the sequent calculus for classical first-order logic with equality [16, 28] whose proof rules we show in Fig. 1.³ Sequents are expressions of the form $\Gamma \Rightarrow \Delta$, for finite sets of formulas Γ and Δ . We write Γ, Δ and Γ, φ as a shorthand for $\Gamma \cup \Delta$ and $\Gamma \cup \{\varphi\}$ respectively, and $\text{fv}(\Gamma)$ for the set of free variables of the formulas in the set Γ . A sequent $\Gamma \Rightarrow \Delta$ is valid if and only if the formula $\bigwedge_{\varphi \in \Gamma} \varphi \rightarrow \bigvee_{\psi \in \Delta} \psi$ is.

3.1 The Finitary Proof System

We briefly summarise the finitary proof system for \mathcal{L}_{RTC} . For more details see [10, 11]. We write $\varphi(x_1, \dots, x_n)$ to emphasise that the formula φ may contain x_1, \dots, x_n as free variables.

► **Definition 6.** The proof system RTC_G for \mathcal{L}_{RTC} is defined by adding to $\mathcal{LK}_=$ the following inference rules:

$$\overline{\Gamma \Rightarrow \Delta, (\text{RTC}_{x,y} \varphi)(s, s)} \quad (1)$$

$$\frac{\Gamma \Rightarrow \Delta, (\text{RTC}_{x,y} \varphi)(s, r) \quad \Gamma \Rightarrow \Delta, \varphi \left\{ \frac{r}{x}, \frac{t}{y} \right\}}{\Gamma \Rightarrow \Delta, (\text{RTC}_{x,y} \varphi)(s, t)} \quad (2)$$

$$\frac{\Gamma, \psi(x), \varphi(x, y) \Rightarrow \Delta, \psi \left\{ \frac{y}{x} \right\}}{\Gamma, \psi \left\{ \frac{s}{x} \right\}, (\text{RTC}_{x,y} \varphi)(s, t) \Rightarrow \Delta, \psi \left\{ \frac{t}{x} \right\}} \quad x \notin \text{fv}(\Gamma, \Delta) \text{ and } y \notin \text{fv}(\Gamma, \Delta, \psi) \quad (3)$$

Rule (3) is a generalized induction principle. It states that if an extension of formula ψ is closed under the relation induced by φ , then it is also closed under the reflexive transitive closure of that relation. In the case of arithmetic this rule captures the induction rule of Peano's Arithmetics PA [11].

³ Here we take $\mathcal{LK}_=$ to include the substitution rule, which was not a part of the original systems.

3.2 Infinitary Proof Systems

We now present our infinitary proof systems for \mathcal{L}_{RTC} which are based on the principle of infinite descent. This is in contrast to infinite-width proof systems based on a variant of the infinite branching ω -rule [25, 17]. Such systems have been widely investigated and known to be useful for attaining completeness (e.g. for arithmetics). Nonetheless, the infinite ω -rule renders them practically useless for automated reasoning. Since our motivation here is that of effectiveness and automation we opt for a finite system in which we allow infinite-height, non-well-founded proofs.

► **Definition 7.** The infinitary proof system RTC_G^ω for \mathcal{L}_{RTC} is defined like RTC_G , but replacing Rule (3) by:

$$\frac{\Gamma, s = t \Rightarrow \Delta \quad \Gamma, (\text{RTC}_{x,y} \varphi)(s, z), \varphi \left\{ \frac{z}{x}, \frac{t}{y} \right\} \Rightarrow \Delta}{\Gamma, (\text{RTC}_{x,y} \varphi)(s, t) \Rightarrow \Delta} \quad (4)$$

where z is fresh, i.e. z does not occur free in Γ , Δ , or $(\text{RTC}_{x,y} \varphi)(s, t)$. The formula $(\text{RTC}_{x,y} \varphi)(s, z)$ in the right-hand premise is called the *immediate ancestor* (cf. [7, §1.2.3]) of the principal formula, $(\text{RTC}_{x,y} \varphi)(s, t)$, in the conclusion.

There is an asymmetry between Rule (2), in which the intermediary is an arbitrary term r , and Rule (4), where we use a variable z . This is necessary to obtain the soundness of the cyclic proof system. It is used to show that when there is a counter-model for the conclusion of a rule, then there is also a counter-model for one of its premises that is, in a sense that we make precise below, “smaller”. In the case that $s \neq t$, using a fresh z allows us to pick from *all* possible counter-models of the conclusion, from which we may then construct the required counter-model for the right-hand premise. If we allowed an arbitrary term r instead, this might restrict the counter-models we can choose from, only leaving ones “larger” than the one we had for the conclusion. See Lemma 15 below for more details.

Proofs in this system are possibly infinite derivation trees. However, not all infinite derivations are proofs: only those that admit an infinite descent argument. Thus we use the terminology “pre-proof” for derivations.

► **Definition 8 (Pre-proofs).** An RTC_G^ω *pre-proof* is a possibly infinite (i.e. non-well-founded) derivation tree formed using the inference rules. A *path* in a pre-proof is a possibly infinite sequence of sequents $s_0, s_1, \dots, (s_n)$ such that s_0 is the root sequent of the proof, and s_{i+1} is a premise of s_i for each $i < n$.

The following definitions tell us how to track *RTC* formulas through a pre-proof, and allow us to formalize inductive arguments via infinite descent.

► **Definition 9 (Trace Pairs).** Let τ and τ' be *RTC* formulas occurring in the left-hand side of the conclusion s and a premise s' , respectively, of (an instance of) an inference rule. (τ, τ') is said to be a *trace pair* for (s, s') if the rule is:

- the (Subst) rule, and $\tau = \tau'\theta$ where θ is the substitution associated with the rule instance;
- Rule (4), and either:
 - (a) τ is the principal formula of the rule instance and τ' is the immediate ancestor of τ , in which case we say that the trace pair is *progressing*;
 - (b) otherwise, $\tau = \tau'$.
- any other rule, and $\tau = \tau'$.

► **Definition 10** (Traces). A *trace* is a (possibly infinite) sequence of *RTC* formulas. We say that a trace $\tau_1, \tau_2, \dots, (\tau_n)$ follows a path $s_1, s_2, \dots, (s_m)$ in a pre-proof \mathcal{P} if, for some $k \geq 0$, each consecutive pair of formulas (τ_i, τ_{i+1}) is a trace pair for (s_{i+k}, s_{i+k+1}) . If (τ_i, τ_{i+1}) is a progressing pair then we say that the trace *progresses* at i , and we say that the trace is *infinitely progressing* if it progresses at infinitely many points.

Proofs, then, are pre-proofs which satisfy a global trace condition.

► **Definition 11** (Infinite Proofs). A RTC_G^ω *proof* is a pre-proof in which every infinite path is followed by some infinitely progressing trace.

Clearly, we cannot reason effectively about such infinite proofs in general. In order to do so we need to restrict our attention to those proof trees which are finitely representable. These are the *regular* infinite proof trees, which contain only finitely many *distinct* subtrees. They can be specified as systems of recursive equations or, alternatively, as cyclic *graphs* [14]. Note that a given regular infinite proof may have many different graph representations. One possible way of formalizing such proof graphs is as standard proof trees containing open nodes (called buds), to each of which is assigned a syntactically equal internal node of the proof (called a companion). Due to space limitation, we elide a formal definition of cyclic proof graphs (see, e.g., Sect. 7 in [6]) and rely on the reader's basic intuitions.

► **Definition 12** (Cyclic Proofs). The cyclic proof system CRTC_G^ω for \mathcal{L}_{RTC} is the subsystem of RTC_G^ω comprising of all and only the finite and *regular* infinite proofs (i.e. those proofs that can be represented as finite, possibly cyclic, graphs).

Note that it is decidable whether a cyclic pre-proof satisfies the global trace condition, using a construction involving an inclusion between Büchi automata (see, e.g., [4, 26]). However since this requires complementing Büchi automata (a PSPACE procedure), our system cannot be considered a proof system in the Cook-Reckhow sense [13]. Notwithstanding, checking the trace condition for cyclic proofs found in practice is not prohibitive [23, 29].

3.3 Soundness and Completeness

The rich expressiveness of TC logic entails that the effective system RTC_G which is sound w.r.t. the standard semantics, cannot be complete (much like the case for LKID). It is however both sound and complete w.r.t. Henkin semantics.

► **Theorem 13** (Soundness and Completeness of RTC_G [9]). RTC_G is sound for standard semantics, and also sound and complete for Henkin semantics.

Note that the system RTC_G as presented here does not admit cut elimination. The culprit is the induction rule (3), which does not permute with cut. We may obtain admissibility of cut by using the following alternative formulation of the induction rule which, like the induction rule for LKID, incorporates a cut with the induction formula ψ .

$$\frac{\Gamma \Rightarrow \psi \left\{ \frac{s}{x} \right\} \quad \Gamma, \psi(x), \varphi(x, y) \Rightarrow \psi \left\{ \frac{y}{x} \right\} \quad \Gamma, \psi \left\{ \frac{t}{x} \right\} \Rightarrow \Delta}{\Gamma, (\text{RTC}_{x,y} \varphi)(s, t) \Rightarrow \Delta} \quad x \notin \text{fv}(\Gamma, \Delta), y \notin \text{fv}(\Gamma, \Delta, \psi)$$

For the system with this rule, a simple adaptation of the completeness proof in [9], in the spirit of the corresponding proof for LKID in [6], suffices to obtain cut-free completeness. However, the tradeoff is that the resulting cut-free system no longer has the sub-formula property. In contrast, cut-free proofs in RTC_G do satisfy the sub-formula property, for a generalized notion of a subformula that incorporates substitution instances (as in $\mathcal{LK}_=$).

We remark that the soundness proof of LKID is rather complex since it must handle different types of mutual dependencies between the inductive predicates. For RTC_G the proof is much simpler due to the uniformity of the rules for the RTC operator.

The infinitary system RTC_G^ω , in contrast to the finitary system RTC_G , is both sound and complete w.r.t. the standard semantics. To prove soundness, we make use of the following notion of *measure* for RTC formulas.

► **Definition 14** (Degree of RTC Formulas). For $\phi \equiv (\text{RTC}_{x,y} \varphi)(s, t)$, define $\delta_\phi(M, v) = 0$ if $v(s) = v(t)$, and $\delta_\phi(M, v) = n$ if $v(s) \neq v(t)$ and a_0, \dots, a_n is a minimal-length sequence of elements in the domain of M such that $v(s) = a_0$, $v(t) = a_n$, and $M, v[x := a_i, y := a_{i+1}] \models \varphi$ for $0 \leq i < n$. We call $\delta_\phi(M, v)$ the *degree* of ϕ with respect to the model M and valuation v .

Soundness then follows from the following fundamental lemma.

► **Lemma 15** (Descending Counter-models). *If there exists a standard model M and valuation v that invalidates the conclusion s of (an instance of) an inference rule, then*

- 1) *there exists a standard model M' and valuation v' that invalidates some premise s' of the rule; and*
- 2) *if (τ, τ') is a trace pair for (s, s') then $\delta_{\tau'}(M', v') \leq \delta_\tau(M, v)$. Moreover, if (τ, τ') is a progressing trace pair then $\delta_{\tau'}(M', v') < \delta_\tau(M, v)$.*

As is standard for infinite descent inference systems [4, 5, 6, 15, 23, 29], the above result entails the local soundness of the inference rules (in our case, for standard first-order models). The presence of infinitely progressing traces for each infinite path in a RTC_G^ω proof ensures soundness via a standard infinite descent-style construction.

► **Theorem 16** (Soundness of RTC_G^ω). *If there is a RTC_G^ω proof of $\Gamma \Rightarrow \Delta$, then $\Gamma \Rightarrow \Delta$ is valid (w.r.t. the standard semantics)*

The soundness of the cyclic system is an immediate corollary, since each CRTC_G^ω proof is also a RTC_G^ω proof.

► **Corollary 17** (Soundness of CRTC_G^ω). *If there is a CRTC_G^ω proof of $\Gamma \Rightarrow \Delta$, then $\Gamma \Rightarrow \Delta$ is valid (w.r.t. the standard semantics)*

Following a standard technique (as used in e.g. [6]), we can show cut-free completeness of RTC_G^ω with respect to the standard semantics.

► **Definition 18** (Schedule). A *schedule element* E is defined as any of the following:

- a formula of the form $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \rightarrow \psi$;
- a pair of the form $\langle \forall x \varphi, t \rangle$ or $\langle \exists x \varphi, t \rangle$ where $\forall x \varphi$ and $\exists x \varphi$ are formulas and t is a term;
- a tuple of the form $\langle (\text{RTC}_{x,y} \varphi)(s, t), r, z, \Gamma, \Delta \rangle$ where $(\text{RTC}_{x,y} \varphi)(s, t)$ is a formula, r is a term, Γ and Δ are finite sequences of formulas, and z is a variable not occurring free in Γ, Δ , or $(\text{RTC}_{x,y} \varphi)(s, t)$; or
- a tuple of the form $\langle s = t, x, \varphi, n, \Gamma, \Delta \rangle$ where s and t are terms, x is a variable, φ is a formula, $n \in \{1, 2\}$, and Γ and Δ are finite sequences of formulas.

A *schedule* is a recursive enumeration of schedule elements in which every schedule element appears infinitely often (these exist since our language is countable).

Each schedule corresponds to an exhaustive search strategy for a cut-free proof for each sequent $\Gamma \Rightarrow \Delta$, via the following notion of a “search tree”.

► **Definition 19** (Search Tree). Given a schedule $\{E_i\}_{i>0}$, for each sequent $\Gamma \Rightarrow \Delta$ we inductively define an infinite sequence of (possibly open) derivation trees, $\{T_i\}_{i>0}$, such that T_1 consists of the single open node $\Gamma \Rightarrow \Delta$, and each T_{i+1} is obtained by replacing all suitable open nodes in T_i with applications of first axioms and then the left and right inference rules for the formula in the i^{th} schedule element.

We give the definition of T_{i+1} when E_i is an *RTC* schedule element, i.e. of the form $\langle (RTC_{x,y} \varphi)(s, t), r, z, \Gamma, \Delta \rangle$ (the other cases are similar). T_{i+1} is then obtained by:

1. first closing as such any open node that is an instance of an axiom (after left and right weakening, if necessary);
2. next, replacing every open node $\Gamma', (RTC_{x,y} \varphi)(s, t) \Rightarrow \Delta'$ of the resulting tree for which $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$ with the derivation:

$$\frac{\Gamma', (RTC_{x,y} \varphi)(s, t), s = t \Rightarrow \Delta' \quad \Gamma', (RTC_{x,y} \varphi)(s, t), (RTC_{x,y} \varphi)(s, z), \varphi \left\{ \frac{z}{x}, \frac{t}{y} \right\} \Rightarrow \Delta'}{\Gamma', (RTC_{x,y} \varphi)(s, t) \Rightarrow \Delta'} \quad (4)$$

3. finally, replacing every open node $\Gamma' \Rightarrow \Delta', (RTC_{x,y} \varphi)(s, t)$ of the resulting tree with the derivation:

$$\frac{\Gamma' \Rightarrow \Delta', (RTC_{x,y} \varphi)(s, t), (RTC_{x,y} \varphi)(s, r) \quad \Gamma' \Rightarrow \Delta', (RTC_{x,y} \varphi)(s, t), \varphi \left\{ \frac{r}{x}, \frac{t}{y} \right\}}{\Gamma' \Rightarrow \Delta', (RTC_{x,y} \varphi)(s, t)} \quad (2)$$

The limit of the sequence $\{T_i\}_{i>0}$ is a possibly infinite (and possibly open) derivation tree called the *search tree* for $\Gamma \Rightarrow \Delta$ with respect to the schedule $\{E_i\}_{i>0}$, and denoted by T_ω .

Search trees are, by construction, recursive and cut-free. We construct special “sequents” out of search trees, called *limit sequents*, as follows.

► **Definition 20** (Limit Sequents). When a search tree T_ω is not an RTC_G^ω proof, either:

- (1) it is not even a pre-proof, i.e. it contains an open node; or
- (2) it is a pre-proof but contains an infinite branch that fails to satisfy the global trace condition.

In case 1 it contains an open node to which, necessarily, no schedule element applies (e.g. a sequent containing only atomic formulas), for which we write $\Gamma_\omega \Rightarrow \Delta_\omega$. In case 2 the global trace condition fails, so there exists an infinite path $\{\Gamma_i \Rightarrow \Delta_i\}_{i>0}$ in T_ω which is followed by no infinitely progressing traces; we call this path the *untraceable branch* of T_ω . We then define $\Gamma_\omega = \bigcup_{i>0} \Gamma_i$ and $\Delta_\omega = \bigcup_{i>0} \Delta_i$, and call $\Gamma_\omega \Rightarrow \Delta_\omega$ the *limit sequent*.⁴

Note that use of the word “sequent” here is an abuse of nomenclature, since limit sequents may be infinite and thus technically not sequents. However their purpose is not to play a role in syntactic proofs, but to induce counter-models as follows.

► **Definition 21** (Counter-interpretations). Assume a search tree T_ω which is not a RTC_G^ω proof with limit sequent $\Gamma_\omega \Rightarrow \Delta_\omega$. Let \sim be the smallest congruence relation on terms such that $s \sim t$ whenever $s = t \in \Gamma_\omega$. Define a structure $M_\omega = \langle D, I \rangle$ as follows (where $[t]$ stands for the \sim -equivalence class of t):

- $D = \{[t] \mid t \text{ is a term}\}$ (i.e. the set of terms quotiented by the relation \sim).
- For every k -ary function symbol f : $I(f)([t_1], \dots, [t_k]) = [f(t_1, \dots, t_k)]$
- For every k -ary relation symbol q : $I(q) = \{([t_1], \dots, [t_k]) \mid q(t_1, \dots, t_k) \in \Gamma_\omega\}$

We also define a valuation v_ω for M_ω by $v_\omega(x) = [x]$ for all variables x .

⁴ To be rigorous, we may pick e.g. the left-most open node or untraceable branch.

Counter-interpretations $\langle M_\omega, v_\omega \rangle$ have the following property, meaning that M_ω is a counter-model for the corresponding sequent $\Gamma \Rightarrow \Delta$ if its search tree T_ω is not a proof.

► **Lemma 22.** *If $\psi \in \Gamma_\omega$ then $M_\omega, v_\omega \models \psi$; and if $\psi \in \Delta_\omega$ then $M_\omega, v_\omega \not\models \psi$.*

The completeness result therefore follows since, by construction, a sequent S is contained within its corresponding limit sequents. Thus, for any sequent S , if some search tree T_ω contracted for S is not an RTC_G^ω proof then it follows from Lemma 22 that S is not valid (M_ω is a counter model for it). Hence if S is valid, then T_ω is a recursive RTC_G^ω proof for it.

► **Theorem 23 (Completeness).** *RTC_G^ω is complete for standard semantics.*

We obtain admissibility of cut as the search tree T_ω is cut-free.

► **Corollary 24 (Cut admissibility).** *Cut is admissible in RTC_G^ω .*

3.4 \mathcal{L}_{RTC} with Pairs

To obtain the full inductive expressivity we must allow the formation of the transitive closure of not only binary relations, but any $2n$ -ary relation. In [1] it was shown that taking such a RTC^n operator for every n (instead of just for $n = 1$) results in a more expressive logic, namely one that captures all finitary first-order definable inductive definitions and relations. Nonetheless, from a proof theoretical point of view having infinitely many such operators is suboptimal. Thus, we here instead incorporate the notion of ordered pairs and use it to encode such operators. For example, writing $\langle x, y \rangle$ for the application of the pairing function $\langle \rangle(x, y)$, the formula $(\text{RTC}_{x_1, x_2, y_1, y_2}^2 \varphi)(s_1, s_2, t_1, t_2)$ can be encoded by:

$$(\text{RTC}_{x,y} \exists x_1, x_2, y_1, y_2 . x = \langle x_1, x_2 \rangle \wedge y = \langle y_1, y_2 \rangle \wedge \varphi)(\langle s_1, s_2 \rangle, \langle t_1, t_2 \rangle)$$

Accordingly, we may assume languages that explicitly contain a pairing function, providing that we (axiomatically) restrict to structures that interpret it as such (i.e. the *admissible* structures). For such languages we can consider two induced semantics: admissible standard semantics and admissible Henkin semantics, obtained by restricting the (first-order part of the) structures to be admissible.

The above proof systems are extended to capture ordered pairs as follows.

► **Definition 25.** For a signature containing at least one constant c , and a binary function symbol denoted by $\langle \rangle$, the proof systems $\langle \text{RTC} \rangle_G$, $\langle \text{RTC} \rangle_G^\omega$, and $\langle \text{CRTC} \rangle_G^\omega$ are obtained from RTC_G , RTC_G^ω , CRTC_G^ω (respectively) by the addition of the following rules:

$$\frac{\Gamma \Rightarrow \langle x, y \rangle = \langle u, v \rangle, \Delta}{\Gamma \Rightarrow x = u \wedge y = v, \Delta} \qquad \frac{}{\Gamma, \langle x, y \rangle = c \Rightarrow \Delta}$$

The proofs of Theorems 13 and 23 can easily be extended to obtain the following results for languages with a pairing function. For completeness, the key observation is that the model of the counter-interpretation is one in which every binary function is a pairing function. That is, the interpretation of any binary function is such that satisfies the standard pairing axioms. Therefore, the model of the counter-interpretation is an admissible structure.

► **Theorem 26 (Soundness and Completeness of $\langle \text{RTC} \rangle_G$ and $\langle \text{RTC} \rangle_G^\omega$).** *The proof systems $\langle \text{RTC} \rangle_G$ and $\langle \text{RTC} \rangle_G^\omega$ are both sound and complete for the admissible forms of Henkin and standard semantics, respectively.*

$$\begin{array}{c}
\frac{\Gamma, \psi \left\{ \frac{v}{x} \right\}, (RTC_{x,y} \varphi)(v, w) \Rightarrow \Delta, \psi \left\{ \frac{w}{x} \right\}}{\Gamma, \psi \left\{ \frac{v}{x} \right\}, (RTC_{x,y} \varphi)(v, z) \Rightarrow \Delta, \psi \left\{ \frac{z}{x} \right\}} \text{ (Subst)} \quad \frac{\Gamma, \psi, \varphi \Rightarrow \Delta, \psi \left\{ \frac{y}{x} \right\}}{\Gamma, \psi \left\{ \frac{z}{x} \right\}, \varphi \left\{ \frac{z}{x}, \frac{w}{y} \right\} \Rightarrow \Delta, \psi \left\{ \frac{w}{x} \right\}} \text{ (Subst)} \\
\frac{\Gamma, \psi \left\{ \frac{v}{x} \right\}, (RTC_{x,y} \varphi)(v, z), \varphi \left\{ \frac{z}{x}, \frac{w}{y} \right\} \Rightarrow \Delta, \psi \left\{ \frac{w}{x} \right\}}{\Gamma, \psi \left\{ \frac{v}{x} \right\} \Rightarrow \Delta, \psi \left\{ \frac{v}{x} \right\}} \text{ (WL, WR, Ax)} \quad \vdots \\
\frac{\Gamma, \psi \left\{ \frac{v}{x} \right\}, v = w \Rightarrow \Delta, \psi \left\{ \frac{w}{x} \right\}}{\Gamma, \psi \left\{ \frac{v}{x} \right\}, v = w \Rightarrow \Delta, \psi \left\{ \frac{w}{x} \right\}} \text{ (=L1)} \quad \vdots \\
\frac{\Gamma, \psi \left\{ \frac{v}{x} \right\}, (RTC_{x,y} \varphi)(v, w) \Rightarrow \Delta, \psi \left\{ \frac{w}{x} \right\}}{\Gamma, \psi \left\{ \frac{s}{x} \right\}, (RTC_{x,y} \varphi)(s, t) \Rightarrow \Delta, \psi \left\{ \frac{t}{x} \right\}} \text{ (Subst)} \quad (4)
\end{array}$$

■ **Figure 2** CRTC_G^ω derivation simulating Rule (3). The variables v and w are fresh (i.e. do not occur free in Γ , Δ , φ , or ψ).

4 Relating the Finitary and Infinitary Proof Systems

This section discusses the relation between the explicit and the cyclic system for TC. In Section 4.1 we show that the former is contained in the latter. The converse direction, which is much more subtle, is discussed in Section 4.2.

4.1 Inclusion of RTC_G in CRTC_G^ω

Provability in the explicit induction system implies provability in the cyclic system. The key property is that we can derive the explicit induction rule in the cyclic system, as shown in Figure 2.

► **Lemma 27.** *Rule (3) is derivable in CRTC_G^ω .*

This leads to the following result (an analogue to [6, Thm. 7.6]).

► **Theorem 28.** $\text{CRTC}_G^\omega \supseteq \text{RTC}_G$, and is thus complete w.r.t. Henkin semantics.

Lemma 27 is the TC counterpart of [6, Lemma 7.5]. It is interesting to note that the simulation of the explicit LKID induction rule in the cyclic LKID system is rather complex since each predicate has a slightly different explicit induction rule, which depends on the particular productions defining it. Thus, the construction for the cyclic LKID system must take into account the possible forms of arbitrary productions. In contrast, CRTC_G^ω provides a single, uniform way to unfold an RTC formula: the construction given in Fig. 2 is the cyclic representation of the RTC operator semantics, with the variables v and w implicitly standing for arbitrary terms (that we subsequently substitute for).

This uniform syntactic translation of the explicit RTC_G induction rule into CRTC_G^ω allows us to *syntactically* identify a proper subset of cyclic proofs which is also complete w.r.t. Henkin semantics.⁵ The criterion we use is based on the notion of *overlapping cycles*. Recall the definition of a *basic* cycle, which is a path in a (proof) graph starting and ending at the same point, but containing no other repeated nodes. We say that two distinct (i.e. not identical up to permutation) basic cycles *overlap* if they share any nodes in common, i.e. at

⁵ Note it is not clear that a similar complete structural restriction is possible for LKID.

some point they both traverse the same path in the graph. We say that a cyclic proof is *non-overlapping* whenever no two distinct basic cycles it contains overlap. The restriction to non-overlapping proofs has an advantage for automation, since one has only to search for cycles in one single branch.

► **Definition 29** (Normal Cyclic Proofs). The *normal* cyclic proof system NCRTC_G^ω is the subsystem of RTC_G^ω comprising of all and only the non-overlapping cyclic proofs.

The following theorem is immediate due to the fact that the translation of an RTC_G proof into CRTC_G^ω , using the construction shown in Figure 2, results in a proof with no overlapping cycles.

► **Theorem 30.** $\text{NCRTC}_G^\omega \supseteq \text{RTC}_G$.

Henkin-completeness of the normal cyclic system then follows from Theorem 30 and Theorem 13.

4.2 Inclusions of CRTC_G^ω in RTC_G

This section addresses the question of whether the cyclic system is equivalent to the explicit one, or strictly stronger. In [6] it was conjectured that for the system with inductive definitions, LKID and CLKID^ω are equivalent. Later, it was shown that they are indeed equivalent when containing arithmetics [3, 26]. We obtain a corresponding theorem in Section 4.2.1 for the TC systems. However, it was also shown in [2] that in the general case the cyclic system is stronger than the explicit one. We discuss the general case for TC and its subtleties in Section 4.2.2.

4.2.1 The Case of Arithmetics

Let \mathcal{L}_{RTC} be a language based on the signature $\{0, s, +\}$. Let RTC_G+A and $\text{CRTC}_G^\omega+A$ be the systems for \mathcal{L}_{RTC} obtained by adding to RTC_G and CRTC_G^ω , respectively, the standard axioms of PA together with the *RTC*-characterization of the natural numbers, i.e.:

- (i) $sx = 0 \Rightarrow$
- (ii) $sx = sy \Rightarrow x = y$
- (iii) $\Rightarrow x + 0 = x$
- (iv) $\Rightarrow x + sy = s(x + y)$
- (v) $\Rightarrow (\text{RTC}_{w,u} sw = u)(0, x)$

Note that we do not need to assume multiplication explicitly in the signature, nor do we need to add axioms for it, since multiplication is definable in \mathcal{L}_{RTC} and its standard axioms are derivable [1, 11].

Recall that we can express facts about sequences of numbers in PA by using a β -function such that for any finite sequence k_0, k_1, \dots, k_n there is some c such that for all $i \leq n$, $\beta(c, i) = k_i$. Accordingly, let B be a well-formed formula of the language of PA with three free variables which captures in PA a β -function. For each formula φ of the language of PA define $\varphi^\beta := \varphi$, and define $((\text{RTC}_{x,y} \varphi)(s, t))^\beta$ to be:

$$s = t \vee (\exists z, c. B(c, 0, s) \wedge B(c, sz, t) \wedge (\forall u \leq z. \exists v, w. B(c, u, v) \wedge B(c, su, w) \wedge \varphi^\beta \left\{ \frac{v}{x}, \frac{w}{y} \right\}))$$

The following result, which was proven in [8, 11], establishes an equivalence between RTC_G+A and PA_G (a Gentzen-style system for PA). It is mainly based on the fact that in RTC_G+A all instances of PA_G induction rule are derivable.

► **Theorem 31** (cf. [11]). *The following hold:*

1. $\vdash_{\text{RTC}_G+A} \varphi \Leftrightarrow \varphi^\beta$.
2. $\vdash_{\text{RTC}_G+A} \Gamma \Rightarrow \Delta$ *iff* $\vdash_{\text{PA}_G} \Gamma^\beta \Rightarrow \Delta^\beta$.

We show a similar equivalence holds between the cyclic system CRTC_G^ω and CA_G , a cyclic system for arithmetic shown to be equivalent to PA_G [26].

► **Theorem 32.** $\vdash_{\text{CRTC}_G^\omega+A} \Gamma \Rightarrow \Delta$ *iff* $\vdash_{\text{CA}_G} \Gamma^\beta \Rightarrow \Delta^\beta$.

These results allow us to show an equivalence between the finitary and cyclic systems for TC with arithmetic.

► **Theorem 33.** RTC_G+A and $\text{CRTC}_G^\omega+A$ are equivalent.

Note that the result above can easily be extended to show that adding the same set of additional axioms to both RTC_G+A and $\text{CRTC}_G^\omega+A$ results in equivalent systems. Also note that in the systems with pairs, to embed arithmetics there is no need to explicitly include addition and its axioms. Thus, by only including the signature $\{0, s\}$ and the corresponding axioms for it we can obtain that $\langle \text{RTC} \rangle_G+A$ and $\langle \text{CRTC} \rangle_G^\omega+A$ are equivalent.

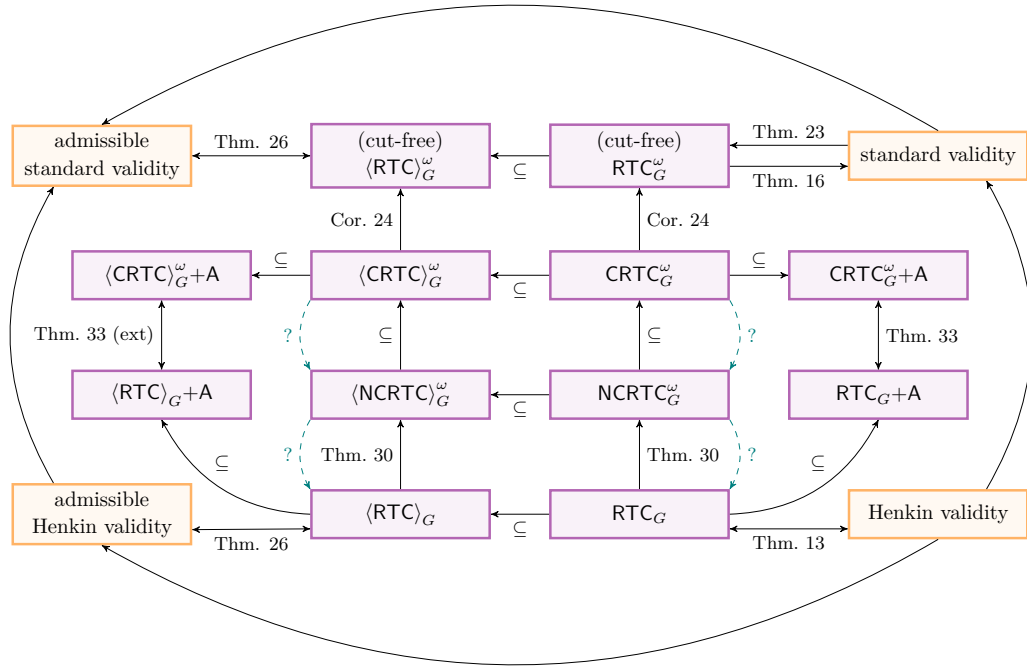
In [3], the equivalence result of [26] was improved to show it holds for any set of inductive predicates containing the natural number predicate \mathbf{N} . On the one hand, our result goes beyond that of [3] as it shows the equivalence for systems with a richer notion of inductive definition, due to the expressiveness of TC. On the other hand, TC does not support restricting the set of inductive predicates, i.e. the RTC operator may operate on any formula in the language. To obtain a finer result which corresponds to that of [3] we need to further explore the transformations between proofs in the two systems. This is left for future work.

4.2.2 The General Case

As mentioned, the general equivalence conjecture between LKID and CLKID^ω was refuted in [2], by providing a concrete example of a statement which is provable in the cyclic system but not in the explicit one. The statement (called 2-Hydra) involves a predicate encoding a binary version of the “hydra” induction scheme for natural numbers given in [20], and expresses that every pair of natural numbers is related by the predicate.⁶ However, a careful examination of this counter-example reveals that it only refutes a strong form of the conjecture, according to which both systems are based on the same set of productions. In fact, already in [2] it is shown that if the explicit system is extended by another inductive predicate, namely one expressing the \leq relation, then the 2-Hydra counter-example becomes provable. Therefore, the less strict formulation of the question, namely whether for any proof in CLKID_ϕ^ω there is a proof in $\text{LKID}_{\phi'}$ for some $\phi' \supseteq \phi$, has not yet been resolved. Notice that in TC the equivalence question is of this weaker variety, since the RTC operator “generates” all inductive definitions at once. That is, there is no *a priori* restriction on the inductive predicates one is allowed to use. Indeed, the 2-Hydra counter-example from [2] can be expressed in \mathcal{L}_{RTC} and proved in CRTC_G^ω . However, this does not produce a counter-example for TC since it is also provable in RTC_G , due to the fact that $s \leq t$ is definable via the RTC formula $(\text{RTC}_{w,u} s w = u)(s, t)$.

Despite our best efforts, we have not yet managed to settle this question, which appears to be harder to resolve in the TC setting. One possible approach to solving it is the semantical one, i.e. exploiting the fact that the explicit system is known to be sound w.r.t. Henkin

⁶ In fact, the falsifying Henkin model constructed in [2] also satisfies the “0-axiom” $(\forall x. 0 \neq s x)$, and the “s-axiom” $(\forall x, y. s x = s y \rightarrow x = y)$ stipulating injectivity of the successor function, and so the actual counter-example to equivalence is the sequent: $(0, s)\text{-axioms} \Rightarrow 2\text{-Hydra}$.



■ **Figure 3** Diagrammatic Summary of our Results.

semantics. This is what was done in [2]. Thus, to show strict inclusion one could construct an alternative statement that is provable in $CRTC_G^\omega$ whilst also demonstrating a Henkin model for TC that is not a model of the statement. However, constructing a TC Henkin model appears to be non-trivial, due to its rich inductive power. In particular, it is not at all clear whether the structure that underpins the LKID counter-model for 2-Hydra admits a Henkin model for TC. Alternatively, to prove equivalence, one could show that $CRTC_G^\omega$ is also sound w.r.t. Henkin semantics. Here, again, proving this does not seem to be straightforward.

In our setting, there is also the question of the inclusion of $CRTC_G^\omega$ in $NCRTC_G^\omega$, which amounts to the question of whether overlapping cycles can be eliminated. Moreover, we can ask if $NCRTC_G^\omega$ is included in RTC_G , independently of whether this also holds for $CRTC_G^\omega$. Again, the semantic approach described above may prove fruitful in answering these questions.

5 Conclusions and Future Work

We developed a natural infinitary proof system for transitive closure logic which is cut-free complete for the standard semantics and subsumes the explicit system. We further explored its restriction to cyclic proofs which provides the basis for an effective system for automating inductive reasoning. In particular, we syntactically identified a subset of cyclic proofs that is Henkin-complete. A summary of the proof systems we have studied in this paper, and their interrelationships, is shown in Figure 3. Where an edge between systems is labelled with an inclusion \subseteq , this signifies that a proof in the source system is already a proof in the destination system.

As mentioned in the introduction, as well as throughout the paper, this research was motivated by other work on systems of inductive definitions, particularly the LKID framework of [6], its infinitary counterpart $LKID^\omega$, and its cyclic subsystem $CLKID^\omega$. In terms of the expressive power of the underlying logic, TC (assuming pairs) subsumes the inductive machinery underlying LKID. This is because for any inductive predicate P of LKID, there

is an \mathcal{L}_{RTC} formula ψ such that for every standard admissible structure M for \mathcal{L}_{RTC} , P has the same interpretation as ψ under M . This is due to Thm. 3 in [1] and the fact that the interpretation of P must necessarily be a recursively enumerable set. As for the converse inclusion, for any positive \mathcal{L}_{RTC} formula there is a production of a corresponding LKID inductive definition. However, the *RTC* operator can also be applied on complex formulas (whereas LKID productions only consider atomic predicates). This indicates that TC might be more expressive. It was noted in [6, p. 1180] that complex formulas may be handled by stratifying the theory of LKID, similar to [21], but the issue of relative expressiveness of the resulting theory is not addressed. While we strongly believe it is the case that TC is strictly more expressive than the logic of LKID, proving so is left for future work. Also left for future research is establishing the comparative status of the corresponding formal proof systems.

In addition to the open question of the (in)equivalence of RTC_G and CRTC_G^ω in the general case, discussed in Section 4.2, several other questions and directions for further study naturally arise from the work of this paper. An obvious one would be to implement our cyclic proof system in order to investigate the practicalities of using TC logic to support automated inductive reasoning. More theoretically it is already clear that TC logic, as a framework, diverges from existing systems for inductive reasoning (e.g. LKID) in interesting, non-trivial ways. The uniformity provided by the transitive closure operator may offer a way to better study the relationship between implicit and explicit induction, e.g. in the form of cuts required in each system, or the relative complexity of proofs that each system admits. Moreover, it seems likely that *coinductive* reasoning can also be incorporated into the formal system. Determining whether, and to what extent, these are indeed the case is left for future work.

References

- 1 Arnon Avron. Transitive Closure and the Mechanization of Mathematics. In F. D. Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, volume 28 of *Applied Logic Series*, pages 149–171. Springer, Netherlands, 2003. doi:10.1007/978-94-017-0253-9_7.
- 2 Stefano Berardi and Makoto Tatsuta. Classical System of Martin-Löf’s Inductive Definitions Is Not Equivalent to Cyclic Proof System. In *Proceedings of FOSSACS, Uppsala, Sweden, April 22–29, 2017*, pages 301–317, Berlin, Heidelberg, 2017. Springer. doi:10.1007/978-3-662-54458-7_18.
- 3 Stefano Berardi and Makoto Tatsuta. Equivalence of Inductive Definitions and Cyclic Proofs Under Arithmetic. In *Proceedings of LICS, Reykjavik, Iceland, June 20–23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005114.
- 4 James Brotherston. Formalised Inductive Reasoning in the Logic of Bunched Implications. In *Proceedings of SAS, Kongens Lyngby, Denmark, August 22–24, 2007*, pages 87–103, 2007. doi:10.1007/978-3-540-74061-2_6.
- 5 James Brotherston, Richard Bornat, and Cristiano Calcagno. Cyclic Proofs of Program Termination in Separation Logic. In *Proceedings of POPL, San Francisco, California, USA, January 7–12, 2008*, pages 101–112, 2008. doi:10.1145/1328438.1328453.
- 6 James Brotherston and Alex Simpson. Sequent Calculi for Induction and Infinite Descent. *Journal of Logic and Computation*, 21(6):1177–1216, 2010.
- 7 Samuel R. Buss. *Handbook of Proof Theory*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 1998.
- 8 Liron Cohen. Ancestral Logic and Equivalent Systems. Master’s thesis, Tel-Aviv University, Israel, 2010.
- 9 Liron Cohen. Completeness for Ancestral Logic via a Computationally-Meaningful Semantics. In *Proceedings of TABLEAUX, Brasilia, Brazil, September 25–28, 2017*, pages 247–260, 2017. doi:10.1007/978-3-319-66902-1_15.

- 10 Liron Cohen and Arnon Avron. Ancestral Logic: A Proof Theoretical Study. In U. Kohlenbach, editor, *Logic, Language, Information, and Computation*, volume 8652 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 2014.
- 11 Liron Cohen and Arnon Avron. The Middle Ground–Ancestral Logic. *Synthese*, pages 1–23, 2015.
- 12 Liron Cohen and Reuben N. S. Rowe. Infinitary and Cyclic Proof Systems for Transitive Closure Logic. *CoRR*, abs/1802.00756, 2018. [arXiv:1802.00756](https://arxiv.org/abs/1802.00756).
- 13 Stephen A. Cook and Robert A. Reckhow. The Relative Efficiency of Propositional Proof Systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979.
- 14 Bruno Courcelle. Fundamental Properties of Infinite Trees. *Theor. Comput. Sci.*, 25:95–169, 1983. [doi:10.1016/0304-3975\(83\)90059-2](https://doi.org/10.1016/0304-3975(83)90059-2).
- 15 Anupam Das and Damien Pous. A Cut-Free Cyclic Proof System for Kleene Algebra. In *Proceedings of TABLEAUX, Brasilia, Brazil, September 25–28, 2017*, pages 261–277, 2017. [doi:10.1007/978-3-319-66902-1_16](https://doi.org/10.1007/978-3-319-66902-1_16).
- 16 Gerhard Gentzen. Untersuchungen über das Logische Schließen. I. *Mathematische Zeitschrift*, 39(1):176–210, 1935. [doi:10.1007/BF01201353](https://doi.org/10.1007/BF01201353).
- 17 Jean-Yves Girard. *Proof Theory and Logical Complexity*, volume 1. Humanities Press, 1987.
- 18 Leon Henkin. Completeness in the Theory of Types. *Journal of Symbolic Logic*, 15(2):81–91, 1950.
- 19 Ryo Kashima and Keishi Okamoto. General Models and Completeness of First-order Modal μ -calculus. *Journal of Logic and Computation*, 18(4):497–507, 2008.
- 20 Laurie Kirby and Jeff Paris. Accessible Independence Results for Peano Arithmetic. *Bulletin of the London Mathematical Society*, 14(4):285–293, 1982. [doi:10.1112/blms/14.4.285](https://doi.org/10.1112/blms/14.4.285).
- 21 Per Martin-Löf. Hauptsatz for the Intuitionistic Theory of Iterated Inductive Definitions. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 179–216. Elsevier, 1971. [doi:10.1016/S0049-237X\(08\)70847-4](https://doi.org/10.1016/S0049-237X(08)70847-4).
- 22 Per Martin-Löf and Giovanni Sambin. *Intuitionistic Type Theory*, volume 9. Bibliopolis Napoli, 1984.
- 23 Reuben N. S. Rowe and James Brotherston. Automatic Cyclic Termination Proofs for Recursive Procedures in Separation Logic. In *Proceedings of CPP, Paris, France, January 16–17, 2017*, pages 53–65, 2017. [doi:10.1145/3018610.3018623](https://doi.org/10.1145/3018610.3018623).
- 24 Luigi Santocanale. A Calculus of Circular Proofs and Its Categorical Semantics. In *Proceedings of FOSSACS, Grenoble, France, April 8–12, 2002*, pages 357–371. Springer Berlin Heidelberg, 2002. [doi:10.1007/3-540-45931-6_25](https://doi.org/10.1007/3-540-45931-6_25).
- 25 Kurt Schütte. Beweistheoretische Erfassung der unendlichen Induktion in der Zahlentheorie. *Mathematische Annalen*, 122:369–389, 1950/51.
- 26 Alex Simpson. Cyclic Arithmetic Is Equivalent to Peano Arithmetic. In *Proceedings of FOSSACS, Uppsala, Sweden, April 22–29, 2017*, pages 283–300, 2017. [doi:10.1007/978-3-662-54458-7_17](https://doi.org/10.1007/978-3-662-54458-7_17).
- 27 Christoph Sprenger and Mads Dam. On the Structure of Inductive Reasoning: Circular and Tree-Shaped Proofs in the μ -Calculus. In *Proceedings of FOSSACS, Warsaw, Poland, April 7–11, 2003*, pages 425–440. Springer Berlin Heidelberg, 2003. [doi:10.1007/3-540-36576-1_27](https://doi.org/10.1007/3-540-36576-1_27).
- 28 Gaisi Takeuti. *Proof Theory*. Courier Dover Publications, 1987.
- 29 Gadi Tellez and James Brotherston. Automatically Verifying Temporal Properties of Pointer Programs with Cyclic Proof. In *Proceedings of CADE, Gothenburg, Sweden, August 6–11, 2017*, pages 491–508, 2017. [doi:10.1007/978-3-319-63046-5_30](https://doi.org/10.1007/978-3-319-63046-5_30).

A Recursion-Theoretic Characterisation of the Positive Polynomial-Time Functions

Anupam Das¹

University of Copenhagen, Denmark
anupam.das@di.ku.dk

Isabel Oitavem²

CMA and DM, FCT, Universidade Nova de Lisboa, Portugal
oitavem@fct.unl.pt

Abstract

We extend work of Lautemann, Schwentick and Stewart [14] on characterisations of the “positive” polynomial-time predicates (**posP**, also called **mP** by Grigni and Sipser [11]) to function classes. Our main result is the obtention of a function algebra for the positive polynomial-time functions (**posFP**) by imposing a simple uniformity constraint on the bounded recursion operator in Cobham’s characterisation of **FP**. We show that a similar constraint on a function algebra based on *safe recursion*, in the style of Bellantoni and Cook [3], yields an “implicit” characterisation of **posFP**, mentioning neither explicit bounds nor explicit monotonicity constraints.

2012 ACM Subject Classification Theory of computation → Recursive functions

Keywords and phrases Monotone complexity, Positive complexity, Function classes, Function algebras, Recursion-theoretic characterisations, Implicit complexity, Logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.18

Acknowledgements The authors would like to thank Patrick Baillot, Sam Buss, Reinhard Kahle and the anonymous reviewers for several helpful discussions on this subject.

1 Introduction

Monotone functions abound in the theory of computation, e.g. sorting a string, and detecting cliques in graphs. They have been comprehensively studied in the setting of *circuit complexity*, via \neg -free circuits (usually called “monotone circuits”), cf. [13]. Most notably, Razborov’s seminal work [20] gave exponential lower bounds on the size of monotone circuits, and later refinements, cf. [1, 23], separated them from non-monotone circuits altogether.

The study of *uniform* monotone computation is a much less developed subject. Grigni and Sipser began a line of work studying the effect of restricting “negation” in computational models [11, 10]. One shortfall of their work was that deterministic classes lacked a bona fide treatment, with positive models only natively defined for nondeterministic classes. This means that positive versions of, say, **P** must rather be obtained via indirect characterisations, e.g. as **ALOGSPACE**. Later work by Lautemann, Schwentick and Stewart solved this problem by proposing a model of deterministic computation whose polynomial-time predicates coincide

¹ This work was supported by a Marie Skłodowska-Curie fellowship, *Monotonicity in Logic and Complexity*, ERC project 753431.

² This work is partially supported by the Portuguese Science Foundation, FCT, through the projects UID/MAT/00297/2013 and PTDC/MHC-FIL/2583/2014.



with several characterisations of \mathbf{P} once “negative” operations are omitted [14, 15]. This induces a robust definition of a class “**posP**”, the *positive* polynomial-time predicates [11, 10].

In this paper we extend this line of work to associated function classes (see, e.g., [5]), which are of natural interest for logical approaches to computational complexity, e.g. [4, 7]. Noting that several of the characterisations proposed by [14] make sense for function classes (and, indeed, coincide), we propose a *function algebra* for the “positive polynomial-time functions” on binary words (**posFP**) based on Cobham’s *bounded recursion on notation* [6]. We show that this algebra indeed coincides with certain characterisations proposed in [14], and furthermore give a function algebra based on *safe recursion*, in the style of Bellantoni and Cook [3]. The latter constitutes an entirely *implicit* characterisation of **posFP**, mentioning neither explicit bounds nor explicit monotonicity constraints. As far as we know, this is the first implicit approach to monotone computation.

This paper is structured as follows. In Sect. 2 we present preliminaries on monotone functions on binary strings and recall some notions of positive computation from [14, 15]. We show also that these models compute the same class of functions (Thm. 7), inducing our definition of **posFP**. In Sect. 3 we recall Cobham’s function algebra for **FP**, based on bounded recursion on notation, and introduce a uniform version of it, $u\mathbf{C}$, which we show is contained in **posFP** in Sect. 4 (Thm. 17). In Sect. 5 we prove some basic properties about $u\mathbf{C}$; we characterise the *tally* functions of $u\mathbf{C}$, those that return unary outputs on unary inputs, as just the unary codings of linear space functions on \mathbb{N} , by giving an associated function algebra (Thm. 21). We use this to recover a proof that $u\mathbf{C}$ is closed under a *simultaneous* version of its recursion scheme (Thm. 28), tracking the length of functions rather than usual methods relying on explicit pairing functions. In Sect. 6 we show the converse result that $u\mathbf{C}$ contains **posFP** (Thm. 30). Finally, in Sect. 7 we give a characterisation of **posFP** based on “safe” recursion (Thm. 36), and we give some concluding remarks in Sect. 8.

Throughout this work, we follow the convention of [14, 15], reserving the word “monotone” for the semantic level, and rather using “positive” to describe restricted models of computation.

2 Monotone functions and positive computation

We consider binary strings (or “words”), i.e. elements of $\{0,1\}^* = \bigcup_{n \in \mathbb{N}} \{0,1\}^n$, and for $x \in \{0,1\}^n$ we write $x(j)$ for the j^{th} bit of x , where $j = 0, \dots, n-1$. We follow the usual convention that bits are indexed from right (“least significant”) to left (“most significant”), e.g. as in [5]; for instance the word 011 has 0^{th} bit 1, 1^{st} bit 1 and 2^{nd} bit 0.

We write ε, s_0, s_1 for the usual generators of $\{0,1\}^*$, i.e. ε denotes the empty string, $s_0x = x0$ and $s_1x = x1$. We also write 1^n for 1 concatenated with itself n times, for $n \in \mathbb{N}$.

We consider functions of type $\{0,1\}^* \times \dots \times \{0,1\}^* \rightarrow \{0,1\}^*$. For $n \in \mathbb{N}$, we define \leq^n as the n -wise product order of \leq on $\{0,1\}$, i.e. for $x, y \in \{0,1\}^n$ we have $x \leq^n y$ if $\forall j < n. x(j) \leq y(j)$. The partial order \leq on $\{0,1\}^*$ is the union of all \leq^n , for $n \in \mathbb{N}$. A function $f : (\{0,1\}^*)^k \rightarrow \{0,1\}^*$ is *monotone* if $x_1 \leq y_1, \dots, x_k \leq y_k \implies f(\vec{x}) \leq f(\vec{y})$.

► **Example 1.** A recurring example we will consider is the *sorting* function $sort(x)$, which takes a binary word input and rearranges the bits so that all 0s occur before all 1s, left-right. Clearly $sort$ is monotone, and can be given the following recursive definition:

$$\begin{aligned} sort(\varepsilon) &= \varepsilon \\ sort(s_0x) &= 0sort(x) \\ sort(s_1x) &= sort(x)1 \end{aligned} \tag{1}$$

While in the binary case it may seem rather simple, we will see that *sort* nonetheless exemplifies well the difference between positive and non-positive computation.

One particular well-known feature of monotone functions, independent of any machine model, is that they are rather oblivious: the length of the output depends only on the length of the inputs:

► **Observation 2.** *Let $f(x_1, \dots, x_k)$ be a monotone function. Then, whenever $|x_1| = |y_1|, \dots, |x_k| = |y_k|$, we also have that $|f(\vec{x})| = |f(\vec{y})|$.*

Proof. Let $n_j = |x_j| = |y_j|$, for $1 \leq j \leq k$. We have both $f(\vec{x}) \leq f(1^{n_1}, \dots, 1^{n_k})$ and $f(\vec{y}) \leq f(1^{n_1}, \dots, 1^{n_k})$, by monotonicity, so indeed all these outputs have the same length. ◀

One way to define a positive variant of **FP** is to consider \neg -free circuits that are in some sense uniform. [14, 15] followed this approach too for **P**, showing that one of the strongest levels of uniformity (**P**) and one of the weakest levels (“quantifier-free”) needed to characterise **P** indeed yield the same class of languages when describing \neg -free circuits. We show that a similar result holds for classes of functions, when allowing circuits to have many output wires. Most of the techniques used in this section are standard, so we keep to a high-level exposition, rather dedicating space to examples of the notions of positive computation presented.

We consider Δ_0 -uniformity rather than quantifier-free uniformity in [14, 15] since it is easier to present and suffices for our purposes. (We point out that this subsumes, say, **L**-uniformity, as explained in the Remark below.) Recall that a Δ_0 formula is a first-order formula over $\{0, 1, +, \times, <\}$ where all quantifiers of the form $\exists x < t$ or $\forall x < t$ for a term t . A Δ_0 -formula $\varphi(n_1, \dots, n_k)$ is interpreted over \mathbb{N} in the usual way, and naturally computes the set $\{\vec{n} \in \mathbb{N}^k : \mathbb{N} \models \varphi(\vec{n})\}$.

► **Definition 3** (Positive circuits). A family of k -argument \neg -free circuits is a set $\{C(\vec{n})\}_{\vec{n} \in \mathbb{N}^k}$, where each $C(\vec{n})$ is a circuit with arbitrary fan-in \vee and \wedge gates,³ given as a tuple $(N, D, E, I_1, \dots, I_k, O)$, where $[N] = \{n < N\}$ is the set of gates, $D \subseteq [N]$ is the set of \vee gates (remaining gates are assumed to be \wedge), $E \subseteq [N] \times [N]$ is the set of (directed) edges (requiring $E(m, n) \implies m < n$), $I_j \subseteq [n_j] \times [N]$ contains just pairs (l, n) s.t. the l^{th} bit of the j^{th} input is connected to the gate n , and $O \subseteq [N]$ is the (ordered) set of output gates.

If these sets are polynomial-time computable from inputs $(1^{n_1}, \dots, 1^{n_k})$ then we say the circuit family is **P**-uniform. Similarly, we say the family is Δ_0 -uniform if $N(\vec{n})$ is a term (i.e. a polynomial) in \vec{n} and there are Δ_0 -formulae $D(n, \vec{n}), E(m, n, \vec{n}), I_j(l, n, \vec{n}), O(n, \vec{n})$ computing the associated sets.

The specification of a circuit family above is just a variant of the usual “direct connection language” from circuit complexity, cf. [22]. Notice that, importantly, we restrict the set O of output gates to depend only on the length of the inputs, not their individual bit-values; this is pertinent thanks to Prop. 2. Also, when it is convenient, we may construe I_j as a function $[n_j] \rightarrow \mathcal{P}([N])$, by Currying.

► **Remark.** Δ_0 -sets are well known to be complete for the linear-time hierarchy [24]. However, since we only need to manipulate “unary” inputs in the notion of Δ_0 -uniformity above, the circuits generated are actually **LH**-uniform, where **LH** is the logarithmic-time hierarchy, the uniform version of **AC**⁰ [2]. See, e.g., [5] Sect. 6.3 for related discussions on **LH** and, e.g., [7] Sect. IV.3 for some relationships between Δ_0 and **AC**⁰.

³ By convention, a \vee gate with zero inputs outputs 0, while a \wedge gate with zero inputs outputs 1.

► **Example 4** (Circuits for sorting). Let us write $th(j, x)$ for the $(j - 1)^{\text{th}}$ bit of $sort(x)$, for $1 \leq j \leq |x|$. We also set $th(0, x) = 1$ and $th(j, x) = 0$ for $j > |x|$. Notice that $th(j, x) = 1$ precisely if there are at least j 1s in x , i.e. it is a *threshold* function. We assume that the input j is given in unary, for monotonicity, but as an abuse of notation write, say, j rather than 1^j throughout this example to lighten the notation. (Later, in Sect. 5, we will be more formal when handling unary inputs.)

We have the following recurrence, for $j > 0$:

$$th(j, s_i x) = th(j, x) \vee (i \wedge th(j - 1, x)) \quad (2)$$

Notice that this recurrence treats the $i = 0$ and $i = 1$ cases in the “same way”. This corresponds to the notion of *uniformity* that we introduce in our function algebras later. We can use this recurrence to construct polynomial-size \neg -free circuits for sorting. For an input x of size n , write x^l for the prefix $x(l - 1) \cdots x(0)$. Informally, we construct a circuit with $n + 1$ “layers” (numbered $0, \dots, n$), where the l^{th} layer outputs $th(n, x^l) \cdots th(0, x^l)$; the layers are connected to each other according to the recurrence in (2), with $th(0, x^l)$ always set to 1. Each layer will thus have $2(n + 1)$ gates, with $(n + 1)$ disjunction gates (computing the functions $th(j, x^l)$), and $n + 1$ intermediate conjunction gates. We assign odd numbers to disjunction gates and even numbers to conjunction gates, so that the total number of gates is $N(n) = 2(n + 1)^2$ and $D(n) = \{2r + 1 : r < (n + 1)^2\}$. The sets $E(r, s, n)$ and $I(r, n)$ can be given a routine description, and the set $O(r, n)$ of output gates consists of just the final layer of disjunction gates (except the rightmost), computing $th(n, x) \cdots th(1, x)$, i.e. $O(r, n) = \{2(n + 1)^2 - 2r - 1 : r < n\}$. It is not hard to see that such circuits are not only \mathbf{P} -uniform, but also Δ_0 -uniform.

Now we introduce a machine model for uniform positive computation. The definition of a multitape machine below is essentially from [19]. The monotonicity criterion is identical to that from [14, 15], though we also allow auxiliary “work” tapes so that the model is easier to manipulate. This also means that we do not need explicit accepting and rejecting states with the further monotonicity requirements from [14, 15], since this is subsumed by the monotonicity requirement on writing 0s and 1s: predicates can be computed in the usual way by Boolean valued functions, with 0 indicating “reject” and 1 indicating “accept”.

► **Definition 5** (Positive machines). A k -tape (deterministic) *Turing machine* (TM) is a tuple $M = (Q, \Sigma, \delta, s, h)$ where:

- Q is a finite set of (non-final) *states*.
- $\Sigma \supseteq \{\triangleright, \square, 0, 1\}$ is a finite set, called the *alphabet*.
- $\delta : Q \times \Sigma^k \rightarrow (Q \cup \{h\}) \times (\Sigma \times \{\leftarrow, -, \rightarrow\})^k$ such that, whenever $\delta(q, \sigma_1, \dots, \sigma_k) = (q, \tau_1, d_1, \dots, \tau_k, d_k)$, if $\sigma_i = \triangleright$ then $\tau_i = \triangleright$ and $d_i = \rightarrow$.
- $s \in Q$ is the *initial state*.
- Q and Σ are disjoint, and neither contains the symbols $h, \leftarrow, -, \rightarrow$.

We call h the *final state*, \triangleright the “beginning of tape marker”, \square the “blank” symbol, and $\leftarrow, -, \rightarrow$ are the *directions* “left”, “stay” and “right”.

Now, write $\mathcal{I} = Q \times \Sigma^k$ and $\mathcal{O} = (Q \cup \{h\}) \times (\Sigma \times \{\leftarrow, -, \rightarrow\})^k$, so that δ is a function $\mathcal{I} \rightarrow \mathcal{O}$. We define partial orders $\leq_{\mathcal{I}}$ and $\leq_{\mathcal{O}}$ on \mathcal{I} and \mathcal{O} resp. as follows:

- $(q, \sigma_1, \dots, \sigma_k) \leq_{\mathcal{I}} (q', \sigma'_1, \dots, \sigma'_k)$ if $q = q'$ and, for $i = 1, \dots, k$, either $\sigma_i = \sigma'_i$, or both $\sigma_i = 0$ and $\sigma'_i = 1$.
- $(q, \sigma_1, d_1, \dots, \sigma_k, d_k) \leq_{\mathcal{O}} (q', \sigma'_1, d'_1, \dots, \sigma'_k, d'_k)$ if $q = q'$ and, for $i = 1, \dots, k$, we have $d_i = d'_i$ and either $\sigma_i = \sigma'_i$, or both $\sigma_i = 0$ and $\sigma'_i = 1$.

We say that M is *positive* (a PTM) if $\delta : \mathcal{I} \rightarrow \mathcal{O}$ is monotone with respect to $\leq_{\mathcal{I}}$ and $\leq_{\mathcal{O}}$, i.e. $I \leq_{\mathcal{I}} I' \implies \delta(I) \leq_{\mathcal{O}} \delta(I')$.

A *run* of input strings $x_1, \dots, x_k \in \{0, 1\}^*$ on M is defined in the usual way (see, e.g., [19]), beginning from the initial state s and initialising the i^{th} tape to $\triangleright x_i \square^\omega$, for $i = 1, \dots, k$. If M halts, i.e. reaches the state h , its *output* is whatever is printed on the k^{th} tape at that moment, up to the first \square symbol.

We say that a function $f : (\{0, 1\}^*)^k \rightarrow \{0, 1\}^*$ is *computable by a PTM* if there is a k' -tape PTM M , with $k' \geq k$, such that M halts on every input and, for inputs $(x_1, \dots, x_k, \varepsilon, \dots, \varepsilon)$, outputs $f(x_1, \dots, x_k)$.

The monotonicity condition on the transition function above means that the value of a Boolean read does not affect the next state or cursor movements (this reflects the “obliviousness” of monotone functions, cf. Prop. 2). Moreover, it may only affect the *Boolean* symbols printed: the machine may read 0 and print 0 but read 1 and print 1, in otherwise-the-same situation. However, if in one situation it prints a non-Boolean σ when reading a Boolean 0 or 1, it must also print σ when reading the other.

► **Example 6** (Machines for sorting). A simple algorithm for sorting a binary string x is as follows: do two passes of x , first copying the 0s in x onto a fresh tape, then appending the 1s.⁴ However, it is not hard to see that a machine directly implementing this algorithm will not be positive. Instead, we may again use the recurrence from (2).

We give an informal description of a PTM that sorts a binary string. The machine has four tapes; the first is read-only and stores the input, say x with $|x| = n$. As in Ex. 4, we inductively compute $t^l = th(n, x^l) \cdot \dots \cdot th(0, x^l)$, for $l \leq n$. The second and third tape are used to temporarily store t^l , while the fourth is used to compute the sorting of the next prefix t^{l+1} . At each step the cursors on the working tapes move to the next bit and the transition function implements the recurrence from (2), calculating the next bit of t^{l+1} and writing it to the fourth tape. Notice that the cursor on the third tape remains one position offset from the cursor on the second and fourth tapes, cf. (2). Once t^{l+1} has been completely written on the fourth tape the machine copies it over the contents of the second and third tapes and erases the fourth tape before moving onto the next bit of the first tape and repeating the process. Finally, once the first tape has been exhausted, the machine copies the contents of the second (or third) tape, except the last bit (corresponding to $th(x, 0) = 1$), onto the fourth tape and halts.

► **Theorem 7.** *The following function classes are equivalent:*

- (1) *Functions on $\{0, 1\}^*$ computable by Δ_0 -uniform families of \neg -free circuits.*
- (2) *Functions on $\{0, 1\}^*$ computable by multi-tape PTMs that halt in polynomial time.*
- (3) *Functions on $\{0, 1\}^*$ computable by \mathbf{P} -uniform families of \neg -free circuits.*

This result is similar to analogous ones found in [14] for positive versions of the predicate class \mathbf{P} . It uses standard techniques so we give only a sketch of the proof below. Notice that the equivalence of models thus holds for any level of uniformity between Δ_0 and \mathbf{P} , e.g. for \mathbf{L} -uniform \neg -free circuits, cf. the Remark on p. 3.

Proof sketch of Thm. 7. We show that (1) \subseteq (2) \subseteq (3) \subseteq (1). The containments are mostly routine, though (3) \subseteq (1) requires some subtlety due to the positivity condition on circuits. For this we rely on an observation from [10]. Let $C(\vec{n})$ be a \mathbf{P} -uniform family of \neg -free

⁴ Recall that, while bits are indexed from right to left, machines read from left to right.

circuits, specified by polynomial-time programs $N, D, E, I_1, \dots, I_k, O$. Since the circuit-value problem is \mathbf{P} -complete under even \mathbf{AC}^0 -reductions (see, e.g., [7]), we may recover Δ_0 -uniform polynomial-size circuits (with negation) computing each of $N, D, E, I_1, \dots, I_k, O$, cf. the Remark on p. 3. However, these circuits take only unary strings of 1s as inputs, and so all negations can be pushed to the bottom (by De Morgan laws) and eliminated, yielding input-free \neg -free circuits for each of $N, D, E, I_1, \dots, I_k, O$ and their complements (by dualising gates). We may use these as “subcircuits” to compute the relevant local properties of $C(\vec{n})$. In particular, every internal gate n of $C(\vec{n})$ may be replaced by the following configuration (progressively, beginning from the highest-numbered gate $N(\vec{n}) - 1$):

$$\vee \left(\begin{array}{l} D(n, \vec{n}) \wedge \left(\bigvee_{m < n} (m \wedge E(m, n, \vec{n})) \vee \bigvee_{j=1}^k \bigvee_{l < n_j} (x(l) \wedge I_j(l, n)) \right) \\ \neg D(n, \vec{n}) \wedge \left(\bigwedge_{m < n} (m \vee \neg E(m, n, \vec{n})) \wedge \bigwedge_{j=1}^k \bigwedge_{l < n_j} (x(l) \vee \neg I_j(l, n)) \right) \end{array} \right)$$

This entire construction can be made Δ_0 -uniform, upon a suitable renumbering of gates.

The proof of (2) \subseteq (3) follows a standard construction (see, e.g., [19]), observing that the positivity criterion on PTMs entails local monotonicity and hence allows us to construct circuits that are \neg -free. (Similar observations are made in [11, 10, 14, 15]). Suppose Q, Σ and $\{\leftarrow, -, \rightarrow\}$ are encoded by Boolean strings such that distinct elements are incomparable under \leq , (except $0 \leq 1$ for $0, 1 \in \Sigma$). Thus we may construe δ as a bona fide monotone Boolean function of fixed input arities, and thus has some (constant-size) \neg -free circuit thanks to adequacy of the basis $\{\vee, \wedge\}$, say C_δ . Now, on a fixed input, consider “configurations” of the form $(q, x_1, n_1, \dots, x_k, n_k)$, where $q \in Q$, x_i is the content of the i^{th} tape (up to the halting time bound) and n_i is the associated cursor position (encoded in unary). We may use C_δ to construct polynomial-size \neg -free circuits mapping the machine configuration at time t to the configuration at time $t + 1$. By chaining these circuits together polynomially many times (determined by the halting time bound), we may thus obtain a circuit that returns the output of the PTM. This entire construction remains \mathbf{P} -uniform, as usual.

The proof of (1) \subseteq (2) is also routine, building a PTM “evaluator” for \neg -free circuits, where \neg -freeness allows us to satisfy the positivity condition on TMs. We rely on the fact that the Δ_0 -specifications may be entirely encoded in *unary* on a PTM, so that they are monotone, in polynomial-time. We do not go into details here since, in particular, this containment is subsumed by our later results, Thm. 17 and Thm. 30, which show that (1) $\subseteq u\mathbf{C} \subseteq$ (2), for the algebra $u\mathbf{C}$ we introduce in the next section. \blacktriangleleft

► **Definition 8** (Positive \mathbf{FP}). The function class \mathbf{posFP} is defined to be the set of functions on $\{0, 1\}^*$ computed by any of the equivalent models from Thm. 7.

► **Remark.** The notion of *positive* computation was previously studied in [11, 14, 15]. One interesting point already noted in those works is that, for a complexity class, its positive version is not, in general, just its monotone members. This follows from a seminal result of Razborov [20], and later improvements [1, 23]: there are polynomial-time monotone predicates (and hence polynomial-size circuits with negation) for which the only \neg -free circuits are exponential in size. In particular, $\mathbf{posFP} \subsetneq \{f \in \mathbf{FP} : f \text{ monotone}\}$.

3 An algebra $u\mathbf{C}$ for \mathbf{posFP}

We present a *function algebra* for \mathbf{posFP} by considering “uniform” versions of recursion operators. We write $[\mathcal{F}; \mathcal{O}]$ for the function class generated by a set of initial functions \mathcal{F} and a set of operations \mathcal{O} , and generally follow conventions and notations from [5].

Let us first recall Cobham's function algebra for the polynomial-time functions, **FP**. This algebra was originally formulated over natural numbers, though we work with a version here over binary words, essentially as in [9, 18].

Define $\pi_j^k(x_1, \dots, x_k) := x_j$ and $x\#y := 1^{|x||y|}$. We write **comp** for the operation of function composition.

► **Definition 9.** A function f is defined by *bounded recursion on notation* (BRN) from g, h_0, h_1, k if $|f(x, \vec{x})| \leq |k(x, \vec{x})|$ for all x, \vec{x} and:

$$\begin{aligned} f(\varepsilon, \vec{x}) &= g(\vec{x}) \\ f(s_0x, \vec{x}) &= h_0(x, \vec{x}, f(x, \vec{x})) \\ f(s_1x, \vec{x}) &= h_1(x, \vec{x}, f(x, \vec{x})) \end{aligned} \quad (3)$$

We write **C** for the function algebra $[\varepsilon, s_0, s_1, \pi_j^k, \#, \text{comp}, \text{BRN}]$.

► **Theorem 10** ([6]). $\mathbf{C} = \mathbf{FP}$.

Notice that $\varepsilon, s_0, s_1, \pi_j^k, \#$ are monotone, and the composition of two monotone functions is again monotone. However, non-monotone functions are definable using BRN, for instance:

$$\begin{aligned} \text{cond}(\varepsilon, y_\varepsilon, y_0, y_1) &= y_\varepsilon \\ \text{cond}(s_0x, y_\varepsilon, y_0, y_1) &= y_0 \\ \text{cond}(s_1x, y_\varepsilon, y_0, y_1) &= y_1 \end{aligned} \quad (4)$$

This ‘‘conditional’’ function is definable since we do not force any connection between h_0 and h_1 in (3). Insisting on $h_0 \leq h_1$ would retain monotonicity, but this condition is external and not generally checkable. Instead, we can impose monotonicity implicitly by somewhat ‘‘uniformising’’ BRN. First, we will need to recover certain monotone variants of the conditional:

► **Definition 11** (Meets and joins). We define $x \wedge y = z$ by $|z| = \min(|x|, |y|)$ and $z(j) = \min(x(j), y(j))$, for $j < \min(|x|, |y|)$. We define analogously $x \vee y = z$ by $|z| = \max(|x|, |y|)$ and $z(j) = \max(x(j), y(j))$, for $j < \max(|x|, |y|)$.

Note that, in the case of $x \vee y$ above, if $|x| < |y|$ and $|x| \leq j < \max(|x|, |y|)$, then $x(j)$ is not defined and we set $z(j) = y(j)$. We follow an analogous convention when $|y| < |x|$.

► **Definition 12** (The function algebra $u\mathbf{C}$). We say that a function is defined by *uniform bounded recursion on notation* ($u\text{BRN}$) from g, h, k if $|f(x, \vec{x})| \leq |k(x, \vec{x})|$ for all x, \vec{x} and:

$$\begin{aligned} f(\varepsilon, \vec{x}) &= g(\vec{x}) \\ f(s_0x, \vec{x}) &= h(0, x, \vec{x}, f(x, \vec{x})) \\ f(s_1x, \vec{x}) &= h(1, x, \vec{x}, f(x, \vec{x})) \end{aligned} \quad (5)$$

We define $u\mathbf{C}$ to be the function algebra $[\varepsilon, s_0, s_1, \pi_j^k, \#, \wedge, \vee; \text{comp}, u\text{BRN}]$.

Notice that \wedge and \vee are clearly **FP** functions, therefore they are in **C**. Moreover, notice that (5) is the special case of (3) when $h_i(x, \vec{x}, y)$ has the form $h(i, x, \vec{x}, y)$. So, we have that $u\mathbf{C} \subseteq \mathbf{C} = \mathbf{FP}$. We will implicitly use this observation later to ensure that the outputs of $u\mathbf{C}$ functions have lengths which are polynomially bounded on the lengths of the inputs.

The main result of this work is that $u\mathbf{C} = \mathbf{posFP}$. The two directions of the equality are proved in the sections that follow, in the form of Thms. 17 and 30. Before that, we make some initial observations about $u\mathbf{C}$.

► **Proposition 13.** uC contains only monotone functions.

Proof. The proof is by induction on the definition of f . The relevant case is when f is defined by $uBRN$. It suffices to show that f is monotone in its first input, which we do by induction on its length. Let $w \leq x$. If $|w| = |x| = 0$, then they are both ε and we are done. Otherwise let $w = s_i w'$ and $x = s_j x'$. Then $f(w, \vec{y}) = h(i, w', \vec{y}, f(w', \vec{y})) \leq h(j, x', \vec{y}, f(x', \vec{y}))$ by the inductive hypothesis, since $i \leq j$ and $w' \leq x'$, and we are done. ◀

► **Proposition 14.** $uC + cond = C$.⁵

Proof. The left-right inclusion follows from the definition of $cond$ by BRN in (4). For the right-left inclusion, we again proceed by induction on the definition of functions in C , and the relevant case is when f is defined by BRN , say from g, h_0, h_1, k . In this case, we may recover a definition of f using $uBRN$ by writing $h(i, x, \vec{x}, y) = cond(i, g(\vec{x}), h_0(x, \vec{x}, y), h_1(x, \vec{x}, y))$. ◀

As expected, uC contains the usual predecessor function, least significant parts, concatenation, and a form of iterated predecessor:

► **Proposition 15** (Basic functions in uC). uC contains the following functions:⁶

$$\begin{array}{llll} p(\varepsilon) := \varepsilon & lsp(\varepsilon) := \varepsilon & x \cdot \varepsilon := x & msp(|\varepsilon|, y) := y \\ p(s_i x) := x & lsp(s_i x) := i & x \cdot (s_i y) := s_i(x \cdot y) & msp(|s_i x|, y) := p(msp(|x|, y)) \end{array}$$

Proof. All these definitions are instances of $uBRN$, with bounding function $\#(s_1 x, s_1 y)$. ◀

Notice that, in the above definition of concatenation and throughout this work, we write $s_i x$ for $s_0 x \vee i$. We also sometimes simply write xy instead of $x \cdot y$.

We may also extract individual bits and test for the empty string in:

► **Proposition 16** (Bits and tests). uC contains the following functions:

$$\begin{array}{ll} bit(|x|, y) := lsp(msp(|x|, y)) & cond_\varepsilon(\varepsilon, y, z) := y \\ & cond_\varepsilon(s_i x, y, z) := z \end{array}$$

4 posFP contains uC

One direction of our main result follows by standard techniques:

► **Theorem 17.** $uC \subseteq \text{posFP}$.

It is not hard to see that one can extract (uniform) \neg -free circuits from a uC program, but we instead give a PTM for each function of uC .

Proof sketch of Thm. 17. The proof is by induction on the function definitions. We prove that for all $f \in uC$ there exists a PTM M_f computing f in polynomial time. For the initial functions the result is straightforward, and composition is routine.

We give the important case of when f is defined by $uBRN$ from functions $g, h, k \in uC$, as in (5); we will assume there are no side variables \vec{x} , for simplicity, though the general case is similar. Let $|f(x)| \leq b(|x|)$ for some polynomial $b(n)$ (since, in particular, $f(x) \in C = \mathbf{FP}$). By the inductive hypothesis, there are PTMs M_g (with t tapes) and M_h (with $3 + u$ tapes)

⁵ Here we write $[\mathcal{F}; \mathcal{O}] + f$ for the function algebra $[\mathcal{F}, f; \mathcal{O}]$.

⁶ Notice that we could have equivalently defined $lsp(x)$ as $x \wedge 1$.

computing, respectively, g and h in time bounded by $p_g(n)$ and $p_h(1, m, n)$ for inputs of lengths n and $(1, m, n)$, respectively, for appropriate polynomials p_g, p_h . We assume that M_g and M_h halt scanning the first cell of each tape. In case of M_h we also assume that the content of tapes 1 and 2 are not changed during the computation (i.e. are read-only), and that the machine halts with the output in tape 3 with the other u tapes empty. We may define an auxiliary machine, M , with 3 tapes. Whenever the recursion input x is on tape 1, every time we run M , it writes the two first inputs of a call to h on tapes 2 and 3 and shifts the cursor in x one bit along. This means that a bit of x will be on tape 2 and a prefix of x , up to that bit, will be on tape 3.

Such M may be constructed so that it is a positive TM which works in time bounded by $2|x| + 1$.

Now, we describe a positive TM M_f (with $3 + u + t$ tapes) computing f as follows:

1. Run M_g (over the last t tapes of M_f);
2. Enter state s , run M (over tapes 1-3), and if M reaches state H , halt;
3. Run M_h (over tapes 2,3, $3 + u + t$, and tapes 4 to $u + 3$ of M_f , in this order);
4. Go to (2).

Each run of M shifts the cursor of the input tape one cell to the right, so, as expected, it halts after $|x|$ repetitions of the loop above, and hence operates in polynomial time. ◀

5 Some properties of the algebra $u\mathcal{C}$

We conduct some “bootstrapping” in the algebra $u\mathcal{C}$, both for self-contained interest and also for use later on to prove the converse of Thm. 17 in Sect. 6.

5.1 An algebra for lengths: tally functions of $u\mathcal{C}$ and linear space

We characterise the *tally functions* of $u\mathcal{C}$, i.e. those with unary inputs and outputs, as just the unary codings of functions on \mathbb{N} computable in linear space. We carry this argument out in a recursion-theoretic setting so that the exposition is more self-contained.

To distinguish functions on \mathbb{N} from functions on $\{0, 1\}^*$, we use variables m, n etc. to vary over \mathbb{N} . We will also henceforth write \underline{n} for 1^n , to lighten the presentation when switching between natural numbers and binary words.

Further to Prop. 2, for functions in $u\mathcal{C}$ we may actually compute output lengths in a simple function algebra over \mathbb{N} .

► **Definition 18.** Let $0, 1, +, \times, \min, \max$ have their usual interpretations over \mathbb{N} . $f(n, \vec{n})$ is defined by *bounded recursion*, written **BR**, from g, h, k if $f(n, \vec{n}) \leq k(n, \vec{n})$ for all n, \vec{n} and:

$$\begin{aligned} f(0, \vec{n}) &= g(\vec{n}) \\ f(n+1, \vec{n}) &= h(n, \vec{n}, f(n, \vec{n})) \end{aligned}$$

We write \mathcal{E}^2 for the function algebra $[0, 1, +, \times, \min, \max, \pi_j^k; \text{comp}, \text{BR}]$ over \mathbb{N} .

Let us write **FLINSPACE** for the class of functions on \mathbb{N} computable in linear space (see, e.g., [5]). The following result is well-known:

► **Proposition 19** ([21]). $\mathcal{E}^2 = \text{FLINSPACE}$.

For a list of arguments $\vec{x} = (x_1, \dots, x_k)$, let us write $|\vec{x}|$ for $(|x_1|, \dots, |x_k|)$.

► **Lemma 20.** For $f(\vec{x}) \in u\mathcal{C}$, there is a $l_f(\vec{n}) \in \mathcal{E}^2$ such that $|f(\vec{x})| = l_f(|\vec{x}|)$.

Proof. We proceed by induction on the definition of f in $u\mathcal{C}$. For the initial functions we have: $|\varepsilon| = 0$, $|s_0x| = |x| + 1$, $|s_1x| = |x| + 1$, $|x\#y| = |x||y|$, $|\pi_j^k(x_1, \dots, x_n)| = |x_j|$, $|x \wedge y| = \min(|x|, |y|)$, and $|x \vee y| = \max(|x|, |y|)$.

If f is defined by composition, the result is immediate from composition in \mathcal{E}^2 . Finally, if $f(x, \vec{x})$ is defined by $u\text{BRN}$ from functions $g, h, k \in u\mathcal{C}$, as in (5), then we have,

$$\begin{aligned} |f(\varepsilon, \vec{x})| &= |g(\vec{x})| \\ |f(s_i x, \vec{x})| &= |h(1, x, \vec{x}, f(x, \vec{x}))| \end{aligned}$$

and we may define l_f by BR from l_g, l_h and l_k , by the inductive hypothesis. \blacktriangleleft

By appealing to the lengths of $\varepsilon, s_1, \cdot, \#, \wedge, \vee, u\text{BRN}$, we also have a converse result to Lemma 20 above, giving the following characterisation of the tally functions of $u\mathcal{C}$:

► **Theorem 21.** *Let $f : \mathbb{N}^k \rightarrow \mathbb{N}$. Then the binary string function $f(|\vec{x}|)$ is in $u\mathcal{C}$ if and only if the natural number function $f(\vec{n})$ is in \mathcal{E}^2 .*

Proof sketch. The left-right implication follows from Lemma 20 above, and the right-left implication follows by simulating \mathcal{E}^2 -definitions with unary codings in $u\mathcal{C}$. \blacktriangleleft

Thanks to this result, we will rather work in \mathcal{E}^2 when reasoning about tally functions in $u\mathcal{C}$, relying on known facts about **FLINSPACE** (see, e.g., [5]).

In $u\mathcal{C}$, we may also use unary codings to “iterate” other functions. We write $f(\vec{n}, \vec{y}) \in u\mathcal{C}$ if there is $f'(\vec{x}, \vec{y}) \in u\mathcal{C}$ such that $f'(\vec{n}, \vec{y}) = f(\vec{n}, \vec{y})$, for all $\vec{n} \in \mathbb{N}$.

► **Observation 22 (Length iteration).** *$u\mathcal{C}$ is closed under the bounded length iteration operation: we may define $f(\underline{n}, \vec{x})$ from $g(\vec{x})$, $h(\underline{n}, \vec{x}, y)$ and $k(\underline{n}, \vec{x})$ as:*

$$\begin{aligned} f(\underline{0}, \vec{x}) &:= g(\vec{x}) \\ f(\underline{n+1}, \vec{x}) &:= h(\underline{n}, \vec{x}, f(\underline{n}, \vec{x})) \end{aligned}$$

as long as $|f(\underline{n}, \vec{x})| \leq |k(\underline{n}, \vec{x})|$.

In fact, bounded length iteration is just a special case of $u\text{BRN}$, and we will implicitly use this when iterating functions by length. This is crucial for deriving closure properties of $u\mathcal{C}$, as in the next subsection, and for showing that $u\mathcal{C} \supseteq \text{posFP}$ in Sect. 6.

► **Remark (Some iterated functions).** For $h(x, \vec{x}) \in u\mathcal{C}$, the following functions are in $u\mathcal{C}$:

$$\begin{aligned} \bigvee_{j < |x|} h(j, \vec{x}) &:= h(|x| - 1, \vec{x}) \vee \dots \vee h(0, \vec{x}) & \bigvee x &:= \bigvee_{j < |x|} \text{bit}(j, x) \\ \bigwedge_{j < |x|} h(j, \vec{x}) &:= h(|x| - 1, \vec{x}) \wedge \dots \wedge h(0, \vec{x}) & \bigwedge x &:= \bigwedge_{j < |x|} \text{bit}(j, x) \\ \bigodot_{j < |x|} h(j, \vec{x}) &:= h(|x| - 1, \vec{x}) \cdot \dots \cdot h(0, \vec{x}) \end{aligned}$$

Notice that, as for the definitions of $\bigvee x$ and $\bigwedge x$ above, we may use iterated operators with various limit formats, implicitly assuming that these are definable in $u\mathcal{C}$.

► **Example 23 (A program for sorting).** Notice that the recurrence in (1), while an instance of BRN , is not an instance of $u\text{BRN}$, since it is not uniform. However, we may give a positive definition by $u\text{BRN}$ based, once again, on the recurrence (2):

$$\begin{aligned} \text{sort}(\varepsilon) &= \varepsilon \\ \text{sort}(s_i x) &= \bigodot_{j < |x|} (\text{bit}(j + 1, s_1 \text{sort}(x)) \vee (i \wedge \text{bit}(j, s_1 \text{sort}(x)))) \end{aligned}$$

5.2 $u\mathbf{C}$ is closed under simultaneous $u\mathbf{BRN}$

To exemplify the robustness of the algebra $u\mathbf{C}$ it is natural to show closure under certain variants of recursion. While we do not explicitly use these results later, the technique should exemplify how other textbook-style results may be obtained for $u\mathbf{C}$. We also point out that the ideas herein are implicitly used in Sect. 6 where we inline a treatment of a restricted version of “course-of-values” recursion.

One of the difficulties in reasoning about $u\mathbf{C}$ is that it is not clear how to define appropriate (monotone) (de)pairing functions, which are usually necessary for such results. Instead, we rely on analogous results for \mathcal{E}^2 , before “lifting” them to $u\mathbf{C}$, thanks to Thm. 21 and Prop. 2. We give a self-contained exposition for the benefit of the reader but, since **FLINSPACE** and algebras like \mathcal{E}^2 are well known, we will proceed swiftly; see, e.g., [5] for more details.

Notice that we have the following functions in \mathcal{E}^2 ,

$$n \dot{-} m := \max(n - m, 0) \quad \text{and} \quad \text{cond}_0(x, y, z) := \begin{cases} y & \text{if } x = 0 \\ z & \text{otherwise} \end{cases}$$

thanks to Thm. 21 and the fact that $\text{msp}(|x|, y)$ and cond_ε are in $u\mathbf{C}$. Thus we may define,

$$\text{le}(m, n) := \begin{cases} 0 & \text{if } n \dot{-} m = 0 \\ 1 & \text{otherwise} \end{cases} \quad \text{and} \quad \left\lfloor \frac{n}{2} \right\rfloor := \sum_{i < n} \text{le}(2i + 1, n)$$

by bounded recursion. This allows us to define in \mathcal{E}^2 a simple pairing function:

► **Proposition 24** (Pairing in \mathcal{E}^2). *The following function is in \mathcal{E}^2 :*

$$\langle n_0, n_1 \rangle := \left\lfloor \frac{(n_0 + n_1)(n_0 + n_1 + 1)}{2} \right\rfloor + n_0$$

We now show that we have the analogous *depairing* functions, due to the fact that bounded minimisation is available in **FLINSPACE**.

► **Lemma 25** (Bounded minimisation, [12]). \mathcal{E}^2 is closed under bounded minimisation: if $f(n, \vec{n}) \in \mathcal{E}^2$ then so is the following function:

$$\mathfrak{s}(\mu m < n).(f(m, \vec{n}) = 0) := \begin{cases} m + 1 & m < n \text{ is least s.t. } f(m, \vec{n}) = 0 \\ 0 & f(m, \vec{n}) > 0 \text{ for all } m < n \end{cases}$$

Proof. Appealing to BR, we have $\mathfrak{s}(\mu m < 0).(f(m, \vec{n}) = 0) = 0$ and,

$$\begin{aligned} & \mathfrak{s}(\mu m < n + 1).(f(m, \vec{n}) = 0) \\ = & \begin{cases} n + 1 & \text{if } \mathfrak{s}(\mu m < n).(f(m, \vec{n}) = 0) = 0, f(n, \vec{n}) = 0 \\ 0 & \text{if } \mathfrak{s}(\mu m < n).(f(m, \vec{n}) = 0) = 0, f(n, \vec{n}) \neq 0 \\ \mathfrak{s}(\mu m < n).(f(m, \vec{n}) = 0) & \text{if } \mathfrak{s}(\mu m < n).(f(m, \vec{n}) = 0) \neq 0 \end{cases} \end{aligned}$$

by two applications of the conditional cond_0 . ◀

► **Proposition 26** (Depairing). *For $i \in \{0, 1\}$, the function β_i with $\beta_i(\langle n_0, n_1 \rangle) = n_i$ is in \mathcal{E}^2 .*

Proof. We have $\beta_0(n) = \mathfrak{s}(\mu n_0 < n).(\mathfrak{s}(\mu n_1 < n).(\langle n_0, n_1 \rangle = n) \neq 0) \dot{-} 1$, which is definable by bounded minimisation and appropriate conditionals.⁷ $\beta_1(n)$ is defined analogously, by switching $\mathfrak{s}(\mu n_0 < n)$ and $\mathfrak{s}(\mu n_1 < n)$. ◀

⁷ Notice that $\langle n_0, n_1 \rangle = n$ iff $\max(\langle n_0, n_1 \rangle \dot{-} n, n \dot{-} \langle n_0, n_1 \rangle) = 0$.

Thanks to (de)pairing, we have the following (well-known) result:

► **Proposition 27.** \mathcal{E}^2 is closed under simultaneous bounded recursion: we may define f_1, \dots, f_p from $g_1, h_1, k_1, \dots, g_p, h_p, k_p$ if $f_j(n, \vec{n}) \leq k_j(n, \vec{n})$ for all n, \vec{n} , for $1 \leq j \leq p$, and:

$$\begin{aligned} f_j(0, \vec{n}) &= g_j(\vec{n}) \\ f_j(n+1, \vec{n}) &= h_j(n, \vec{n}, f_1(n, \vec{n}), \dots, f_p(n, \vec{n})) \end{aligned}$$

This result, along with Lemma 20, allows us to show that $u\mathbf{C}$ is closed under the simultaneous form of $u\mathbf{BRN}$, by using *concatenation* instead of pairing:

► **Theorem 28.** $u\mathbf{C}$ is closed under simultaneous $u\mathbf{BRN}$: we may define f_1, \dots, f_p from $g_1, h_1, k_1, \dots, g_p, h_p, k_p$ if $|f_j(x, \vec{x})| \leq |k_j(x, \vec{x})|$ for all x, \vec{x} , for $1 \leq j \leq p$, and:

$$\begin{aligned} f_j(\varepsilon, \vec{x}) &= g_j(\vec{x}) \\ f_j(s_i x, \vec{x}) &= h_j(i, x, \vec{x}, f_1(x, \vec{x}), \dots, f_p(x, \vec{x})) \end{aligned}$$

Proof sketch. For $1 \leq j \leq p$, we have g_j, h_j, k_j are in $u\mathbf{C}$, therefore by Lemma 20 there exist, in \mathcal{E}^2 , functions l_{g_j}, l_{h_j} and l_{k_j} computing their output lengths in terms of their input lengths. Appealing to simultaneous bounded recursion (Prop. 27), we may define in the natural way functions $l_{f_j} \in \mathcal{E}^2$ such that $|f_j(x, \vec{x})| = l_{f_j}(|x|, |\vec{x}|)$ for all x, \vec{x} .

Now, using concatenation, we define the following function in $u\mathbf{C}$ by $u\mathbf{BRN}$,

$$\begin{aligned} F(\varepsilon, \vec{x}) &= g_1(\vec{x}) \cdot \dots \cdot g_p(\vec{x}) \\ F(s_i x, \vec{x}) &= h_1(i, x, \vec{x}, \vec{F}(x, \vec{x})) \cdot \dots \cdot h_p(i, x, \vec{x}, \vec{F}(x, \vec{x})), \end{aligned}$$

where $\vec{F} = (F_1, \dots, F_p)$ and each $F_j(x, \vec{x})$ is $F(x, \vec{x})$ without its leftmost $l_{f_1}(|x|, |\vec{x}|) + \dots + l_{f_{j-1}}(|x|, |\vec{x}|)$ and its rightmost $l_{f_{j+1}}(|x|, |\vec{x}|) + \dots + l_{f_p}(|x|, |\vec{x}|)$ bits, i.e.,

$$F_j(x, \vec{x}) = \text{msp}(l_{f_{j+1}}(|x|, |\vec{x}|) + \dots + l_{f_p}(|x|, |\vec{x}|), F(x, \vec{x})) \wedge l_{f_j}(|x|, |\vec{x}|)$$

The bounding function is just the concatenation of all the $k_j(x, \vec{x})$, for $1 \leq j \leq p$. Now we may conclude by noticing that $f_j(x, \vec{x}) = F_j(x, \vec{x})$, for $1 \leq j \leq p$. ◀

6 $u\mathbf{C}$ contains posFP

We are now ready to present our proof of the converse to Thm. 17. For this we appeal to the characterisation (1) from Thm. 7 of **posFP** as Δ_0 -uniform families of \neg -free circuits. Since Δ_0 formulae compute just the predicates of the linear-time hierarchy, the following result is not surprising, though we include it for completeness of the exposition:

► **Lemma 29** (Characteristic functions of Δ_0 sets). *Let φ be a Δ_0 -formula with free variables amongst \vec{n} . There is a function $f_\varphi(\vec{n}) \in \mathcal{E}^2$ such that:*

$$f_\varphi(\vec{n}) = \begin{cases} 0 & \mathbb{N} \not\models \varphi(\vec{n}) \\ 1 & \mathbb{N} \models \varphi(\vec{n}) \end{cases}$$

Proof. We already have functions for all terms (written s, t , etc.), i.e. polynomials, due to the definition of \mathcal{E}^2 . We proceed by induction on the structure of φ , which we assume by De Morgan duality is written over the logical basis $\{\neg, \wedge, \forall\}$:

■ For atomic formulae we use the length conditional to define appropriate functions:

$$f_{s < t}(\vec{n}) := \begin{cases} 1 & s \dot{-} (t + 1) = 0 \\ 0 & \text{otherwise} \end{cases} \quad f_{s = t}(\vec{n}) := \begin{cases} 1 & \max(s \dot{-} t, t \dot{-} s) = 0 \\ 0 & \text{otherwise} \end{cases}$$

- If φ is $\neg\psi$ then we define f_φ , using the conditional, as follows:

$$f_\varphi(\vec{n}) := \begin{cases} 1 & f_\psi(\vec{n}) = 0 \\ 0 & \text{otherwise} \end{cases}$$

- If φ is $\psi \wedge \chi$ then we define f_φ as follows:

$$f_\varphi(\vec{n}) := \min(f_\psi(\vec{n}), f_\chi(\vec{n}))$$

- If φ is $\forall n < t. \psi(n, \vec{n})$ then we define $f_\varphi(t, \vec{n})$, by BR, as follows:

$$\begin{aligned} f_\varphi(0, \vec{n}) &:= 1 \\ f_\varphi(n+1, \vec{n}) &:= \min(f_\psi(n, \vec{n}), f_\varphi(n, \vec{n})) \end{aligned}$$

Using this result, we may argue for the converse of Thm. 17.

► **Theorem 30.** $\text{posFP} \subseteq u\mathcal{C}$.

Proof. Working with the characterisation (1) from Thm. 7 of **posFP**, we use Lemma 29 above to recover characteristic functions of sets specifying a \neg -free circuit family $C(\vec{n})$ in \mathcal{E}^2 . Writing $N, D, E, I_1, \dots, I_k, O$ for the associated characteristic functions (in \mathcal{E}^2), we define an “evaluator” program in $u\mathcal{C}$, taking advantage of Thm. 21, that progressively evaluates the circuit as follows. Given inputs \vec{x} of lengths \vec{n} , we will define a function $Val(\underline{n}, \vec{x})$ that returns the concatenation of the outputs of the gates $< n$ in $C(\vec{n})$, by length iteration, cf. Obs. 22.

The base case of the iteration is simple, with $Val(\underline{0}, \vec{x}) := \varepsilon$. For the inductive step we need to set up some intermediate functions. Suppressing the parameters \vec{n} , we define the function $\iota(\underline{n}, \vec{x})$ returning the concatenation of input bits sent to the n^{th} gate:

$$\iota(\underline{n}, \vec{x}) := \bigodot_{m < |x_1|} \left(\underline{I_1}(m, n) \wedge \text{bit}(\underline{m}, x_1) \right) \cdot \dots \cdot \bigodot_{m < |x_k|} \left(\underline{I_k}(m, n) \wedge \text{bit}(\underline{m}, x_k) \right)$$

Now we define the value $val(\underline{n}, \vec{x})$ of the n^{th} gate in terms of $Val(\underline{n}, \vec{x})$, appealing again to the iterated operators from Rmk. 5.1, and testing for the empty string:⁸

$$val(\underline{n}, \vec{x}) := \begin{cases} \bigwedge \iota(\underline{n}, \vec{x}) \wedge \bigwedge_{m < n} \left(\underline{(1 \div E)}(m, n) \vee \text{bit}(\underline{m}, Val(\underline{n}, \vec{x})) \right) & \text{if } \underline{D}(n) = \underline{0} \\ \bigvee \iota(\underline{n}, \vec{x}) \vee \bigvee_{m < n} \left(\underline{E}(m, n) \wedge \text{bit}(\underline{m}, Val(\underline{n}, \vec{x})) \right) & \text{if } \underline{D}(n) = \underline{1} \end{cases}$$

Finally we may define $Val(\underline{n+1}, \vec{x}) := val(\underline{n}, \vec{x}) \cdot Val(\underline{n}, \vec{x})$. At this point we may define the output $C(\vec{x})$ of the circuit as $\bigodot_{m < N} \left(\underline{O}(m) \wedge \text{bit}(\underline{m}, Val(\underline{N}, \vec{x})) \right)$. ◀

7 A characterisation based on safe recursion

In [3] Bellantoni and Cook give an *implicit* function algebra for **FP**, not mentioning any explicit bounds, following seminal work by Leivant, [16, 17], who first gave a *logical* implicit characterisation of **FP**. In this section we give another function algebra for **posFP** in the style of Bellantoni and Cook’s, using “safe recursion”. Our argument follows closely the structure of the original argument in [3]; it is necessary only to verify that those results go through once an appropriate uniformity constraint is imposed. We write normal-safe functions as usual: $f(\vec{x}; \vec{y})$ where \vec{x} are the *normal* inputs and \vec{y} are the *safe* inputs.

⁸ Formally, here we follow the usual convention that $\bigvee \varepsilon = 0$ and $\bigwedge \varepsilon = 1$.

► **Definition 31** (Function algebra $u\mathbf{B}$). We say that f is defined by *safe composition*, written *scomp*, from functions g, \vec{r}, \vec{s} if: $f(\vec{x}; \vec{y}) = g(\vec{r}(\vec{x};); \vec{s}(\vec{x}; \vec{y}))$. We say that f is defined by *uniform safe recursion on notation* ($u\mathbf{SRN}$) from functions g and h if:

$$\begin{aligned} f(\varepsilon, \vec{x}; \vec{y}) &= g(\varepsilon, \vec{x}; \vec{y}) \\ f(\mathbf{s}_i x, \vec{x}; \vec{y}) &= h(x, \vec{x}; i, \vec{y}, f(x, \vec{x}; \vec{y})) \end{aligned}$$

We define $u\mathbf{B} := [\varepsilon, \mathbf{s}_0^1, \mathbf{s}_1^1, \pi_j^{1;k}, \wedge^2, \vee^2, p^1, \text{cond}_\varepsilon^3; \text{scomp}, u\mathbf{SRN}]$. Here, superscripts indicate the arity of the function, which we often omit. We will show that the normal part of $u\mathbf{B}$ computes precisely \mathbf{posFP} , following the same argument structure as [3].

► **Lemma 32** (Bounding lemma). *For all $f \in u\mathbf{B}$, there is a polynomial $b_f(\vec{m}, \vec{n})$ (with natural coefficients) such that, for all \vec{x}, \vec{y} , $|f(\vec{x}; \vec{y})| \leq b_f(|\vec{x}|, |\vec{y}|)$.*

Proof idea. We show by that for $f \in u\mathbf{B}$, by induction on its definition, there exists a polynomial $q_f(\vec{n})$ such that, for all \vec{x}, \vec{y} , $|f(\vec{x}; \vec{y})| \leq q_f(|\vec{x}|) + \max_j(|y_j|)$. (This is just a special case of the same property for \mathbf{B} from [3].) ◀

► **Proposition 33.** *If $f(\vec{x}; \vec{y}) \in u\mathbf{B}$, then we have $f(\vec{x}, \vec{y}) \in u\mathbf{C}$.*

Proof sketch. We proceed by induction on the definition of f ; the only interesting case is when f is defined by $u\mathbf{SRN}$. In this case we define f analogously to $u\mathbf{BRN}$, taking the bounding function to be $b_f(|\vec{x}|, |\vec{y}|)$, where b_f is obtained from Lemma 32 above. ◀

Therefore we have that $u\mathbf{B}$ is contained in $u\mathbf{C}$, and consequently in \mathbf{posFP} . In order to establish the other inclusion we slightly reformulate the function algebra $u\mathbf{C}$. We write $u\mathbf{C}' := [\varepsilon, s_0, s_1, \pi_j^n, \wedge, \vee; \text{comp}, u\mathbf{BRN}']$, where $u\mathbf{BRN}'$ is defined as $u\mathbf{BRN}$ but with the bounding polynomial $k \in [\varepsilon, s_1, \pi_j^n, \cdot, \#; \text{comp}]$. It is clear that $u\mathbf{C}$ is contained in $u\mathbf{C}'$; namely the function $\#$ can easily be defined (as in, e.g., the proof of Prop. 35 later). We will prove that $u\mathbf{C}'$ is contained in $u\mathbf{B}$.

► **Lemma 34.** *For all $f \in u\mathbf{C}'$ there is a polynomial $p_f(n)$ and some $f'(w; \vec{x}) \in u\mathbf{B}$ such that, for all \vec{x}, w , $(|w| \geq p_f(|\vec{x}|) \Rightarrow f(\vec{x}) = f'(w; \vec{x}))$.*

Proof sketch. The proof is similar to the proof of the analogous statement for \mathbf{FP} given in [3], with routine adaptations to deal with uniformity. We proceed by induction on the definition of f in $u\mathbf{C}'$, with the interesting case being when f is defined by $u\mathbf{BRN}'$, say from functions g, h and k . Let g', p_g, h' and p_h be the appropriate functions and polynomials obtained by the inductive hypothesis. We would like to define $f' \in u\mathbf{B}$ and a polynomial p_f such that, for all w, x, \vec{x} , whenever $|w| \geq p_f(|x|, |\vec{x}|)$ one has $f(x, \vec{x}) = f'(w; x, \vec{x})$. The problem is that in $u\mathbf{B}$, due to the normal-safe constraints, one cannot define f' directly by recursion on x . Therefore we introduce in $u\mathbf{B}$ some auxiliary functions. Define,

$$\begin{aligned} \text{msp}(|\varepsilon|; y) &:= y & \text{msp}(|x|, y;) &:= \text{msp}(|x|; y) \\ \text{msp}(|\mathbf{s}_i x|; y) &:= p_i(\text{msp}(|x|; y)) & X(z, w; x) &:= \text{msp}(|\text{msp}(|z|, w;)|; x) \\ & & I(z, w; x) &:= X(\mathbf{s}_1 z, w; x) \wedge 1 \end{aligned}$$

by $u\mathbf{SRN}$ and by safe composition. The function X is used to “simulate” the recursion over x , with x in a safe input position. Now, by $u\mathbf{SRN}$, we define $F(\varepsilon, w; x, \vec{x}) := \varepsilon$ and,

$$\begin{aligned} &F(\mathbf{s}_i z, w; x, \vec{x}) \\ := &\begin{cases} g'(w; \vec{x}) & \text{if } X(\mathbf{s}_1 z, w; x) = \varepsilon \\ h'(w; I(z, w; x), X(z, w; x), \vec{x}, F(z, w; x, \vec{x})) & \text{otherwise} \end{cases} \end{aligned} \quad (6)$$

using a length conditional, cf. Prop. 16. From here we set $f'(w; x) := F(w, w; x, \vec{x})$ and also,

$$p_f(|x|, |\vec{x}|) := p_h(1, |x|, |\vec{x}|, b_f(|x|, |\vec{x}|)) + p_g(|\vec{x}|) + |x| + 1,$$

where b_f is a polynomial bounding the length of the outputs of f (which exists since $f \in uC'$).

Given x, \vec{x} , take w such that $|w| \geq p_f(|x|, |\vec{x}|)$. We will prove, by subinduction on $|u|$, that, if $|w| - |x| \leq |u| \leq |w|$, then $F(u, w; x, \vec{x}) = f(X(u, w; x), \vec{x})$. Since $X(w, w; x) = x$, we thus obtain that $f'(w; x, \vec{x}) = F(w, w; x, \vec{x}) = f(x, \vec{x})$, as required.

Let us take an arbitrary u such that $|w| - |x| \leq |u| \leq |w|$. Note that $|w| - |x| \geq 1$, and thus we may write $u = s_i z$ for some z . We have two cases:

- If $|s_i z| = |w| - |x|$ then $X(s_i z, w; x) = \varepsilon$, and so $F(s_i z, w; x, \vec{x}) = g'(w; \vec{x}) = g(\vec{x}) = f(\varepsilon, \vec{x}) = f(X(s_i z, w; x), \vec{x})$.
- If $|s_i z| > |w| - |x|$ then $X(s_i z, w; x) \neq \varepsilon$ and so:

$$\begin{aligned} F(s_i z, w; x, \vec{x}) &= h'(w; I(z, w; x), X(z, w; x), \vec{x}, F(z, w; x, \vec{x})) && \text{by (6)} \\ &= h(I(z, w; x), X(z, w; x), \vec{x}, F(z, w; x, \vec{x})) && \text{by inductive hypothesis} \\ &= h(I(z, w; x), X(z, w; x), \vec{x}, f(X(z, w; x), \vec{x})) && \text{by subinductive hypothesis} \\ &= f(X(s_i z, w; x), \vec{x}) && \text{by definition of } f. \end{aligned}$$

◀

► **Proposition 35.** *If $f(\vec{x})$ in uC , then we have $f(\vec{x};) \in uB$.*

Proof. For f in uC , recalling that $uC \subseteq uC'$, take $f' \in uB$ and a polynomial p_f given by Lemma 34 above. It suffices to prove that there exists $r \in uB$ such that $|r(\vec{x};)| \geq p_f(|\vec{x}|)$, for all \vec{x} , whence we have $f'(r(\vec{x};); \vec{x}) = f(\vec{x})$ as required, cf. Lemma 34 above. For this we simply notice that the usual definitions of polynomial growth rate functions, e.g. from [3], can be conducted in unary, using only uniform recursion. Namely, define \oplus and \otimes in uB as follows, by $uSRN$,

$$\begin{aligned} \oplus(\varepsilon; y) &:= y & \otimes(\varepsilon; y) &:= \varepsilon \\ \oplus(s_i x; y) &:= s_1(\oplus(x; y)) & \otimes(s_i x; y) &:= \oplus(y; \otimes(x; y)) \end{aligned}$$

so that $|\oplus(x; y)| = |x| + |y|$ and $|\otimes(x; y)| = |x| \times |y|$. By safe composition we may also write $\oplus(x; y;)$ in uB , yielding an appropriate function $r(\vec{x};) \in uB$. ◀

As a consequence of Props. 33 and 35 in this section, and Thms. 17 and 30 earlier, we summarise the contributions of this work in the following characterisation:

► **Theorem 36.** $uB = uC = \text{posFP}$.

8 Conclusions

In this work we observed that characterisations of “positive” polynomial-time computation in [14] are similarly robust in the functional setting. We gave a function algebra uC for **posFP** by *uniformising* the recursion scheme in Cobham’s characterisation for **FP**, and gave a characterisation based on safe recursion too. We also observed that the tally functions of **posFP** are precisely the unary encodings of **FLINSPACE** functions on \mathbb{N} .

uC has a natural generalisation for arbitrary ordered alphabets, not just $\{0, 1\}$. This is similarly the case for the circuit families and machine model we presented in Sect. 2. We believe these, again, induce the same class of functions, and can even be embedded monotonically into $\{0, 1\}$, thanks to appropriate variants of $uBRN$ in uC , e.g. Thm. 28.

Unlike for non-monotone functions, there is an interesting divergence between the monotone functions on binary words and those on the integers. Viewing the latter as *finite sets*, characterised by their binary representation, we see that the notion of monotonicity induced by \subseteq is actually more restrictive than the one studied here on binary words. For example, natural numbers of different lengths may be compared, and the *bit* function is no longer monotone. In fact, a natural way to characterise such functions would be to *further* uniformise recursion schemes, by also relating the base case to the inductive step, e.g.:

$$\begin{aligned} f(0, \vec{x}) &= h(0, 0, \vec{x}, 0) \\ f(s_i x, \vec{x}) &= h(i, x, \vec{x}, f(x, \vec{x})) \end{aligned}$$

Adapting such recursion schemes to provide a “natural” formulation of the positive polynomial-time predicates and functions on \mathbb{N} is the subject of ongoing work.

Finally, this work serves as a stepping stone towards providing *logical theories* whose provably recursive functions correspond to natural monotone complexity classes. *Witnessing theorems* for logical theories typically compile to function algebras on the computation side, and in particular it would be interesting to see if existing theories for monotone *proof complexity* from [8] appropriately characterise positive complexity classes. We aim to explore this direction in future work.

References

- 1 Noga Alon and Ravi B Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1):1–22, 1987.
- 2 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On Uniformity within NC^1 . *J. Comput. Syst. Sci.*, 41(3):274–306, 1990. doi:10.1016/0022-0000(90)90022-D.
- 3 Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992. doi:10.1007/BF01201998.
- 4 Samuel R. Buss. *Bounded arithmetic*, volume 1 of *Studies in Proof Theory*. Bibliopolis, Naples, 1986.
- 5 Peter Clote and Evangelos Kranakis. *Boolean Functions and Computation Models*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002. doi:10.1007/978-3-662-04943-3.
- 6 A. Cobham. The intrinsic computational difficulty of functions. In *Proc. of the International Congress for Logic, Methodology, and the Philosophy of Science*, pages 24–30. Amsterdam, 1965.
- 7 Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.
- 8 Anupam Das. From positive and intuitionistic bounded arithmetic to monotone proof complexity. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 126–135, 2016. doi:10.1145/2933575.2934570.
- 9 Fernando Ferreira. Polynomial time computable arithmetic. In *Contemporary Mathematics*, volume 106, pages 137–156. AMS, 1990.
- 10 Michelangelo Grigni. *Structure in monotone complexity*. PhD thesis, Duke University, 1991.
- 11 Michelangelo Grigni and Michael Sipser. Monotone complexity. In *London Mathematical Society Symposium on Boolean Function Complexity*, New York, NY, USA, 1992. Cambridge University Press.
- 12 Andrzej Grzegorzczuk. *Some classes of recursive functions*. Instytut Matematyczny Polskiej Akademii Nauk, 1953.
- 13 A D Korshunov. Monotone boolean functions. *Russian Mathematical Surveys*, 58(5), 2003.

- 14 Clemens Lautemann, Thomas Schwentick, and Iain A. Stewart. On positive P. In *IEEE Conference on Computational Complexity '96*, 1996.
- 15 Clemens Lautemann, Thomas Schwentick, and Iain A. Stewart. Positive versions of polynomial time. *Inf. Comput.*, 147(2):145–170, 1998. doi:10.1006/inco.1998.2742.
- 16 Daniel Leivant. A foundational delineation of computational feasibility. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 2–11, 1991. doi:10.1109/LICS.1991.151625.
- 17 Daniel Leivant. A foundational delineation of poly-time. *Inf. Comput.*, 110(2):391–420, 1994. doi:10.1006/inco.1994.1038.
- 18 Isabel Oitavem. New recursive characterizations of the elementary functions and the functions computable in polynomial space. *Revista Matemática de la Universidad Complutense de Madrid*, 10(1):109–125, 1997.
- 19 Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.
- 20 A. A. Razborov. Lower bounds on the monotone complexity of some Boolean functions. *Doklady Akademii Nauk SSSR*, 285, 1985.
- 21 Robert W. Ritchie. Classes of predictably computable functions. *Journal of Symbolic Logic*, 28(3):252–253, 1963.
- 22 Walter L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981. doi:10.1016/0022-0000(81)90038-6.
- 23 E. Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 8(1):141–142, 1988.
- 24 Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):23–33, 1976. doi:10.1016/0304-3975(76)90062-1.

Non-Wellfounded Proof Theory For (Kleene+Action)(Algebras+Lattices)

Anupam Das

University of Copenhagen, Copenhagen, Denmark

anupam.das@di.ku.dk

Damien Pous

Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France

damien.pous@ens-lyon.fr

Abstract

We prove cut-elimination for a sequent-style proof system which is sound and complete for the equational theory of Kleene algebra, and where proofs are (potentially) non-wellfounded infinite trees. We extend these results to systems with meets and residuals, capturing ‘star-continuous’ action lattices in a similar way. We recover the equational theory of all action lattices by restricting to regular proofs (with cut) – those proofs that are unfoldings of finite graphs.

2012 ACM Subject Classification Theory of computation → Proof theory, Theory of computation → Regular languages

Keywords and phrases Kleene algebra, proof theory, sequent system, non-wellfounded proofs

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.19

Related Version Long version at <https://hal.archives-ouvertes.fr/hal-01703942>.

Funding This work has been funded by the European Research Council (ERC) under the European Union’s Horizon 2020 programme (*CoVeCe*, grant agreement No. 678157, and *MiLC*, grant agreement No. 753431). This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program “Investissements d’Avenir” (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

1 Introduction

The axioms of *Kleene algebras* are sound and complete for the theory of regular expressions under language equivalence [24, 27, 4]. As a consequence, the equational theory of Kleene algebras is decidable (in fact PSPACE-complete). Models of these axioms of particular interest include formal languages and binary relations. For binary relations, the Kleene star is interpreted as reflexive transitive closure, whence the axioms of Kleene algebra make it possible to reason abstractly about program correctness [22, 23, 3, 19, 1]. The aforementioned decidability result moreover makes it possible to automate interactive proofs [5, 26, 30].

There are however important extensions of Kleene algebras which are not yet fully understood. These include *action algebras* [31], where two ‘residual’ operations are added, *Kleene lattices*, where a ‘meet’ operation is added, and *action lattices* [25], where all three operations are added. Pratt introduced residuals in order to *internalise* the induction rules of the Kleene star, as we explain later; they allow us to express properties of relations such as well-foundedness in a purely algebraic way [12]. Kozen added the meet operation to action algebra to obtain a structure closed under taking matrices. In the context of program verification, meets are useful since they allow us to express conjunctions of local specifications.



© Anupam Das and Damien Pous;

licensed under Creative Commons License CC-BY

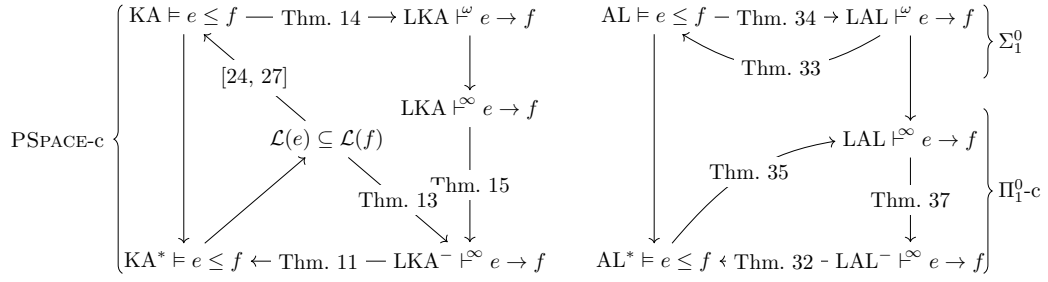
27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 19; pp. 19:1–19:18

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Context and contributions for Kleene algebra and action lattices.

Unfortunately, the decidability of the three corresponding equational theories is still open, and there is no known notion of ‘free model’ for them that is analogous to the rational languages for Kleene algebra. In this paper, we explore a proof-theoretic approach to such questions: we provide sequent calculi that capture these theories which we show admit a form of cut-elimination. Although this does not (yet) give us decidability, it does improve our understanding of these theories:

- we obtain a computational interpretation of proofs of inequalities in our systems as program transformers, which could prove useful to describe free models;
- we recover two conservativity results: action lattices are conservative over (star-continuous) Kleene lattices and action algebra, thanks to the sub-formula property; (these results are also implied by [29]).
- we obtain structural properties, e.g., as Whitman did when he proved cut-elimination for the theory of lattices, which we aim to exploit in consequent research.

We first focus on pure Kleene algebra, which is easier to handle and enables a simpler presentation. Being a well-established theory, we are able to relate our results to existing ones in the literature, identifying which issues become relevant when moving to extensions.

Kleene algebra

In our sequent system, called LKA, proofs are finitely branching, but possibly infinitely deep (i.e. not wellfounded). To prevent fallacious reasoning, we give a simple validity criterion for proofs with cut, and prove that the corresponding system admits cut-elimination. The difficulty in the presence of infinitely deep proofs consists in proving that cut-elimination is productive; we do so by using the natural interpretation of regular expressions as data types for parse-trees [15], and by giving an interpretation of proofs as parse-tree transformers. Such an idea already appears in [18] but in a simpler setting, for a finitary natural deduction system rather than for a non-wellfounded sequent calculus.

The results we prove about LKA are summarised in Fig. 1(left). In addition to cut-elimination (Thm. 15), we prove that the system is sound for all *star-continuous* Kleene algebras (Thm. 11), and conversely, that it is complete w.r.t. the language theoretic interpretation of regular expressions (Thm. 13). We actually refine this latter result by showing that every proof from Kleene algebra axioms can be translated into a *regular* proof with cut (Thm. 14), i.e., a proof with cut which is the unfolding of a finite graph. Note, however, that regularity is not preserved by cut-elimination: the class of cut-free regular proofs in LKA is incomplete w.r.t. Kleene algebra.

Action algebras, Kleene lattices, and action lattices

Despite its finite quasi-equational presentation, the equational theory of Kleene algebra is not finitely based: Redko proved that any finite set of equational axioms must be incomplete [32]. However, by adding two binary operations to the signature, Pratt showed how to obtain a finitely based extension which is conservative over the equational theory of Kleene algebras [31]. These two operations, called *left residual* (\backslash) and *right residual* ($/$), are ‘adjoint’ to sequential composition and, as we mentioned, such structures are called *action algebras*. Kozen then proposed *action lattices* [25], where the signature is extended further to include a binary meet operation (\cap). We call *Kleene lattices* the structures consisting of Kleene algebra extended just with meets.

While both action algebras and action lattices are finitely based and conservatively extend Kleene algebra, they bring some difficulties. By definition, their equational theories are at most Σ_1^0 , so that they must differ from their star-continuous variants which are Π_1^0 -complete [7, 29]. (Buszkowski proved the lower bound and Palka proved the upper bound.) In contrast, by Kozen’s completeness result we have that Kleene algebra and star-continuous Kleene algebra give rise to the same equational theory, which is PSPACE-complete. This matter remains open for Kleene lattices since Buszkowski’s lower bound does not apply.

Residuals and meets naturally correspond to linear implication and additive conjunction [20, 29], from (non-commutative intuitionistic) linear logic [17]. They are also essential connectives in the Lambek-calculus and related substructural logics [28]. We extend LKA accordingly into a system LAL and obtain the results summarised in Fig. 1 (right): LAL is complete for star continuous action lattices (Thm. 35); it still admits cut-elimination (Thm. 37); thus it is also sound w.r.t. star continuous action lattices (Thms. 32). Furthermore we are able to show that its regular fragment with cut is in fact sound and complete for all action lattices (Thm. 33); this somewhat surprising result gives us a nontrivial yet finite proof theoretic representation of the theory of action lattices. The proof of soundness reasons inductively on the cycle structure of such regular proofs, and we crucially exploit the availability of both residuals and meets: for action algebra and Kleene lattices, it remains open whether the corresponding regular fragments with cut are sound.

Thms. 32, 34, and 37 are proved by extending the proofs of Thms. 11, 14, and 15 to deal with the additional connectives. Amongst those, cut-elimination is the most delicate extension, relying on higher types to interpret residuals, and proving that LAL proofs still yield terminating programs in such a setting. Thm. 13 cannot be extended directly, due to the lack of a free model analogous to the regular languages for Kleene algebra when adding residuals or meet. This is why we instead rely on cut-elimination for completeness.

As explained above, while all notions are equivalent in the case of Kleene algebra (Fig. 1 (left)), complexity arguments make it possible to separate the lower and upper parts of Fig. 1 (right), except for Kleene lattices. Whether the upper part is decidable remains open, but it is interesting to note that we managed to characterise action lattices in such a way that the non-regular/regular distinction at the proof-level corresponds precisely to the difference between the star continuous and general cases, respectively. One potentially fruitful direction towards the decidability of action lattices is to characterise the image of regular proofs under cut-elimination. We aim to explore this possibility in future work.

Related work

We briefly discussed the cut-free variant of the system LKA in [10] (with a simpler validity criterion), observing that its regular fragment is incomplete (due to the absence of cut). Our main contribution was a variant of it based on ‘hypersequents’, HKA, whose regular fragment is sound and complete without cut, and admits a PSPACE proof search procedure.

Palka proposed a sequent system for star continuous action lattices in [29], for which she proved cut-elimination. Its non-star rules are precisely those of LKA, but the system is wellfounded and relies on an ‘omega-rule’ for Kleene star with infinitely many premisses, in the traditional school of infinitary proof theory, cf. [33]. Such an approach does not admit a notion of finite proof analogous to our regular proofs, corresponding to the upper parts of Fig. 1. Wurm also proposed a (finite, and thus wellfounded) sequent system for Kleene algebra [34]. Unfortunately his cut-admissibility theorem does not hold – see [10].

The normalisation theory of *linear logic* with (least and greatest) fixed point operators has been studied in [14] and, more comprehensively, in [13]. While the latter is a rather general framework, its exposition still differs significantly from the current work for various reasons. One immediate difference is that their setting is commutative while ours is non-commutative, and so those results are not directly applicable. A more important difference is that they do not have any atoms in their language, reasoning only on closed formulae. This is rather significant from the point of view of normalisation, since the convergence of cut-elimination becomes more complicated in presence of atoms. The argument we give in Sect. 4 uses different ideas that are closely related to the language-based models of our algebras and the natural interpretation of language inclusions as programs [18]. A *game semantics* approach to cut-elimination for non-wellfounded proofs is given in [8], though in that work only finitely many cuts in a proof are considered and so it does not seem sufficient to handle the star rules in this work.

2 Preliminaries on Kleene algebra and extensions

Let A be a finite *alphabet*. *Regular expressions* [21] are generated as follows:

$$e, f ::= e \cdot e \mid e + e \mid e^* \mid 1 \mid 0 \mid a \in A$$

Sometimes we may simply write ef instead of $e \cdot f$. Each expression e generates a rational language $\mathcal{L}(e) \subseteq A^*$, defined in the usual way.

A *Kleene algebra* is a tuple $(K, 0, 1, +, \cdot, *, \leq)$ where $(K, 0, 1, +, \cdot)$ is an idempotent semiring and where the following properties hold, where $x \leq y$ is a shorthand for $x + y = y$.

$$1 + xx^* \leq x^* \quad \text{if } xy \leq y \text{ then } x^*y \leq y \quad \text{if } yx \leq y \text{ then } yx^* \leq y \quad (1)$$

There are several equivalent variants of this definition [9]. Intuitively we have that x^*y (resp., yx^*) is the least fixpoint of the function $z \mapsto y + xz$ (resp., $z \mapsto y + zx$). We write $\text{KA} \models e \leq f$ if the inequality $e \leq f$ holds universally in all Kleene algebras – or, equivalently, if it is derivable from the axioms of Kleene algebra. Kozen [24] and Krob [27] showed that this axiomatisation is complete for language inclusions, corresponding to the right-to-left implication in the following characterisation (the other direction is routine).

► **Theorem 1** ([24, 27]). $\text{KA} \models e \leq f$ if and only if $\mathcal{L}(e) \leq \mathcal{L}(f)$.

A Kleene algebra is *star-continuous* if for all elements x, y, z , xy^*z is the least upper bound of the sequence $(xy^iz)_{i \in \mathbb{N}}$, where $y^0 = 1$ and $y^{i+1} = yy^i$. In presence of the other laws, star-continuity is equivalent to the following condition:

$$\forall xyz, (\forall i \in \mathbb{N}, xy^iz \leq t) \Rightarrow xy^*z \leq t.$$

We write $\text{KA}^* \models e = f$ when the equality $e = f$ holds in all star-continuous Kleene algebras. Formal languages form a star-continuous Kleene algebra, and so by completeness of Kleene algebra w.r.t. rational languages, we have $\text{KA}^* \models e = f$ iff $\text{KA} \models e = f$; this is the triangle on the left in Fig. 1.

$$\begin{array}{c}
\text{cut} \frac{\Delta \rightarrow e \quad \Gamma, e, \Sigma \rightarrow f}{\Gamma, \Delta, \Sigma \rightarrow f} \quad \text{id} \frac{}{e \rightarrow e} \quad 0\text{-l} \frac{}{\Gamma, 0, \Delta \rightarrow e} \quad 1\text{-l} \frac{\Gamma, \Delta \rightarrow e}{\Gamma, 1, \Delta \rightarrow e} \quad 1\text{-r} \frac{}{\rightarrow 1} \\
\text{-l} \frac{\Gamma, e, f, \Delta \rightarrow g}{\Gamma, e \cdot f, \Delta \rightarrow g} \quad \text{+l} \frac{\Gamma, e, \Delta \rightarrow g \quad \Gamma, f, \Delta \rightarrow g}{\Gamma, e + f, \Delta \rightarrow g} \quad \text{*l} \frac{\Gamma, \Delta \rightarrow f \quad \Gamma, e, e^*, \Delta \rightarrow f}{\Gamma, e^*, \Delta \rightarrow f} \\
\text{-r} \frac{\Gamma \rightarrow e \quad \Delta \rightarrow f}{\Gamma, \Delta \rightarrow e \cdot f} \quad \text{+r}_i \frac{\Gamma \rightarrow e_i}{\Gamma \rightarrow e_1 + e_2} \quad i \in \{1, 2\} \quad \text{*r}_1 \frac{}{\rightarrow e^*} \quad \text{*r}_2 \frac{\Gamma \rightarrow e \quad \Delta \rightarrow e^*}{\Gamma, \Delta \rightarrow e^*}
\end{array}$$

■ **Figure 2** The rules of LKA.

An *action lattice* is a Kleene algebra with three additional binary operations, left and right *residuals* ($\backslash, /$), and *meet* (\cap) defined by the following equivalences:

$$\forall xyz, \quad y \leq x \backslash z \Leftrightarrow xy \leq z \Leftrightarrow x \leq z / y \quad \text{and} \quad \forall xyz, \quad z \leq x \cap y \Leftrightarrow z \leq x \wedge z \leq y$$

An *action algebra* is a Kleene algebra with residuals, a *Kleene lattice* is a Kleene algebra with meets. We extend regular expressions accordingly, writing $\text{AL} \models e \leq f$ when the inequation $e \leq f$ holds in all action lattices, and $\text{AL}^* \models e \leq f$ when it holds in all star continuous action lattices. Note that while rational languages are closed under residuals and intersection, thus forming an action lattice, they are not the ‘free’ one: Thm. 1 fails. The equational theories generated by all action lattices and by the star-continuous ones actually differ, cf. [7, 29].

3 The sequent system LKA

A *sequent* is an expression $\Gamma \rightarrow e$, where Γ is a list of regular expressions and e is a regular expression. For such a sequent we refer to Γ as the *antecedent* and e as the *succedent*, or simply the ‘left’ and ‘right’ hand sides, respectively. We say that a sequent $e_1, \dots, e_n \rightarrow e$ is *valid* if $\text{KA}^* \models e_1 \cdots e_n \leq e$. I.e., the comma is interpreted as sequential composition, and the sequent arrow as inclusion. We may refer to expressions as ‘formulae’ when it is more natural from a proof theoretic perspective, e.g. ‘subformula’ or ‘principal formula’.

The rules of LKA are given in Fig. 2. We call LKA^- the subset of LKA where the *cut* rule is omitted (which corresponds to the system called LKA in [10]). Leaving the $*$ -rules aside, these rules are those of the non-commutative variant of intuitionistic linear logic [17], restricted to the following connectives: multiplicative conjunction (\cdot), additive disjunction ($+$) and additive falsity (0) (for which there is no right rule). The rules for Kleene star can be understood as those arising from the characterisation of e^* as a fixed point: $e^* = \mu x.(1 + ex)$. In contrast, Palka [29] follows the alternative interpretation of Kleene star as an infinite sum, $e^* = \sum_i e^i$, whence her left rule for Kleene star with infinitely many premisses, and the infinitely many right rules she uses for this operation.

As previously mentioned, we consider infinitely deep proofs, so it is necessary to impose a validity criterion to ensure that derivations remain sound.

► **Definition 2.** A (binary, possibly infinite) *tree* is a prefix-closed subset of $\{0, 1\}^*$, which we view with the root, ε , at the bottom; elements of $\{0, 1\}^*$ are called *nodes*. A *preproof* is a labelling π of a tree by sequents such that, for every node v with children v_1, \dots, v_n ($n = 0, 1, 2$), the expression $\frac{\pi(v_1) \cdots \pi(v_n)}{\pi(v)}$ is an instance of an LKA rule. Given a node v in a preproof π , we write π_v for the sub-preproof rooted at v , defined by $\pi_v(w) = \pi(vw)$. A

preproof is *regular* if it has finitely many distinct subtrees, i.e. it can be expressed as the infinite unfolding of a finite graph. A preproof is *cut-free* if it does not use the *cut*-rule.

We will use standard proof theoretic terminology about *principal formulas* and *ancestry* in proofs, e.g. from [6]. (see [11, App. A] for further details). The notion of validity below is similar to [13], adapted to our setting.

► **Definition 3.** A *thread* is a maximal path through the graph of (immediate) ancestry in a preproof. By definition it must start at a conclusion formula or at a cut formula and it only goes upwards. A thread is *valid* if it is principal for a $*-l$ step infinitely often. A preproof is *valid* if every infinite branch eventually has a valid thread. A *proof* is a valid preproof. We write $\text{LKA} \vdash^\infty \Gamma \rightarrow e$ if the sequent $\Gamma \rightarrow e$ admits a proof, $\text{LKA} \vdash^\omega \Gamma \rightarrow e$ if it admits a regular proof, and $\text{LKA}^- \vdash^\infty \Gamma \rightarrow e$ if it admits a cut-free proof.

Notice that every valid thread eventually follows a unique (star) formula, by the subformula property. Let us consider some examples of (pre)proofs. In all cases, we will use the symbol \bullet to indicate circularities (i.e. to identify roots of the same subtree), colours to mark some of the threads, and double lines to denote finite derivations.

► **Example 4.** Here is a regular and cut-free proof of $(b + c)^* \rightarrow (c + b)^*$:

$$\frac{\frac{\frac{\vdots}{b + c \rightarrow c + b}}{\rightarrow (c + b)^*} \quad \frac{\frac{\vdots}{(b + c)^* \rightarrow (c + b)^*}}{b + c, (b + c)^* \rightarrow (c + b)^*} \bullet}{(b + c)^* \rightarrow (c + b)^*} \bullet$$

► **Example 5 (Atomicity of identity).** As in many common sequent systems, initial identity steps can be reduced to atomic form, although for this we crucially rely on access to non-wellfounded (yet regular) proofs. As usual, we proceed by induction on the size of an identity step, whence the crucial case is for the Kleene star,

$$\frac{\frac{\frac{\triangle IH}{e \rightarrow e}}{\rightarrow e^*} \quad \frac{\frac{\vdots}{e^* \rightarrow e^*}}{e, e^* \rightarrow e^*} \bullet}{e^* \rightarrow e^*} \bullet$$

where the derivation marked *IH* is obtained by the inductive hypothesis.

Note that while LKA^- satisfies the subformula property, the size and number of sequents occurring in a cut-free proof is not a priori bounded, due to the $*-l$ rule:

► **Example 6 (A non-regular proof).** The only cut-free proof of the sequent $a, a^* \rightarrow a^*a$ is the one on the left below:

$$\frac{\frac{\frac{\vdots}{a, a \rightarrow a^*a}}{a \rightarrow a^*a} \quad \frac{\frac{\vdots}{a, a, a, a^* \rightarrow a^*a}}{a, a, a^* \rightarrow a^*a}}{a, a^* \rightarrow a^*a} \bullet \quad \frac{\frac{\frac{\vdots}{a, a^* \rightarrow a^*a} \bullet}{a \rightarrow a^*a} \quad \frac{\frac{\vdots}{a, a^*a \rightarrow a^*a}}{a, a, a^* \rightarrow a^*a}}{a, a^* \rightarrow a^*a} \bullet$$

This proof contains all sequents of the form $a, \dots, a, a^* \rightarrow a^*a$, whence non-regularity. A regular proof with cuts is given on the right above; see [10] for more details on the lack of regularity in LKA^- and how to recover regularity in a cut-free setting, using ‘hypersequents’.

► **Example 7** (Two invalid preproofs). The following preproofs are not valid; they derive invalid sequents.

$$\begin{array}{c}
 \vdots \\
 \frac{1-r \quad \frac{\rightarrow 1}{*r_2}}{a \rightarrow 1^*} \quad \frac{*r_2 \quad \frac{a \rightarrow 1^*}{*r_2}}{a \rightarrow 1^*} \bullet \\
 \bullet
 \end{array}
 \qquad
 \begin{array}{c}
 \frac{id \quad \frac{id \quad \frac{id \quad \frac{\vdots}{a^* \rightarrow a^*}}{a^* \rightarrow a^*}}{a \rightarrow a} \quad \frac{*r_2 \quad \frac{a, a^* \rightarrow a^*}{cut}}{a, a^* \rightarrow a^*} \quad \frac{*l \quad \frac{\vdots}{a^* \rightarrow b^*}}{a^* \rightarrow b^*} \bullet \\
 \frac{*l \quad \frac{\rightarrow b^*}{*l}}{a^* \rightarrow b^*} \bullet
 \end{array}$$

The left preproof is cut-free and infinite; since it does not contain any $*l$ -rule, it cannot be valid. On the right the principal formula of the $*l$ -rule is the cut formula of the *cut*-rule so that the only infinite thread is the one along the occurrences of b^* , and this formula is never principal for a $*l$ step.

The notion of validity we use here actually generalises the notion of *fairness* we used in [10], where we were working only with cut-free preproofs:

► **Proposition 8.** *A cut-free preproof is valid if and only if it is fair for $*l$, i.e. every infinite branch contains infinitely many occurrences of $*l$.*

Proof sketch. The left-right implication is immediate. Conversely, every infinite path in a fair cut-free preproof has infinitely many $*l$ steps, but there are only finitely many possible principal formulae by the subformula property. One can thus extract a valid thread. ◀

An alternative criterion for cut-free preproofs is obtained as follows:

► **Proposition 9.** *A cut-free preproof is valid if and only if it has no infinite branch with a tail of only $*r_2$ -steps.*

Proof. Define the ‘weight’ of a sequent to be the multiset of its formulae, ordered by the subformula relation. This measure strictly decreases when reading LKA^- rules bottom-up, except for the right premisses of rules $*l$ and $*r_2$; for the latter, it either remains unchanged (when Γ is empty) or it strictly decreases. Thus every infinite branch of a cut-free preproof either contains infinitely many $*l$ steps, or eventually contains only $*r_2$ steps. ◀

Observe that the proof on the left in Ex. 7 does not satisfy this condition.

The cut-free system LKA^- is sound and complete for Kleene algebras. Thanks to the completeness theorem for Kleene algebras, Thm. 1, it suffices to prove soundness with respect to star-continuous Kleene algebra. We first prove the following lemma:

► **Lemma 10.** *If $LKA^- \vdash^\infty \Gamma, e^*, \Delta \rightarrow f$ then, for each $n \in \mathbb{N}$, $LKA^- \vdash^\infty \Gamma, e^n, \Delta \rightarrow f$.*

Proof. We define appropriate preproofs from by induction on n . Replace every direct ancestor of e^* by e^n , adjusting origins as follows,

$$\frac{*l \quad \frac{\Gamma, \Delta \rightarrow f \quad \Gamma, e, e^*, \Delta \rightarrow f}{\Gamma, e^*, \Delta \rightarrow f}}{\Gamma, e^*, \Delta \rightarrow f} \quad \mapsto \quad \frac{1-l \quad \frac{\Gamma, \Delta \rightarrow f}{\Gamma, 1, \Delta \rightarrow f}}{\Gamma, 1, \Delta \rightarrow f} \quad \text{or} \quad \frac{..l \quad \frac{\Gamma, e, e^{n-1}, \Delta \rightarrow f}{\Gamma, e^n, \Delta \rightarrow f}}{\Gamma, e^n, \Delta \rightarrow f}$$

when $n = 0$ or $n > 0$, respectively. In the latter case we appeal to the inductive hypothesis.

Notice that, on branches where e^* is never principal, this is simply a substitution of e^n for e^* everywhere along the branch. The preproof resulting from this entire construction is fair since every infinite branch will share a tail with the proof we began with. ◀

We can now prove soundness w.r.t. star continuous Kleene algebra:

► **Theorem 11** (Soundness). *If $\text{LKA}^- \vdash^\infty e_1, \dots, e_n \rightarrow e$, then $\text{KA}^* \models e_1 \cdots e_n \leq e$.*

Proof. First observe that every rule of LKA is sound: if its premisses are valid then so is its conclusion. Let π be an LKA^- proof of $\Sigma \rightarrow f$. We proceed by structural induction on the multiset of formulae in its conclusion, via case analysis on the last rule. For all but two cases, we just use soundness of the rule and the induction hypotheses. The first remaining case is $*-r_2$, where we must appeal to a sub-induction since the measure does not always strictly decrease in the right premiss (Prop. 9). The last case is $*-l$, where $\Sigma = \Gamma, e^*, \Delta$. By Lem. 10, π can be transformed into proofs π_n of $\Gamma, f^n, \Delta \rightarrow f$ for each $n \in \mathbb{N}$. Each π_n derives a sequent whose weight is strictly smaller than that of $\Sigma \rightarrow f$, which is thus valid by the inductive hypothesis. Finally, this means that $\Gamma, f^*, \Delta \rightarrow g$ is valid, by star-continuity. ◀

For completeness of LKA^- , we can get a direct proof by starting from the free model of rational languages (Fig. 1). This strategy is no longer possible for Kleene lattices, action algebras and action lattices, for which we will need to go through cut-elimination. We first prove completeness for sequents whose antecedent is a word:

► **Lemma 12.** *If $a_1 \dots a_n$ is a word in $\mathcal{L}(e)$ for some expression e , then there is a finite proof of the sequent $a_1, \dots, a_n \rightarrow e$ using only right logical rules.*

Proof. By a straightforward induction on e . ◀

► **Theorem 13** (Completeness). *If $\mathcal{L}(e_1 \cdots e_n) \subseteq \mathcal{L}(e)$ then $\text{LKA}^- \vdash^\infty e_1, \dots, e_n \rightarrow e$.*

Proof. This is proved like in [10] for HKA: all left rules of LKA^- are invertible so that they can be applied greedily; doing so, one obtains an infinite tree whose leaves are sequents of the shape $a_1, \dots, a_k \rightarrow e$, with $k \geq 0$, where $a_1 \dots a_k$ is a word in $\mathcal{L}(e_1 \cdots e_n)$ and thus in $\mathcal{L}(e)$ by assumption. Those leaves can be replaced by finite derivations using by Lem. 12. Notice, that we obtain fairness, since any infinite branch of only left rules must contain $*-l$ infinitely often. ◀

The previous proof builds infinite and non-regular derivations whenever the language of the starting antecedent is infinite. For instance, it would yield the proof given on the left in Ex. 6. By using a different technique, we show in the following theorem, that we can get regular proofs if we allow the cut-rule.

► **Theorem 14** (Regular completeness). *If $\text{KA} \models e \leq f$ then $\text{LKA} \vdash^\omega e \rightarrow f$.*

Proof. We prove the statement for equalities. Consider the relation \equiv defined by $e \equiv f$ if $\text{LKA} \vdash^\omega e \rightarrow f$ and $\text{LKA} \vdash^\omega f \rightarrow e$. This relation is an equivalence on regular expressions thanks to the cut rule, and it is easily shown to be preserved by all contexts (i.e. it is a congruence). Also remark that we have $e + f \equiv f$ iff $\text{LKA} \vdash^\omega e \rightarrow f$, thanks to the cut-rule and the rules about sum. It then suffices to show that regular expressions quotiented by \equiv form a Kleene algebra. The (in)equational axioms defining KA can be proved by finite derivations. The only difficulty is in dealing with the two implications from the definition of Kleene algebra (1). We implement them as follows:

$$\begin{array}{c}
 \vdots \\
 \text{*} \text{-} l \frac{}{e^*, f \rightarrow f} \bullet \\
 \text{IH} \\
 \text{cut} \frac{\text{id} \frac{}{f \rightarrow f} \quad e, f \rightarrow f}{e, e^*, f \rightarrow f}}{\text{*} \text{-} l \frac{}{e^*, f \rightarrow f} \bullet}
 \end{array}
 \qquad
 \begin{array}{c}
 \text{IH} \\
 \text{cut} \frac{\text{id} \frac{}{f \rightarrow f} \quad f, e \rightarrow f}{f, e, e^* \rightarrow f} \quad \text{*} \text{-} l \frac{}{f, e^* \rightarrow f} \bullet \\
 \text{*} \text{-} l \frac{}{f \rightarrow f} \quad \text{*} \text{-} l \frac{}{f, e^* \rightarrow f} \bullet
 \end{array}$$

where the derivations marked IH are obtained from the inductive hypothesis. The preproofs we construct in this way are valid and regular, by inspection. In particular, the only infinite branch not in IH in the above derivations has a valid thread on e^* , coloured in green. ◀

Note the asymmetry when we interpret the two implications: the premisses of the cut rule are swapped when we move from one to the other. This asymmetry comes from the fact that we have a single left rule for Kleene star, which unfolds the star from the left.

4 Cut-elimination for LKA

This section is devoted to proving the following cut-elimination theorem.

► **Theorem 15.** *If $LKA \vdash^\infty \Gamma \rightarrow e$ then $LKA^- \vdash^\infty \Gamma \rightarrow e$.*

Combined with Thm. 11, it establishes the soundness of our criterion for proofs with cuts. This serves as a ‘warm-up’ for the analogous result for the extended system (Sect. 6), which is obtained using the same template.

We show that proofs can be considered as certain *transducers*, transforming *parse-trees* of input words of languages computed by terms. We design them so that a given computation only explores a finite prefix of the proof, which we call the *head*. We then prove that cut-reductions, restricted to the head of a proof, preserve these computations, always terminate, and eventually produce some non-cut rules. We can then repeatedly apply this procedure to remove all cuts from an infinite proof, in a productive way.

4.1 Programs from proofs

We first define programs and their reduction semantics, based on which we prove cut-elimination, in Sect. 4.2. We fix in this section a (valid) LKA proof π and we let v range over its nodes, which we recall are elements of $\{0, 1\}^*$, cf. Dfn. 2.

► **Definition 16** (Programs). *Programs* are defined by the following syntax, where x ranges over a countable set of *variables*, and i ranges over $\{1, 2\}$.

$$M, N ::= x \mid \star \mid \langle M, N \rangle \mid \text{in}_i M \mid [] \mid M :: N \mid v[\vec{M}]$$

Intuitively, programs compute parse-trees for words belonging to the language of an expression. Given a node v of π such that $\pi(v) = \Gamma \rightarrow e$, the last entry corresponds to the application of the subproof π_v , rooted at v , to a list \vec{M} of programs for the antecedent (Γ); it should eventually return a parse-tree for the succedent (e). This is formalised using the following notion of types.

► **Definition 17** (Typing environment). A *typing environment*, written E , is a list of pairs of variables and expressions (written $x : e$), together with a finite antichain of nodes: for any two nodes v and w in the antichain, v is not a prefix of w . We write E, F for the concatenation of two typing environments, which is defined only when this antichain condition on nodes is preserved.

Intuitively, typing environments keep track of which variables and proof nodes are used in a program, to impose linearity constraints; these constraints become crucial when we add residuals and meets, in Sect. 5.

► **Definition 18** (Types). A program M has type e in an environment E , written $E \vdash M : e$ if this judgement can be derived from the rules in Fig. 3.

$$\begin{array}{c}
 \frac{}{x : e \vdash x : e} \quad \frac{}{\vdash \star : 1} \quad \frac{E \vdash M : e \quad F \vdash N : f}{E, F \vdash \langle M, N \rangle : ef} \quad \frac{E \vdash M : e_i}{E \vdash \text{in}_i M : e_1 + e_2} \quad \frac{}{\vdash [] : e^*} \\
 \\
 \frac{E \vdash M : e \quad E' \vdash N : e^*}{E, E' \vdash M :: N : e^*} \quad \frac{\forall i, E_i \vdash M_i : e_i \quad \pi(v) = e_1, \dots, e_n \rightarrow f}{v, E_1, \dots, E_n \vdash v[\vec{M}] : f}
 \end{array}$$

■ **Figure 3** Typing rules for programs.

► **Example 19.** With the proof from Ex. 4, letting ε denote the root node, we have:

$$\begin{array}{l}
 \varepsilon, x : b, y : c \vdash \varepsilon[\text{in}_0 x :: \text{in}_1 y :: []] : (c + b)^* \\
 \varepsilon, z : b + c, z' : b + c, q : (b + c)^* \vdash \varepsilon[z :: z' :: q] : (c + b)^*
 \end{array}$$

► **Observation 20.** Let x_1, \dots, x_n be variables. We have $a_1 \dots a_n \in \mathcal{L}(e)$ iff there exists a program M such that $x_1 : a_1, \dots, x_n : a_n \vdash M : e$. This (unused) observation has no counterpart when considering extensions of Kleene algebra, where there is no longer an appropriate notion of ‘language’ for expressions that constitutes a free model.

► **Definition 21** (Reduction). *Reduction*, written \rightsquigarrow , is the closure under all contexts of the following rules defined by case analysis on the last step of the subproof π_v rooted at v . These rules are written concisely for lack of space; in each case, $v0$ and $v1$ are the nodes of the premisses, when they exist. We moreover assume that the sizes of the vectors match those that arise from the various rules. See [11, App. B] for an extensive definition.

$$\begin{array}{ll}
 id : v[M] \rightsquigarrow M & cut : v[\vec{M}, \vec{N}, \vec{P}] \rightsquigarrow v1[\vec{M}, v0[\vec{N}], \vec{P}] \\
 1-l : v[\vec{M}, \star, \vec{N}] \rightsquigarrow v0[\vec{M}, \vec{N}] & 1-r : v[] \rightsquigarrow \star \\
 \cdot-l : v[\vec{M}, \langle M, N \rangle, \vec{N}] \rightsquigarrow v0[\vec{M}, M, N, \vec{N}] & \cdot-r : v[\vec{M}, \vec{N}] \rightsquigarrow \langle v0[\vec{M}], v1[\vec{N}] \rangle \\
 +-l : v[\vec{M}, \text{in}_i M, \vec{N}] \rightsquigarrow vi[\vec{M}, M, \vec{N}] & +-r_i : v[\vec{M}] \rightsquigarrow \text{in}_i(v0[\vec{M}]) \\
 *-l : v[\vec{M}, [], \vec{N}] \rightsquigarrow v0[\vec{M}, \vec{N}] \text{ and} & *-r_1 : v[] \rightsquigarrow [] \\
 v[\vec{M}, M :: N, \vec{N}] \rightsquigarrow v1[\vec{M}, M, N, \vec{N}] & *-r_2 : v[\vec{M}, \vec{N}] \rightsquigarrow v0[\vec{M}] :: v1[\vec{N}]
 \end{array}$$

When useful we write, say, \rightsquigarrow_{cut} to indicate a reduction according to the *cut* rule above.

► **Example 22.** Continuing with the proof from Ex. 4 we have the following complete reductions. The second program still contains calls to proofs in the end because the inputs were under-specified.

$$\begin{array}{ll}
 \varepsilon[\text{in}_0 x :: \text{in}_1 y :: []] & \varepsilon[z :: z' :: q] \\
 \rightsquigarrow 1[\text{in}_0 x, \text{in}_1 y :: []] & \rightsquigarrow 1[z, z' :: q] \\
 \rightsquigarrow 10[\text{in}_0 x] :: 11[\text{in}_1 y :: []] & \rightsquigarrow 10[z] :: 11[z' :: q] \\
 \rightsquigarrow 100[x] :: 11[\text{in}_1 y :: []] & \rightsquigarrow 10[z] :: 111[z', q] \\
 \rightsquigarrow \text{in}_1 x :: 11[\text{in}_1 y :: []] & \rightsquigarrow 10[z] :: 1110[\text{in}_1 y] :: 1111[[]] \\
 \rightsquigarrow \text{in}_1 x :: 111[\text{in}_1 y, []] & \rightsquigarrow 10[z] :: 1110[z'] :: 1111[q] \\
 \rightsquigarrow \text{in}_1 x :: 1110[\text{in}_1 y] :: 1111[[]] & \\
 \rightsquigarrow^4 \text{in}_1 x :: \text{in}_0 y :: [] &
 \end{array}$$

As one might expect, we have subject reduction. We need the following notion of *extension* to state it properly.

► **Definition 23** (Extension). Given two typing environments E, E' , we say that E' extends E if E and E' coincide after removing all nodes, and if all nodes in E' are either already in E , or are immediate successors of some nodes in E .

► **Proposition 24** (Subject reduction). *If $E \vdash M : e$ and $M \rightsquigarrow M'$, then $E' \vdash M' : e$ for some environment E' extending E .*

For instance, along the reductions on the left in Ex. 22, the antichain part of the typing environment evolves as follows: $\{\varepsilon\}, \{1\}, \{10, 11\}, \{100, 11\}, \{11\}, \{111\}, \{1110, 1111\}, \emptyset$.

Our objective now is to prove that well-typed programs terminate. For the sake of simplicity, we work in the sequel with the ‘leftmost innermost’ strategy: a redex $v[\vec{M}]$ is fired only when the programs in \vec{M} are irreducible and there are no other redexes to its left.

► **Definition 25** (Runs). The *run* of a program M is the sequence of nodes corresponding to the redexes fired during the (potentially infinite) leftmost innermost reduction of M .

► **Lemma 26**. *If $E \vdash M : e$ then every node w appears at most once in the run of M ; in this case we have that $w = uv$ for some nodes u, v with u in E and, for every prefix v' of v , uv' appears in the run of M before w .*

Proof. These are immediate consequences of Prop. 24. ◀

In particular, the run of a well-typed program has finitely many connected components. We finally obtain that well typed programs terminate, thanks to the validity criterion.

► **Proposition 27**. *If $E \vdash M : e$, then the run of M is finite.*

Proof. Suppose the run of M is infinite. Then by Lem. 26 and König’s Lemma one can extract an infinite branch of π which is contained in the run. By validity, this branch must eventually have a thread along a star formula f^* which is infinitely often principal. By analysis of the reduction rules, and thanks to the innermost strategy, we may find an infinite sequence of programs of type f^* whose sizes are strictly decreasing, which is impossible. ◀

4.2 Cut reduction

Our cut-elimination argument is driven by a standard set of cut reduction rules, which we do not have space to present in the main text. These include key and commutative cases, as usual, and are fully presented in [11, App. D]. To produce an infinite cut-free proof, we must show that we may produce proofs with arbitrarily large cut-free prefixes in a continuous manner. The main difficulty is to show that such a procedure is *productive*, i.e., eventually produces non-cut steps. To this end, we use the previous notion of ‘run’ to drive cut-reductions.

► **Definition 28** (Head). Let π be a proof of $\Gamma \rightarrow e$. The *head* of π , written $hd(\pi)$, is the run of the program $\varepsilon[\vec{x}]$ in π , where \vec{x} is a list of variables of the same length as Γ .

Note that the above program is well-typed in the appropriate environment. The head is a sequence of nodes, but we shall sometimes see it as the underlying sequence of programs. Also note that the nodes of a cut step appearing in the head correspond to program reductions where the redex is a cut (\rightsquigarrow_{cut}).

► **Definition 29** (Weight). The *weight* of a proof π , written $w(\pi)$, is the multiset of cut-reductions in its head, ordered by their distance to the end of the head.

► **Lemma 30.** *Let π' be obtained from a proof π by a cut-reduction. We have that:*

- (i) π' is a valid proof;
- (ii) $|hd(\pi')| \leq |hd(\pi)|$, where $|s|$ is the length of a sequence s ;
- (iii) if the reduced cut was the last \rightsquigarrow_{cut} step in $hd(\pi)$, then $w(\pi') < w(\pi)$.

Proof sketch. By case analysis; key cases strictly decrease the length of the head while it is only conserved by commutative cases. We list and discuss all cases in [11, App. D]; one of the two *-key cases is the following one:

$$\begin{array}{c} \frac{\Delta \rightarrow e \quad \Sigma \rightarrow e^*}{\text{cut} \frac{\Delta, \Sigma \rightarrow e^*}{\Gamma, \Delta, \Sigma, \Pi \rightarrow f}} \quad \frac{\Gamma, \Pi \rightarrow f \quad \Gamma, e, e^*, \Pi \rightarrow f}{*l \frac{\Gamma, e^*, \Pi \rightarrow f}{\Gamma, \Delta, \Sigma, \Pi \rightarrow f}}}{\mapsto} \quad \frac{\Delta \rightarrow e \quad \frac{\Sigma \rightarrow e^* \quad \Gamma, e, e^*, \Pi \rightarrow f}{\text{cut} \frac{\Gamma, e, \Sigma, \Pi \rightarrow f}{\Gamma, \Delta, \Sigma, \Pi \rightarrow f}}}{\text{cut} \frac{\Delta \rightarrow e}{\Gamma, \Delta, \Sigma, \Pi \rightarrow f}} \end{array}$$

If the reduced cut (with conclusion at v) occurs in the head of π then the heads of the two proofs only differ by the following subsequences, inside some evaluation context:

$$\begin{array}{ll} \rightsquigarrow_{cut_{e^*}} & v[\vec{M}, \vec{N}, \vec{O}, \vec{P}] \\ \rightsquigarrow & v1[\vec{M}, v0[\vec{N}, \vec{O}], \vec{P}] \\ \rightsquigarrow^n & v1[\vec{M}, v00[\vec{N} :: v01[\vec{O}], \vec{P}]] \\ \rightsquigarrow^o & v1[\vec{M}, N' :: O', \vec{P}] \\ \rightsquigarrow & v11[\vec{M}, N', O', \vec{P}] \end{array} \quad \begin{array}{ll} & v[\vec{M}, \vec{N}, \vec{O}, \vec{P}] \\ \rightsquigarrow_{cut_e} & v1[\vec{M}, v0[\vec{N}], \vec{O}, \vec{P}] \\ \rightsquigarrow^n & v1[\vec{M}, N', \vec{O}, \vec{P}] \\ \rightsquigarrow_{cut_{e^*}} & v11[\vec{M}, N', v10[\vec{O}], \vec{P}] \\ \rightsquigarrow^o & v11[\vec{M}, N', O', \vec{P}] \end{array}$$

(Note that the programs $\vec{M}, \vec{N}, \vec{O}, \vec{P}$ are irreducible due to the innermost strategy, and that $v00$ in the starting proof and $v0$ in the resulting one both point to the same subproof: $\pi_{v00} = \pi'_{v0}$.) The new head is shorter by one step, and the initial cut on e^* is replaced by two cuts which are closer to the end of the head.

Commutative cases do not always shorten the head, but either they move the cut closer to its end, or the head no longer visits it. For instance, when the left premiss of the reduced cut ends with a $-l$ step, the rule is:

$$\frac{\frac{\Delta, e, f, \Sigma \rightarrow g}{-l \frac{\Delta, ef, \Sigma \rightarrow g}{\Gamma, g, \Pi \rightarrow h}}}{\text{cut} \frac{\Gamma, \Delta, ef, \Sigma, \Pi \rightarrow h}}{\mapsto} \quad \frac{\frac{\Delta, e, f, \Sigma \rightarrow g \quad \Gamma, g, \Pi \rightarrow h}{\text{cut} \frac{\Gamma, \Delta, e, f, \Sigma, \Pi \rightarrow h}}}{-l \frac{\Gamma, \Delta, ef, \Sigma, \Pi \rightarrow h}}{\Gamma, \Delta, ef, \Sigma, \Pi \rightarrow h}}$$

If the head of π goes through the step $v[\vec{M}, \vec{N}, O, \vec{P}, \vec{Q}] \rightsquigarrow_{cut_{ef}} v1[\vec{M}, v0[\vec{N}, O, \vec{P}], \vec{Q}]$, then there are two cases to consider:

- either $O = \langle O_1, O_2 \rangle$ and the sequence continues with $v1[\vec{M}, v00[\vec{N}, O_1, O_2, \vec{P}], \vec{Q}]$; then in π' we get $v[\vec{M}, \vec{N}, O, \vec{P}, \vec{Q}] \rightsquigarrow v0[\vec{M}, \vec{N}, O_1, O_2, \vec{P}, \vec{Q}] \rightsquigarrow_{cut_{ef}} v01[\vec{M}, v00[\vec{N}, O_1, O_2, \vec{P}], \vec{Q}]$; the length is preserved and the cut has been pushed towards the end;
- or not, and the head of π' stops earlier, without visiting the cut on ef anymore, thus decreasing the weight.

For (iii), the assumption that the cut-reduction took place on the last cut of the head is used in some of the cases to ensure that the weights of the other cuts in the head do not increase (e.g., in some of the right $-r$ and $*-r_2$ cases). ◀

► **Proposition 31 (Productive cut-reduction).** *For a proof π , there exists a proof π' obtained from π by a sequence of cut-reductions, which does not start with a cut.*

Proof. By induction on the weight, reduce the last cut visited by the head until the head no longer contains any cut. The resulting proof cannot start with a cut, by definition. ◀

We can finally prove cut-elimination.

Proof of Thm. 15. Focus on a lowest cut, at node v , and apply Prop. 31 to the corresponding subproof π_v . By iterating this process, we obtain in the limit a cut-free preproof π' with the same conclusion as the starting one. Moreover, thanks to Lem. 30.(i), all heads of subproofs of π' are finite, so that π' is valid by Prop. 9: an infinite branch of $*\text{-}r_2$ steps would give rise to a subproof with an infinite head. ◀

5 Action algebras, Kleene lattices, and action lattices

We now consider extensions of Kleene algebra by residuals and meets, as axiomatised in [31] and [25]. We first extend the system LKA with the following rules, which are standard from substructural logic [28, 16]. We write LAL for the corresponding system.

$$\begin{array}{ccc} \frac{\Delta \rightarrow e \quad \Gamma, f, \Sigma \rightarrow g}{\Gamma, \Delta, e \setminus f, \Sigma \rightarrow g} \quad \backslash\text{-}l & \frac{\Delta \rightarrow e \quad \Gamma, f, \Sigma \rightarrow g}{\Gamma, f/e, \Delta, \Sigma \rightarrow g} \quad \text{/}\text{-}l & \frac{\Gamma, e_i, \Delta \rightarrow f}{\Gamma, e_1 \cap e_2, \Delta \rightarrow f} \quad \cap\text{-}l_i \quad i \in \{1, 2\} \\ \\ \frac{e, \Gamma \rightarrow f}{\Gamma \rightarrow e \setminus f} \quad \backslash\text{-}r & \frac{\Gamma, e \rightarrow f}{\Gamma \rightarrow f/e} \quad \text{/}\text{-}r & \frac{\Gamma \rightarrow e \quad \Gamma \rightarrow f}{\Gamma \rightarrow e \cap f} \quad \cap\text{-}r \end{array}$$

We define judgements as previously. Except for Thm. 33, the results below also hold for action algebras and Kleene lattices using the appropriate fragment of LAL. We prove soundness w.r.t. star-continuous models exactly like for Kleene algebra (Thm. 11).

► **Theorem 32 (Soundness).** *If $\text{LAL}^- \vdash^\infty e_1, \dots, e_n \rightarrow e$, then $\text{AL}^* \models e_1 \cdots e_n \leq e$.*

As announced in the introduction, regular proofs are sound for all (non-necessarily star-continuous) action lattices. We prove it using proof-theoretical arguments to translate every regular proof into an inductive proof from the axioms of action lattices.

► **Theorem 33 (Regular soundness).** *If $\text{LAL}^\omega \vdash e_1, \dots, e_n \rightarrow f$ then $\text{AL} \models e_1 \cdots e_n \leq f$.*

Proof. We prove the statement for all regular proofs in **-normal form*, where every backpointer points to a ‘validating’ $*\text{-}l$ -step: every infinite branch of the starting proof has a valid thread; since the proof is regular, this thread must be infinitely often principal for $*\text{-}l$ -step of some sequent of the branch; cut the infinite branch by using a backpointer the second time this sequent appears in the branch.

We proceed by induction on the number of simple cycles in such a proof π . The interesting case is when π ends with a $*\text{-}l$ step that is the target of a backpointer. Colour red all ancestors of its principal formula that are the same expression, say e^* . Let $\{\Gamma_i, e^*, \Delta_i \rightarrow f_i\}_{i \in I}$ be the set of all sequents in π with e^* principal and let $\{\pi_i^l : \Gamma_i, \Delta_i \rightarrow f_i\}_{i \in I}$ and $\{\pi_i^r : \Gamma_i, e, e^*, \Delta_i \rightarrow f_i\}_{i \in I}$ be the corresponding subproofs rooted at their left and right premisses, respectively.

Define expressions $g_i = \prod \Gamma_i$, $d_i = \prod \Delta_i$, $h_i = (g_i \setminus f_i) / d_i$, and $h = \bigcap_{i \in I} h_i$. For $i \in I$, construct proofs $\pi_i^{r'}$ from π_i^r by replacing each e^* by h , modifying critical steps as follows:

$$\frac{\Gamma_j, \Delta_j \rightarrow f_j \quad \Gamma_j, e, e^*, \Delta_j \rightarrow f_j}{\Gamma_j, e^*, \Delta_j \rightarrow f_j} \quad \text{*}\text{-}l \quad \mapsto \quad \left. \begin{array}{c} \frac{\frac{\Gamma_j, \Delta_j \rightarrow d_j}{\Delta_j \rightarrow d_j} \quad \frac{\Gamma_j, g_j \setminus f_j \rightarrow f_j}{\Gamma_j, g_j \setminus f_j \rightarrow f_j}}{\Gamma_j, h_j, \Delta_j \rightarrow f_j} \quad \text{/}\text{-}l \quad \frac{\Gamma_j, h_j, \Delta_j \rightarrow f_j}{\Gamma_j, h, \Delta_j \rightarrow f_j} \quad \cap\text{-}l \\ \frac{\Gamma_j \rightarrow g_j \quad f_j \rightarrow f_j}{\Gamma_j, g_j \setminus f_j \rightarrow f_j} \quad \backslash\text{-}l \quad \frac{\Gamma_j \rightarrow g_j}{\Gamma_j \rightarrow g_j} \quad \text{/}\text{-}r \end{array} \right\} \rho_j$$

Note that the proofs π_i^l and $\pi_i^{r'}$ have fewer simple cycles than π , so that by the induction hypothesis we have that $g_i d_i \leq f_i$ and $g_i e h d_i \leq f_i$ hold universally in action lattices, for all $i \in I$. From here we deduce $1 \leq h$ and $eh \leq h$ using the laws about residuals and conjunction. Thus we have $e^* \leq h$ by star induction (1). Finally note that following the above proof ρ_j we have in action lattices that $g_j h d_j \leq f_j$ and thus $g_j e^* d_j \leq f_j$. We conclude by choosing the appropriate j such that $(\Gamma_j, \Delta_j, f_j)$ is (Γ, Δ, f) . ◀

Note that we crucially rely on the presence of both residuals and meet to compute invariants for Kleene stars in the above proof, so that it does not immediately carry over to action algebras and Kleene lattices.

Conversely, the regular fragment of LAL (with cut) is complete for action lattices.

► **Theorem 34** (Regular completeness). *If $\text{AL} \models e \leq f$ then $\text{LAL} \vdash^\omega e \rightarrow f$.*

Proof. The axioms defining meet and residual immediately translate to finite derivations in LAL, so we may simply extend the proof of Thm. 14. ◀

Note that the regular fragment cannot be complete for star continuous models: a regular proof is a finite verifiable object and the equational theory of star-continuous action lattices is Π_1^0 -hard [7]. The full, non-regular system is however complete for star-continuous models:

► **Theorem 35** (Star-continuous completeness). *If $\text{AL}^* \models e \leq f$ then $\text{LAL} \vdash^\infty e \rightarrow f$.*

Proof. As for Thms. 14 and 34, consider the relation \equiv' defined by $e \equiv' f$ if $\text{LAL} \vdash^\infty e \rightarrow f$ and $\text{LAL} \vdash^\infty f \rightarrow e$. Expressions quotiented by this slightly larger relation also form an action lattice, which we prove star-continuous using the natural simulation of an ω -rule for Kleene star: combine proofs $(\pi_i)_{i \in \mathbb{N}}$ of the sequents $(\Gamma, e^i, \Delta \rightarrow f)_{i \in \mathbb{N}}$ as follows:

$$\frac{\frac{\frac{\frac{\Gamma, e, \Delta \rightarrow f}{\Gamma, e, \Delta \rightarrow f}}{\pi_0} \quad \frac{\frac{\frac{\Gamma, e, e, \Delta \rightarrow f}{\Gamma, e, e, \Delta \rightarrow f}}{\pi_1} \quad \frac{\frac{\Gamma, e, e, \Delta \rightarrow f}{\Gamma, e, e, \Delta \rightarrow f}}{\pi_2} \quad \dots}{\Gamma, e, e, \Delta \rightarrow f}}{\Gamma, e, e^*, \Delta \rightarrow f}}{\Gamma, e^*, \Delta \rightarrow f}}{\pi_i} \quad \text{*-l} \quad \text{*-l} \quad \text{*-l} \quad \dots \quad \text{*-l}$$

The remaining property to establish is cut-elimination: combined with Thm. 32 it gives soundness of proofs with cut w.r.t. star-continuous models, and combined with Thm. 35 it gives completeness of LAL^- w.r.t. these models.

6 Cut-elimination in LAL

The main alteration to the proof for LKA is that we need a more sophisticated notion of programs. We associate linear functions to residuals, and additive pairs to meets: a program for $e \cap f$ waits to see whether the environment wants a value for e or a value for f – but not both, and reacts accordingly. We thus extend the syntax of programs (Dfn. 16) to include λ -abstractions, which will be used for residuals, and a new kind of pairs for meets.

$$M, N ::= x \mid \star \mid \langle M, N \rangle \mid \text{in}_i M \mid [] \mid M :: N \mid \pi[\vec{M}] \mid \lambda x.M \mid \langle\langle M, N \rangle\rangle$$

The type system (Fig. 3) is extended by the following rules, where in the final rule, E_1 and E_2 are extensions of E (cf. Dfn. 23).

$$\frac{x : e, E \vdash M : f}{E \vdash \lambda x.M : e \setminus f} \quad \frac{E, x : e \vdash M : f}{E \vdash \lambda x.M : f/e} \quad \frac{E_1 \vdash M : e \quad E_2 \vdash N : f}{E \vdash \langle\langle M, N \rangle\rangle : e \cap f}$$

► **Lemma 36** (Substitution lemma). *If $E \vdash N : e$ and $F, x : e, F' \vdash M : f$ with F, E, F' defined, then $F, E, F' \vdash M\{N/x\} : f$, where $M\{N/x\}$ is M with x substituted by N .¹*

The following reductions are added, using the same conventions as in Dfn. 21:

$$\begin{array}{ll} \cap\text{-}l_i : v[\vec{M}, \langle\langle N_1, N_2 \rangle\rangle, \vec{P}] \rightsquigarrow v0[\vec{M}, N_i, \vec{P}] & \cap\text{-}r : v[\vec{M}] \rightsquigarrow \langle\langle v0[\vec{M}], v1[\vec{M}] \rangle\rangle \\ \backslash\text{-}l : v[\vec{M}, \vec{N}, \lambda x.F, \vec{P}] \rightsquigarrow v1[\vec{M}, F\{v0[\vec{N}]/x\}, \vec{P}] & \backslash\text{-}r : v[\vec{M}] \rightsquigarrow \lambda x.v0[x, \vec{M}] \\ /\text{-}l : v[\vec{M}, \lambda x.F, \vec{N}, \vec{P}] \rightsquigarrow v1[\vec{M}, F\{v0[\vec{N}]/x\}, \vec{P}] & /\text{-}r : v[\vec{M}] \rightsquigarrow \lambda x.v0[\vec{M}, x] \end{array}$$

One has to be careful about what we deem to be evaluation contexts: lambda abstractions and additive pairs are *not* considered evaluation contexts. This is crucial to obtain subject-reduction: otherwise some redexes duplicated by the $\cap\text{-}r$ rule can be active at the same time, thus breaking the property of Lem. 26 used in our termination proof that a given node appears at most once in the run of a program.

Despite this subtlety, Prop. 24 (subject reduction) and Prop. 27 (termination) are proved for this extended system exactly as in the Kleene algebra case – see [11, App. C]. It thus remains to show that the new cut reductions do not increase the length of heads, and strictly decrease the weight (Lem. 30). The key cases are easy: they strictly decrease the length and replace the cut by smaller ones. Amongst the commutative cases, some care is required when a right introduction rule appears on the right of the cut. For instance, for meet:

$$\text{cut} \frac{\Delta \rightarrow f \quad \cap\text{-}r \frac{\Gamma, f, \Sigma \rightarrow e_1 \quad \Gamma, f, \Sigma \rightarrow e_2}{\Gamma, f, \Sigma \rightarrow e_1 \cap e_2}}{\Gamma, \Delta, \Sigma \rightarrow e_1 \cap e_2} \mapsto \text{cut} \frac{\Delta \rightarrow f \quad \Gamma, f, \Sigma \rightarrow e_1 \quad \text{cut} \frac{\Delta \rightarrow f \quad \Gamma, f, \Sigma \rightarrow e_2}{\Gamma, \Delta, \Sigma \rightarrow e_1}}{\cap\text{-}r \frac{\Gamma, \Delta, \Sigma \rightarrow e_1 \quad \Gamma, \Delta, \Sigma \rightarrow e_2}{\Gamma, \Delta, \Sigma \rightarrow e_1 \cap e_2}}$$

If the head of π contains the sequence,

$$v[\vec{M}, \vec{N}, \vec{O}] \rightsquigarrow v1[\vec{M}, v0[\vec{N}], \vec{O}] \rightsquigarrow^n v1[\vec{M}, N', \vec{O}] \rightsquigarrow \langle\langle v10[\vec{M}, N', \vec{O}], v11[\vec{M}, N', \vec{O}] \rangle\rangle$$

where v is the reduced cut-node, then in the head of π' we just get:

$$v[\vec{M}, \vec{N}, \vec{O}] \rightsquigarrow \langle\langle v0[\vec{M}, \vec{N}, \vec{O}], v1[\vec{M}, \vec{N}, \vec{O}] \rangle\rangle$$

Here we see the need for $\langle\langle -, - \rangle\rangle$ not being an evaluation context: the computations involving \vec{N} would otherwise be duplicated, thus potentially increasing the length of the run. If the head of π never touches the produced additive pair, then the head of π' is strictly shorter, and the cut on $e_1 \cap e_2$ is not visited anymore. Otherwise, this pair can only be destroyed by a $\cap\text{-}l_i$ rule: $\langle\langle v10[\vec{M}, N', \vec{O}], v11[\vec{M}, N', \vec{O}] \rangle\rangle \rightsquigarrow v1i[\vec{M}, N', \vec{O}]$, and the head of π' can ‘catch up’ by doing:

$$\langle\langle v0[\vec{M}, \vec{N}, \vec{O}], v1[\vec{M}, \vec{N}, \vec{O}] \rangle\rangle \rightsquigarrow vi[\vec{M}, \vec{N}, \vec{O}] \rightsquigarrow vi1[\vec{M}, vi0[\vec{N}], \vec{O}] \rightsquigarrow^n vi1[\vec{M}, N', \vec{O}]$$

The size of the head has not changed, but the cut is closer to the end. The analogous case for residuals is similar since the creation of a λ -abstraction temporarily blocks reductions; other cases can be found in [11, App. D]. Finally, by the same argument as for Thm. 15 we obtain:

► **Theorem 37** (Cut elimination). *If $\text{LAL} \vdash^\infty \Gamma \rightarrow e$ then $\text{LAL}^- \vdash^\infty \Gamma \rightarrow e$.*

One useful application of this cut-elimination result is the following alternative proof of the upper bound result of Palka for star-continuous action lattices:

¹ More precisely, the occurrences of x selected by the typing derivation of M .

► **Corollary 38** (Palka [29]). AL^* is in Π_1^0 .

Proof. We say that a sequent $\Gamma \rightarrow e$ has a d -derivation, for $d \in \mathbb{N}$, if there is a LAL^- derivation ending in $\Gamma \rightarrow e$ for which each branch has length d , or otherwise terminates at a correct initial sequent in length $< d$. To avoid validity issues, we assume that the left premiss of every $*-r_2$ step has nonempty antecedent, so that all preproofs become valid without sacrificing provability (cf. Prop. 9). We define a Π_1^0 predicate $\text{Prov}(\Gamma \rightarrow e)$ as $\forall d \in \mathbb{N}$. “there is a d -derivation of $\Gamma \rightarrow e$ ”. Notice that this is indeed Π_1^0 since the size of a d -derivation is exponentially bounded. Furthermore, if $\text{Prov}(\Gamma \rightarrow e)$ then, by the infinite pigeonhole principle, we may recover an infinite proof of $\Gamma \rightarrow e$, by inductively choosing premisses resulting in larger derivations that nonetheless prefix infinitely many d -derivations. ◀

7 Conclusions

We presented a simple sequent system LKA that admits non-wellfounded proofs and showed it to be sound and complete for Kleene algebra, KA, by consideration of the free model of rational languages. We showed that its regular fragment is already complete, in the presence of cut, by a direct simulation of KA. We also gave a cut-elimination result for LKA, obtaining an alternative proof of completeness of its cut-free fragment.

We were able to generalise these arguments to an extended system LAL of Kleene algebras with residuals and meets, resulting in a sound and complete cut-free system for the equational theory of star-continuous action lattices, AL^* . Thanks to the subformula property for cut-free proofs, this also gives us proof-theoretical characterisations of star-continuous action algebras and Kleene lattices. This yields alternative proofs of several results of Palka [29], namely conservativity of AL^* over its fragments, as well as their membership in Π_1^0 .

Finally, we characterised the theory of all action lattices by just the regular proofs of LAL. Whether the equational theory of action lattices is decidable remains open. It would be interesting to see if techniques such as interpolants for our system LAL, or a characterisation of the image of cut-elimination on cut-free proofs, might yield decidability.

It would be natural to consider systems which are commutative and/or contain arbitrary fixed points, bringing the subject matter closer to that of [13]. We would however not be able to arrive at a similar subformula property once fixed point formulae are allowed to contain meets and residuals, since this property is essentially thanks to the presence of only ‘positive’ connectives in KA, from the point of view of focusing [2].

References

- 1 C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: semantic foundations for networks. In *Proc. POPL*, pages 113–126. ACM, 2014. doi:10.1145/2535838.2535862.
- 2 J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.
- 3 A. Angus and D. Kozen. Kleene algebra with tests and program schematology. Technical Report TR2001-1844, CS Dpt., Cornell University, July 2001. URL: <http://hdl.handle.net/1813/5831>.
- 4 Maurice Boffa. Une condition impliquant toutes les identités rationnelles. *Informatique Théorique et Applications*, 29(6):515–518, 1995. URL: http://www.numdam.org/article/ITA_1995__29_6_515_0.pdf.


- 5 Thomas Braibant and Damien Pous. An efficient Coq tactic for deciding Kleene algebras. In *Proc. 1st ITP*, volume 6172 of *Lecture Notes in Computer Science*, pages 163–178. Springer Verlag, 2010. doi:10.1007/978-3-642-14052-5_13.
- 6 Samuel R. Buss. An introduction to proof theory. *Handbook of proof theory*, 137:1–78, 1998.
- 7 Wojciech Buszkowski. On action logic: Equational theories of action algebras. *J. Log. Comput.*, 17(1):199–217, 2007. doi:10.1093/logcom/ex1036.
- 8 Pierre Clairambault. Least and greatest fixpoints in game semantics. In *Proc. FoSSaCS*, pages 16–31, 2009. doi:10.1007/978-3-642-00596-1_3.
- 9 J. H. Conway. *Regular algebra and finite machines*. Chapman and Hall, 1971.
- 10 Anupam Das and Damien Pous. A cut-free cyclic proof system for Kleene algebra. In *Proc. TABLEAUX*, volume 10501 of *Lecture Notes in Computer Science*, pages 261–277. Springer Verlag, 2017. doi:10.1007/978-3-319-66902-1_16.
- 11 Anupam Das and Damien Pous. Non-Wellfounded Proof Theory For (Kleene+Action)(Algebras+Lattices). Full version of this extended abstract, 2018. URL: <https://hal.archives-ouvertes.fr/hal-01703942>.
- 12 H. Doornbos, R. Backhouse, and J. van der Woude. A calculational approach to mathematical induction. *Theoretical Computer Science*, 179(1-2):103–135, 1997. doi:10.1016/S0304-3975(96)00154-5.
- 13 Amina Doumane, David Baelde, and Alexis Saurin. Infinitary proof theory: the multiplicative additive case. In *CSL*, volume 62 of *LIPICs*, pages 42:1–42:17, 2016. doi:10.4230/LIPICs.CSL.2016.42.
- 14 Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In *Proc. CSL*, volume 23 of *LIPICs*, pages 248–262, 2013. doi:10.4230/LIPICs.CSL.2013.248.
- 15 Alain Frisch and Luca Cardelli. Greedy regular expression matching. In *Proc. ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 618–629. Springer Verlag, 2004. doi:10.1007/978-3-540-27836-8_53.
- 16 N. Galatos, P. Jipsen, T. Kowalski, and H. Ono. *Residuated Lattices: An Algebraic Glimpse at Substructural Logics*. Elsevier, 2007.
- 17 J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
- 18 Fritz Henglein and Lasse Nielsen. Regular expression containment: coinductive axiomatization and computational interpretation. In *Proc. POPL 2011*, pages 385–398. ACM, 2011. doi:10.1145/1926385.1926429.
- 19 C. A. R. Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent Kleene Algebra. In *Proc. CONCUR*, volume 5710 of *Lecture Notes in Computer Science*, pages 399–414. Springer Verlag, 2009. doi:10.1007/978-3-642-04081-8_27.
- 20 P. Jipsen. From semirings to residuated Kleene lattices. *Studia Logica*, 76(2):291–303, 2004. doi:10.1023/B:STUD.0000032089.54776.63.
- 21 S. C. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–41. Princeton University Press, 1956. URL: http://www.rand.org/pubs/research_memoranda/2008/RM704.pdf.
- 22 D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Log.*, 1(1):60–76, 2000. doi:10.1145/343369.343378.
- 23 D. Kozen and M.-C. Patron. Certification of compiler optimizations using Kleene algebra with tests. In *Proc. CL2000*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 568–582. Springer Verlag, 2000. doi:10.1007/3-540-44957-4_38.
- 24 Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. In *Proc. LICS*, pages 214–225. IEEE, 1991. doi:10.1109/LICS.1991.151646.

- 25 Dexter Kozen. On action algebras. In J. van Eijck and A. Visser, editors, *Logic and Information Flow*, pages 78–88. MIT Press, 1994.
- 26 A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *Journal of Algebraic Reasoning*, 49(1):95–106, 2012. doi:10.1007/s10817-011-9223-4.
- 27 D. Krob. Complete systems of B-rational identities. *Theoretical Computer Science*, 89(2):207–343, 1991. doi:10.1016/0304-3975(91)90395-I.
- 28 Joachim Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65:154–170, 1958.
- 29 Ewa Palka. An infinitary sequent system for the equational theory of *-continuous action lattices. *Fundamenta Informaticae*, pages 295–309, 2007. URL: <http://iospress.metapress.com/content/r5p53611826876j0/>.
- 30 Damien Pous. Kleene Algebra with Tests and Coq tools for while programs. In *Proc. ITP*, volume 7998 of *Lecture Notes in Computer Science*, pages 180–196. Springer Verlag, 2013. doi:10.1007/978-3-642-39634-2_15.
- 31 V. Pratt. Action logic and pure induction. In *Proc. JELIA*, volume 478 of *Lecture Notes in Computer Science*, pages 97–120. Springer Verlag, 1990. doi:10.1007/BFb0018436.
- 32 Volodimir Nikiforovich Redko. On defining relations for the algebra of regular events. *Ukrainskii Matematicheskii Zhurnal*, 16:120–126, 1964.
- 33 Kurt Schütte. *Proof Theory*. Grundlehren der mathematischen Wissenschaften 225. Springer Berlin Heidelberg, 1977. Translation of Beweistheorie, 1968.
- 34 Christian Wurm. Kleene algebras, regular languages and substructural logics. In *Proc. GandALF*, EPTCS, pages 46–59, 2014. doi:10.4204/EPTCS.161.7.

Symmetric Circuits for Rank Logic

Anuj Dawar

Department of Computer Science and Technology, University of Cambridge, UK
anuj.dawar@cl.cam.ac.uk

 <https://orcid.org/0000-0003-4014-8248>

Gregory Wilsenach¹

Department of Computer Science and Technology, University of Cambridge, UK
gregory.wilsenach@cl.cam.ac.uk

Abstract

Fixed-point logic with rank (FPR) is an extension of fixed-point logic with counting (FPC) with operators for computing the rank of a matrix over a finite field. The expressive power of FPR properly extends that of FPC and is contained in P, but it is not known if that containment is proper. We give a circuit characterization for FPR in terms of families of symmetric circuits with rank gates, along the lines of that for FPC given by [Anderson and Dawar 2017]. This requires the development of a broad framework of circuits in which the individual gates compute functions that are not symmetric (i.e., invariant under all permutations of their inputs). This framework also necessitates the development of novel techniques to prove the equivalence of circuits and logic. Both the framework and the techniques are of greater generality than the main result.

2012 ACM Subject Classification Theory of computation → Circuit complexity, Theory of computation → Finite Model Theory, Theory of computation → Complexity theory and logic

Keywords and phrases fixed-point logic with rank, circuits, symmetric circuits, uniform families of circuits, circuit characterization, circuit framework, finite model theory, descriptive complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.20

Related Version A full version of this paper is available at [7], <https://arxiv.org/abs/1804.02939>.

1 Introduction

The study of extensions of fixed-point logics plays an important role in the field of descriptive complexity theory. In particular, fixed-point logic with counting (FPC) has become a reference logic in the search for a logic for polynomial-time (see [2]). In this context, Anderson and Dawar [1] provide an interesting characterization of the expressive power of FPC in terms of circuit complexity. They show that the properties expressible in this logic are exactly those that can be decided by polynomially-uniform families of circuits (with threshold gates) satisfying a natural *symmetry* condition. Not only does this illustrate the robustness of FPC as a complexity class within P by giving a distinct and natural characterization of it, it also demonstrates that the techniques for proving inexpressibility in the field of finite model theory can be understood as lower-bound methods against a natural circuit complexity class. This raises an obvious question (explicitly posed in the concluding section of [1]) of how to obtain circuit characterizations of logics more expressive than FPC, such as choiceless polynomial time (CPT) and fixed-point logic with rank (FPR). It is this last question that we address in this paper.

¹ Funding provided by the Gates Cambridge Scholarship.



Fixed-point logic with rank extends the expressive power of FPC by means of operators that allow us to define the rank of a matrix over a finite field. Such operators are natural extensions of counting – counting the dimension of a definable vector space rather than just the size of a definable set. At the same time they make the logic rich enough to express many of the known examples that separate FPC from P. Rank logics were first introduced in [5]. The version FPR we consider here is that defined by Grädel and Pakusa [9] where the prime characteristic is a parameter to the rank operator, and we do not have a distinct operator for each prime number. Formal definitions of these logics are given in Section 2. We give a circuit characterization, in terms of symmetric circuits, of FPR. One might think, at first sight, that this is a simple matter of extending the circuit model with gates for computing the rank of a matrix. It turns out, however, that the matter is not so simple as the symmetry requirement interacts in surprising ways with such rank gates. It requires a new framework for defining classes of such circuits, which yields remarkable new insights.

The word *symmetry* is used in more than one sense in the context of circuits (and also in this paper). We say that a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is symmetric if the value of the function on a string s is determined by the number of 1s in s . In other words, f is invariant under *all* permutations of its input. In contrast, when we consider the input to a Boolean function to be the adjacency matrix of an n -vertex graph, for example, and $f : \{0, 1\}^{\binom{n}{2}} \rightarrow \{0, 1\}$ decides a graph property, then f is invariant under all permutations of its input induced by permutations of the n vertices of the graph. We call such a function *graph-invariant*. More generally, for a relational vocabulary τ and a standard encoding of n -element τ -structures as strings over $\{0, 1\}$, we can say that function taking such strings as input is τ -invariant if it is invariant under permutations induced by the n elements. A circuit C computing such an invariant function is said to be *symmetric* if every permutation of the n elements extends to an automorphism of C . It is families of symmetric circuits in this sense that characterize FPC in [1]. The restriction to symmetric circuits arises naturally in the study of logics and has appeared previously under the names of generic circuits in the work of [8] and explicitly order-invariant circuits in the work of Otto [15]. In this paper, we use the word “symmetric”, and context is used to distinguish the meaning of the word as applied to circuits from its meaning as applied to Boolean functions.

The main result of [1] says that the properties of τ -structures definable in FPC are exactly those that can be decided by P-uniform families of symmetric circuits using AND, OR, NOT and majority gates. Note that each of these gates itself computes a Boolean function that is symmetric in the strong sense identified above. On the other hand, a gate for computing a rank threshold function, e.g. one that takes as input a $n \times n$ matrix and outputs 1 if the rank of the matrix is greater than a threshold t , is *not* symmetric. In our circuit characterization of FPR we necessarily have to consider such non-symmetric gates. Indeed, we can show that P-uniform families of symmetric circuits using gates for *any* symmetric functions do not take us beyond the power of FPC. This is a further illustration of the robustness of FPC. In order to go beyond it, we need to introduce gates for Boolean functions that are not symmetric. We construct a systematic framework for including functions computing τ -invariant functions for arbitrary multi-sorted relational vocabularies τ in Section 3. We also explore what it means for such circuits to be symmetric.

The proof of the circuit characterization of FPC relies on the *support theorem* proved in [1]. This establishes that for any P-uniform family of circuits using AND, OR, NOT and majority gates there is a constant k such that every gate has a support of size at most k . That is to say that we can associate with every gate g in the circuit C_n (the circuit in the family that works on n -element structures) a subset X of $[n]$ of size at most k such that any

permutation of $[n]$ fixing X pointwise extends to an automorphism of C_n that fixes g . This theorem is crucial to the translation of the family of circuits into a formula of FPC, which is the difficult (and novel) direction of the equivalence. In attempting to do the same with circuits that now use rank-threshold gates we are faced with the difficulty that the proof of the support theorem in [1] relies in an essential way on the fact that the Boolean function computed at each gate is symmetric. We are able to overcome this difficulty and prove a support theorem for circuits with rank gates but this requires substantial, novel technical machinery.

Another crucial ingredient in the proof of Anderson and Dawar is that we can eliminate redundancy in the circuit C_n by making it *rigid*. That is, we can ensure that the *only* automorphisms of C_n are those that are induced by permutations of $[n]$. Here we face the difficulty that identifying the symmetries and eliminating redundancy in a circuit that involves gates computing τ -invariant functions requires us to solve the isomorphism problem for τ -structures. This is a hard problem (or, at least, one that we do not know how to solve efficiently) even when the τ -structures are 0-1-matrices. We overcome this difficulty by placing a further restriction on circuits that we call *transparency*. Circuits satisfying this condition have the property that their lack of redundancy is transparent.

In the characterization of FPC, the translation from formulas into families of circuits is easy and, indeed, standard. In our case, we have to show that formulas of FPR translate into uniform families of circuits using rank-threshold gates that are symmetric and transparent. This is somewhat more involved technically and presented in Section 5. Finally, with all these tools in place, the translation of such P-uniform families of circuits into formulas of FPR given in Section 6 completes the characterization. This still requires substantial new techniques. The translation of circuits to formulas in [1] relies on the fact that in order to evaluate a gate computing a symmetric Boolean function, it suffices to count the number of inputs that evaluate to true and there is a bijection between the orbits of a gate and tuple assignments to its support. When counting is no longer sufficient, this bijection has to preserve more structure and demonstrating this in the case of matrices requires new insight.

Space limitations prevent us from giving details of proofs. These and much more detail can be found in the full version of this paper [7].

2 Background

We write \mathbf{Sym}_X to denote the group of all permutations of the set X . Let G be a group and X be a set on which a group action is defined and let $S \subseteq X$. Let $\mathbf{Stab}_G(S) := \{\pi \in G : \forall s \in S, \pi(s) = s\}$. For $n \in \mathbb{N}$ we write \mathbf{Sym}_n to abbreviate $\mathbf{Sym}_{[n]}$ and write $\mathbf{Stab}_n(S)$ to abbreviate $\mathbf{Stab}_{\mathbf{Sym}_n}(S)$. In the event that the group is obvious from context we omit the subscript entirely. We let A^B denote the set of injections from the set B to the set A .

2.1 Logic

A *vocabulary* is a finite sequence of relation symbols (R_1, \dots, R_k) , each of which has a fixed *arity*. We let $r_i \in \mathbb{N}$ denote the arity of the relation symbol R_i . A *many-sorted vocabulary* is a tuple of the form (R, S, ν) , where R is a relational vocabulary, S is a finite sequence of *sort* symbols, and ν is a function that assigns to each $R_i \in R$ a tuple $\nu(R_i) := (s_1, \dots, s_{r_i})$, where for each $j \in [r_i]$, $s_j \in S$. We call $\nu(R_i)$ the *type* of R_i . A τ -*structure* \mathcal{A} is a tuple $(U, R_1^{\mathcal{A}}, \dots, R_k^{\mathcal{A}})$ where $U = \uplus_{s \in S} U_s$ is a disjoint union of non-empty sets and is called the *universe* of \mathcal{A} , and for all $i \in [k]$, $R_i^{\mathcal{A}} \subseteq U_{s_1} \times \dots \times U_{s_{r_i}}$, where $(s_1, \dots, s_{r_i}) = \nu(R_i)$. The size of \mathcal{A} , denoted by $|\mathcal{A}|$, is the cardinality of its universe. All structures in this paper are finite.

We assume the reader is familiar with first-order logic (FO), inflationary fixed-point logic (FP), fixed-point logic with counting (FPC), and first-order logic with counting quantifiers (FOC). For details on these logics please see [10, 13].

2.2 Rank Logic

Let $\text{FPR}[\tau]$ denote *fixed-point logic with rank* over the vocabulary τ . FPR extends FP with an operator that denotes the rank of a definable matrix over a finite field, as well as other mechanisms for reasoning about quantity. Each variable in a formula of FPR is either a *number* or *vertex* variable, with vertex variables interpreted by elements of the universe and number variables interpreted by natural numbers. All atomic formulas in $\text{FP}[\tau]$ are atomic formulas in $\text{FPR}[\tau]$. We say that t is a *number term* if t is a number variable or if t is an application of the *rank operator*, i.e. $t := [\mathbf{rk}(\vec{x}, \vec{v} \leq \vec{t}, \vec{y}, \vec{\mu} \leq \vec{s}, \pi \leq \eta). \phi]$, where ϕ is a number term or formula, \vec{t} and \vec{s} are tuples of number terms bounding the sequences of number variables $\vec{\mu}$ and \vec{v} , and η is a number term bounding the number variable π . If t_1 and t_2 are number terms then $t_1 < t_2$ and $t_1 = t_2$ are atomic formulas. The formulas of FPR are formed by closing the set of atomic formulas under the usual Boolean connectives, the first-order quantifiers, and the fixed-point operator. When quantifying over number variables we only allow bounded quantification. Second-order variables, such as those that appear in a fixed-point application, may have mixed-type. For more detail on the syntax and semantics of FPR please see [9].

Let $\text{FOR}[\tau]$ be the set of formulas in $\text{FPR}[\tau]$ without an application of the fixed-point operator. We define for each prime p , and natural number r , a rank quantifier \mathbf{rk}_p^r , such that $\mathbf{rk}_p^r \vec{x} \vec{y}. \phi$ is interpreted as $[\mathbf{rk}(\vec{x}, \vec{y}, \pi). \phi] \geq t_r$, where π is assigned to p and t_r is a number term that evaluates to r . Let \mathcal{R} be the set of all such quantifiers and $\text{FO}+\text{rk}[\tau]$ be the closure of $\text{FO}[\tau]$ under \mathcal{R} . For more details on rank quantifiers see [5].

3 Generalizing Symmetric Circuits

A *Boolean basis* is a set of Boolean functions. We always use \mathbb{B} to denote a basis. Let \mathbb{B}_{std} denote the *standard basis* containing the Boolean functions computing AND, OR and NOT for each arity. Let \mathbb{B}_{maj} denote the *majority basis*, i.e. the extension of \mathbb{B}_{std} with functions computing majority for each arity.

A Boolean circuit C over a basis \mathbb{B} is a directed acyclic graph in which each internal gate g is labelled with a function $f_g : \{0, 1\}^q \rightarrow \{0, 1\} \in \mathbb{B}$ where q is the fan-in of g . Notice that if f_g were allowed to be arbitrary then an order would need to be imposed on the children of g to ensure unambiguous evaluation. As such, the usual notion of a circuit as a directed acyclic graph with no structure on the children of any gate g implicitly assumes that f_g is invariant under all permutations of its inputs – i.e. f_g is a symmetric function. It is easy to see that the standard basis and majority basis contain only symmetric functions.

Anderson and Dawar [1] characterize the expressive power of FPC in terms of symmetric circuits over the majority basis. This circuit model cannot be strengthened by extending the basis by symmetric functions (see [7]). As our ultimate aim is a circuit characterisation of FPR, which is strictly more expressive than FPC, we would like to consider circuits defined over bases containing non-symmetric Boolean functions. In particular, we are interested in bases containing rank-threshold functions – i.e. functions that take in a matrix and decide if the matrix understood as having entries in some prime field has rank less than some threshold. While these functions are not symmetric in the full sense, they are symmetric in the sense of being invariant under row-column permutations.

To lead up to this we first develop a general framework of structured Boolean functions. These are functions whose inputs naturally encode τ -structures, rather than just matrices or strings, and where the output is invariant under the natural symmetries of such structures. We therefore define symmetric circuits in a general form where gates can be labelled by *isomorphism-invariant* structured functions.

3.1 Structured Functions

Let X be a finite set and $F : \{0, 1\}^X \rightarrow \{0, 1\}$. It is common to consider Boolean functions that take strings as input, which would correspond to taking $X = [n]$ for some $n \in \mathbb{N}$. The natural notion of symmetry for such functions is invariance under arbitrary permutations of X , i.e. the usual notion of a symmetric (Boolean) function. Alternatively, we might want to consider Boolean functions that take in more complex algebraic structures as input, which would involve selecting an index set X such that the input to the function encodes an appropriate structure. For example, if we are interested in functions that take directed graphs as inputs we would let $X = V^2$ for some vertex set V . We notice that in this case the natural symmetry condition would not be invariance under arbitrary permutation, but rather invariance under the action of a permutation of V .

In this subsection we formalise this notion and define a class of functions that take in many-sorted structures and define a natural symmetry notion for such functions. Let $\tau := (R, S, \nu)$ be a many-sorted vocabulary and let $D := \bigsqcup_{s \in S} D_s = \{(s, d) : d \in D_s\}$, be a disjoint union of non-empty sets. Let $\text{str}(\tau, D)$ be the τ -structure with universe D and such that every relation is full (i.e. contains all possible tuples). We let $\text{ind}(\tau, D)$ be the disjoint union of all the relations in $\text{str}(\tau, D)$, i.e. $\text{ind}(\tau, D) = \bigsqcup_{R_i \in R} R_i^{\text{str}(\tau, D)} := \{(\vec{a}, R_i) : \vec{a} \in R_i^{\text{str}(\tau, D)}, R_i \in R\}$. We often abbreviate $(\vec{a}, R_i) \in \text{ind}(\tau, D)$ by \vec{a}_{R_i} . We call $\text{ind}(\tau, D)$ the *index defined by* (τ, D) .

We think of $\text{ind}(\tau, D)$ as containing all those tuples that may appear in a relation in a τ -structure or, equivalently, the collection of ground atoms in the vocabulary τ with elements from the domain D . In this way each element of $\{0, 1\}^{\text{ind}(\tau, D)}$ encodes a τ -structure with universe D . We call a function $F : \{0, 1\}^{\text{ind}(\tau, D)} \rightarrow \{0, 1\}$ a (τ, D) -*structured function*, or just a *structured function*, and we call τ and D the *vocabulary* and *universe* of F , and denote them by $\text{voc}(F)$ and $\text{unv}(F)$ respectively. We call $\text{ind}(\tau, D)$ the *index* of F , and denote it by $\text{ind}(F)$. We see that F defines a class of τ -structures with universe D . We are especially interested in structured functions that are symmetric in some sense, and hence decide properties of τ -structures, i.e. isomorphism-closed classes of structures.

Let H be a set. We think of a function $f : \text{ind}(\tau, D) \rightarrow H$ as defining a labelling of $\text{str}(\tau, D)$ by H and we identify f with this labelled instance of $\text{str}(\tau, D)$. Let $f : \text{ind}(\tau, D) \rightarrow H$ and $g : \text{ind}(\tau, D') \rightarrow H$. We say that f and g are *isomorphic* if there is an isomorphism $\pi : \text{str}(\tau, D) \rightarrow \text{str}(\tau, D')$ such that $f(\vec{a}_R) = g((\pi\vec{a})_R)$ for all $\vec{a}_R \in \text{ind}(\tau, D)$. In other words, f and g are isomorphic if, and only if, they are isomorphic as (labelled) structures. Notice that if $H = \{0, 1\}$ then f and g define τ -structures and f and g are isomorphic if, and only if, the τ -structures they define are isomorphic.

We say that $F : \{0, 1\}^{\text{ind}(\tau, D)} \rightarrow \{0, 1\}$ is *isomorphism-invariant* if for all $f, g : \text{ind}(\tau, D) \rightarrow \{0, 1\}$ whenever f and g are isomorphic then $F(f) = F(g)$.

3.2 Symmetric Circuits

We now generalise the circuit model in [1] in order to allow for circuits to be defined over bases that include non-symmetric (structured) functions. In this model each gate g is not only associated with an element of the basis, but also with a labelling function. This labelling

function maps an appropriate set of labels (i.e. the index of the structured function associated with g) to the input gates of g . In concord with this generalisation, we also update the circuit-related notions from [1], e.g. circuit automorphisms, symmetry, etc. Moreover, we briefly discuss some of the important complications introduced by our generalisation, and introduce some of the important tools we use to address these complications.

► **Definition 1** (Circuits on Structures). Let \mathbb{B} be a basis of structured functions and ρ be a relational vocabulary, we define a (\mathbb{B}, ρ) -circuit C of order n computing a q -ary query Q as a structure $\langle G, \Omega, \Sigma, \Lambda, L \rangle$.

- G is called the set of gates of C .
- Ω is an injective function from $[n]^q$ to G . The gates in the image of Ω are called the output gates. When $q = 0$, Ω is a constant function mapping to a single output gate.
- Σ is a function from G to $\mathbb{B} \uplus \rho \uplus \{0, 1\}$ such that $|\Sigma^{-1}(0)| \leq 1$ and $|\Sigma^{-1}(1)| \leq 1$. Those gates mapped to $\rho \uplus \{0, 1\}$ are called input gates, with those mapped to ρ called relational gates and those mapped to $\{0, 1\}$ called constant gates. Those gates mapped to \mathbb{B} are called internal gates.
- Λ is a sequence of injective functions $(\Lambda_{R_i})_{R_i \in \mathcal{R}}$ such that Λ_{R_i} maps each relational gate g with $\Sigma(g) = R_i$ to the tuple $\Lambda_{R_i}(g) \in [n]^{r_i}$. When no ambiguity arises we write $\Lambda(g)$ for $\Lambda_{R_i}(g)$.
- L associates with each internal gate g a function $L(g) : \text{ind}(\Sigma(g)) \rightarrow G$ such that if we define a relation $W \subseteq G^2$ by $W(h_1, h_2)$ iff h_2 is an internal gate and h_1 is in the image of $L(h_2)$, then (G, W) is a directed acyclic graph.

The definition requires some explanation. Each gate in G computes a function of its inputs and the relation W on G is the set of “wires”. That is, $W(h, g)$ indicates that the value computed at h is an input to g . However, since the functions are structured, we need more information on the set of inputs to g and this is provided by the labelling L . $\Sigma(g)$ tells us what the function computed at g is, and thus $\text{ind}(\Sigma(g))$ tells us the structure on the inputs and $L(g)$ maps this to the set of gates that form the inputs to g . We let $H_g = \{h \in G : W(h, g)\}$ denote the set of inputs to the gate g . We let $\text{unv}(g)$ denote the universe of $\Sigma(g)$. We call a gate g a *symmetric gate* if $\Sigma(g)$ is a symmetric function and g a *non-symmetric gate* otherwise.

Let ρ be a relational vocabulary, \mathcal{A} be a ρ -structure with universe U of size n , and $\gamma \in [n]^U$. Let $\gamma\mathcal{A}$ be the structure with universe $[n]$ formed by mapping the elements of U in accordance with γ . The evaluation of a (\mathbb{B}, ρ) -circuit C of order n computing a q -ary query Q proceeds by recursively evaluating the gates in the circuit. The evaluation of the gate g for the bijection γ and input structure \mathcal{A} is denoted by $C[\gamma\mathcal{A}](g)$, and is given as follows:

- if g is a constant gate then it evaluates to the bit given by $\Sigma(g)$,
- if g is a relational gate then g evaluates to true iff $\gamma\mathcal{A} \models \Sigma(g)(\Lambda(g))$, and
- if g is an internal gate let $L^{\gamma\mathcal{A}}(g) : \text{ind}(g) \rightarrow \{0, 1\}$ be defined by $L^{\gamma\mathcal{A}}(g)(x) = C[\gamma\mathcal{A}](L(g)(x))$, for all $x \in \text{ind}(g)$. Then g evaluates to true if, and only if, $\Sigma(g)(L^{\gamma\mathcal{A}}(g)) = 1$.

We say that C defines the q -ary query $Q \subseteq U^q$ under γ where $\vec{a} \in Q$ if, and only if, $C[\gamma\mathcal{A}](\Omega(\gamma\vec{a})) = 1$.

We now define a circuit automorphism for a circuit.

► **Definition 2** (Automorphism). Let $C = \langle G, \Omega, \Sigma, \Lambda, L \rangle$ be a (\mathbb{B}, τ) -circuit of order n computing a q -ary query, and where \mathbb{B} is a basis of isomorphism-invariant structured functions. Let $\sigma \in \text{Sym}_n$ and $\pi : G \rightarrow G$ be a bijection such that

- for all output tuples $x \in [n]^q$, $\pi\Omega(x) = \Omega(\sigma x)$,
- for all gates $g \in G$, $\Sigma(g) = \Sigma(\pi g)$,
- for each relational gate $g \in G$, $\sigma\Lambda(g) = \Lambda(\pi g)$, and
- For each pair of gates $g, h \in G$, we have $W(h, g)$ if, and only if, $W(\pi h, \pi g)$ and for each internal gate g we have that $L(\pi g)$ and $\pi \cdot L(g)$ are isomorphic.

We call π an *automorphism* of C , and we say that σ *extends to an automorphism* π . The group of automorphisms of C is called $\mathbf{Aut}(C)$.

We are particularly interested in circuits that have the property that *every* permutation in \mathbf{Sym}_n extends to an automorphism of the circuit.

► **Definition 3** (Symmetry). A circuit C on structures of size n is called *symmetric* if every $\sigma \in \mathbf{Sym}_n$ extends to an automorphism on C .

Suppose C does not contain a relational gate labelled by a relation symbol with non-zero arity. In that case C computes a constant function. For this reason, we always assume a circuit contains at least one relational gate with non-zero arity. Now, by assumption there exists a relational gate in C such that some element of $[n]$ appears in the tuple labelling that gate. By symmetry it follows that every element of $[n]$ appears in a tuple labelling a relational gate in C . It follows that no two distinct elements of \mathbf{Sym}_n agree on all input gates and so we can associate with each $\pi \in \mathbf{Aut}(C)$ a unique $h(\pi) \in \mathbf{Sym}_n$ that it extends and it is easily seen that $h : \mathbf{Aut}(C) \rightarrow \mathbf{Sym}_n$ is a surjective group homomorphism. If h is also injective then we have that each element of σ extends uniquely to an automorphism of the circuit. In this case we say that a circuit has *unique extensions*.

► **Definition 4.** We say that a circuit C of order n has *unique extensions* if for every $\sigma \in \mathbf{Sym}_n$ there is at most one $\pi_\sigma \in \mathbf{Aut}(C)$ such that π_σ extends σ .

Many important technical tools, e.g. the support theorem, are only applicable to circuits with unique extensions. It is for this reason that a notion of a *rigid* circuit is introduced in [1]. Such circuits have unique extensions and it is shown that a symmetric circuit over the basis $\mathbb{B}_{\mathbf{maj}}$ can be converted in polynomial-time to an equivalent rigid one.

We should like to develop a property analogous to rigidity for our framework, as well as a similar polynomial-time translation. However, in our framework the value a gate g computes depends not just on the *set* of gates input to g but also on the structure of this set. This structure must be preserved by the action of an automorphism, and so we require that if π is an automorphism that maps g to g' then $\pi L(g)$ and $L(g')$ are isomorphic. Following from this observation, it can be shown that deciding if a function on the circuit is an automorphism, and indeed deciding almost any symmetry-related property, for circuits with non-symmetric gates is at least as hard as the graph-isomorphism problem. As such, constructing an argument analogous to [1], as well as establishing the numerous other crucial results whose proofs rely on the polynomial-time decidability of various circuit properties, would require the development of a polynomial-time algorithm for graph-isomorphism.

In order to proceed we explicitly restrict our attention to *transparent* circuits. We will define this term below, but before we do we need to define a notion of ‘structural similarity’ between gates that we call *syntactic-equivalence*.

► **Definition 5.** Let $C := \langle G, \Omega, \Sigma, \Lambda, L \rangle$ be a (\mathbb{B}, ρ) -circuit of order n . We recursively define the equivalence relation *syntactic-equivalence*, which we denote using the symbol ‘ \equiv ’, on G as follows. If g and h are both input gates or both output gates then $g \equiv h$ if, and only if, $g = h$. Suppose g and h are both non-output internal gates and we have defined the

syntactic-equivalence relation for all gates of depth less than the depth of either g or h . Then $g \equiv h$ if, and only if, $\Sigma(g) = \Sigma(h)$ and $L(g)/\equiv$ and $L(h)/\equiv$ are isomorphic (as labelled structures).

The intuition is that two gates are syntactically-equivalent if the circuits underneath these two gates are structurally equivalent. The important point is that if two gates are mapped to one another by an automorphism that extends the trivial permutation, then these gates are syntactically-equivalent. In fact, we prove a slightly stronger result.

► **Lemma 6.** *Let C be a circuit of order n and $\sigma \in \mathbf{Sym}_n$. Let $\pi, \pi' \in \mathbf{Aut}(C)$ be automorphisms extending σ . For every gate g in C we have $\pi(g) \equiv \pi'(g)$.*

In this way syntactic-equivalence constrains the automorphism group. We use syntactic-equivalence to establish sufficient conditions for a circuit to have unique extensions and, moreover, for various circuit-properties that reference automorphism to be polynomial-time decidable. With these two ideas in mind we define the following classes of circuits.

► **Definition 7.** Let C be a circuit and g be an internal gate in C . We say g has *injective labels* if $L(g)$ is an injection. We say g has *unique labels* if g has injective labels and no two gates in $W(g, \cdot)$ are syntactically-equivalent. We say C has *injective labels* (resp. *unique labels*) if every gate in C has injective labels (resp. unique labels). We say C is *transparent* if every non-symmetric gate in C has unique labels.

We can translate transparent circuits into circuits with unique labels in polynomial-time. We prove this by first showing that syntactic-equivalence can be computed for transparent circuits in polynomial-time. This follows from a straightforward inductive on depth, starting from the input gates and noting that the syntactic-equivalence classes of the next layer can be computed so long as you can solve the isomorphism problem for the gates in this next layer. This is easy to do for symmetric gates, as we can check set-equivalence easily, and in the case the gate is non-symmetric then this gate has unique labels, and so there is at most one candidate isomorphism, and it is easy to check if a given function is an isomorphism.

► **Lemma 8.** *There is an algorithm that takes as input a transparent circuit C and outputs the syntactic-equivalence relation on the gates of C . The algorithm runs in time polynomial in the size of C .*

The translation from transparent circuits to circuits with unique labels is defined as follows. We define a circuit by collapsing the gates of the input circuit into its syntactic-equivalence classes, i.e. taking a quotient of the circuit by syntactic-equivalence. The resultant circuit almost has unique labels, but for the fact that certain gates computing symmetric functions might not have injective labels. For each offending gate g and each $h \in W(\cdot, g)$ that has t wires to g we add in a sequence of $t - 1$ single-input AND-gates and replace $t - 1$ wires from h to g with wires from each of these AND-gates to g . This construction gives the following result.

► **Lemma 9.** *There is an algorithm that takes as input a (\mathbb{B}, ρ) -transparent circuit C and outputs a $(\mathbb{B} \cup \mathbb{B}_{std}, \rho)$ -circuit C' such that C and C' compute the same function, C' has unique labels, and if C is symmetric then C' is symmetric. Moreover, this algorithm runs in time polynomial in the size of the input circuit.*

We have that transparent circuits can be transformed into circuits with unique labels. We should like to show that circuits with unique labels are analogous to rigid circuits in that

(i) circuits with unique labels have unique extensions and (ii) we can compute the action of an automorphism on a circuit with unique labels in polynomial-time.

Let C be a circuit of order n with unique labels of order n and let $\sigma \in \mathbf{Sym}_n$. We can define $\pi \in \mathbf{Aut}(C)$ as follows. If g is an output or input gate then the image of g is entirely determined by σ . Suppose g is an internal gate, and suppose we have constructed π for all gates h of depth greater than g . We start from the input gates and inductively construct a gate g' that, from Lemma 6, must be syntactically-equivalent to the image of g under π . We notice that, since C has unique-labels, there is at most one child of $\pi(h)$ syntactically-equivalent to g' . We can compute which child using Lemma 8, and we assign $\pi(g)$ to be this child. The above construction can be implemented as a polynomial-time algorithm, with the additional requirement that we halt and output that no automorphism exists if at any stage the construction fails. It is also important to note that at each point in this inductive definition there is always a unique extension of the automorphism to the next layer of gates. We thus have the following two results.

► **Lemma 10.** *If C is a circuit with unique labels then C has unique extensions.*

► **Lemma 11.** *There is an algorithm that takes as input a (\mathbb{B}, ρ) -circuit C of order n with unique labels and $\sigma \in \mathbf{Sym}_n$ and outputs for each gate g the image of g under the action of the unique automorphism extending σ (if it exists). This algorithm runs in time polynomial in the combined size of the input circuit and the encoding of the permutation.*

It remains to use our framework to define a class of circuits with gates that can compute rank. Let $a, b, r, p \in \mathbb{N}$, with p prime. Let $\mathbf{RANK}_p^r[a, b] : \{0, 1\}^{[a] \times [b]} \rightarrow \{0, 1\}$ be the (isomorphism-invariant) structured function with universe $[a] \uplus [b]$, and such that $\mathbf{RANK}_p^r[a, b](M) = 1$ if, and only if, the matrix $M \in \{0, 1\}^{[a] \times [b]}$ has rank at least r over \mathbb{F}_p when the entries of M are interpreted as elements of \mathbb{F}_p . Let $\mathbf{RANK} = \{\mathbf{RANK}_p^r[a, b] : a, b, r, p \in \mathbb{N}, p \text{ prime}\}$ and let the *rank basis* be $\mathbb{B}_{\mathbf{rk}} := \mathbb{B}_{\mathbf{maj}} \cup \mathbf{RANK}$. We call a circuit defined over the rank basis a *rank-circuit*.

We are now ready to state the main theorem of this paper.

► **Theorem 12 (Main Theorem).** *A graph property is decidable by a P-uniform family of transparent symmetric rank-circuits if, and only if, it is definable by an FPR sentence.*

4 Symmetry and Supports

In this section we introduce the definition of a support and supporting partition from [1] and extend the results about supports to our framework.

► **Definition 13.** Let $G \leq \mathbf{Sym}_n$ and let $S \subseteq [n]$. Then S is a *support* for G if $\mathbf{Stab}_n(S) \leq G$.

An important generalisation of the notion of a support is a *supporting partition*.

► **Definition 14.** Let $G \leq \mathbf{Sym}_n$ and \mathcal{P} be a partition of $[n]$. Then \mathcal{P} is a *supporting partition* for G if $\mathbf{Stab}_n(\mathcal{P}) \leq G$.

Let \mathcal{P} and \mathcal{P}' be supporting partitions for G . We say that \mathcal{P}' is as *coarse* as \mathcal{P} , denoted by $\mathcal{P}' \preceq \mathcal{P}$, if every part in \mathcal{P} is contained in a part in \mathcal{P}' . Every group $G \leq \mathbf{Sym}_n$ has a unique coarsest supporting partition [1]. We call this partition the *canonical supporting partition*, and denote it by $\mathbf{SP}(G)$.

It is easy to show that if \mathcal{P} is a supporting partition for $G \leq \mathbf{Sym}_n$ and P is the largest part of \mathcal{P} then $[n] \setminus P$ is a support for G . We say that G has *small support* if there exists

$P \in \mathbf{SP}(G)$ such that $|P| > \frac{n}{2}$, and if G has small support then we call $\text{sp}(G) := [n] \setminus P$ the *canonical support* for G .

We apply the language of supports to circuits. Let C be a symmetric circuit of order n with unique extensions and let g be a gate in C . There is a group action of \mathbf{Sym}_n on the gates of C , given by the isomorphism from \mathbf{Sym}_n to $\mathbf{Aut}(C)$. We say that a partition \mathcal{P} of $[n]$ (resp. a set $S \subseteq [n]$) is a supporting partition (resp. support) for g if \mathcal{P} is a supporting partition for $\mathbf{Stab}(g)$ (resp. S is a support for $\mathbf{Stab}(g)$). We abuse notation and write $\mathbf{SP}(g)$ and $\text{sp}(g)$ for the canonical supporting partition and canonical support for g . Let $\|\mathbf{SP}(g)\|$ denote the smallest value of $|[n] \setminus P|$ for $P \in \mathbf{SP}(g)$. Let $\mathbf{SP}(C)$ denote the largest value of $\|\mathbf{SP}(g)\|$ for g a gate in C . We now state the support theorem and then discuss its proof.

► **Theorem 15.** *For any ϵ and n such that $\frac{2}{3} \leq \epsilon \leq 1$ and $n \geq \frac{128}{\epsilon^2}$, if C is a symmetric circuit of order n with unique labels and $s := \max_{g \in C} |\mathbf{Orb}(g)| \leq 2^{n^{1-\epsilon}}$, then $\mathbf{SP}(C) \leq \frac{33 \log s}{\epsilon \log n}$.*

The proof follows a strategy broadly similar to the one used in [1], and makes use of two lemmas from there. The first lemma gives us that if the index of a group $G \leq \mathbf{Sym}_n$ is small then $\mathbf{SP}(G)$ either has very few or very many parts. The second lemma gives us that for $G \leq \mathbf{Sym}_n$, if $\mathbf{SP}(G)$ has very few parts then it must have a single very large part (and hence a small canonical support). These two results allow us to conclude that every gate g in C has a small canonical support if it has a canonical supporting partition with very few parts. We then prove by structural induction that the canonical supporting partition of every gate has few parts. To be precise, we show that if g is the topologically first gate in the circuit with a canonical supporting partition with too many parts then $|\mathbf{Orb}(g)| > 2^{n^{1-\epsilon}}$, i.e. the orbit is larger than the given bound.

We do this by establishing the existence of a large set H of permutations that each take g to a different gate. To construct H we define a set of triples of the form (σ, h, h') where $\sigma \in \mathbf{Sym}_n$ and $h, h' \in H_g$. Each of these triples is useful in a sense that it guarantees that σ moves g . Moreover, the triples are pairwise independent which means that we can compose them in arbitrary combinations to generate new permutations moving g , while guaranteeing that each such combination gives us a different element in the orbit of g . We have the following as an immediate consequence of the support theorem.

► **Lemma 16.** *Let $\mathcal{C} := (C_n)_{n \in \mathbb{N}}$ be a polynomial-size family of symmetric circuits with unique labels. There exists $k \in \mathbb{N}$ such that $\mathbf{SP}(C_n) \leq k$ for all $n \in \mathbb{N}$.*

Supports of Indexes

In our analysis we not only need to consider supports for gates but also for elements of the universe of a gate. Let C be a circuit with unique extensions and g be a gate in C . We define an action of $\mathbf{Stab}(g)$ on $\text{unv}(g)$ such that $\sigma \cdot a := (L(g)^{-1} \sigma L(g)(\vec{a}_R))(\vec{a}_R^{-1}(a))$, for $\sigma \in \mathbf{Stab}(g)$ and $a \in \text{unv}(g)$, and where $\vec{a}_R \in \text{ind}(g)$ contains the element a .

Since we have a group action of $\mathbf{Stab}(g)$ on $\text{unv}(g)$, but not \mathbf{Sym}_n on $\text{unv}(g)$, we must speak of the support of $a \in \text{unv}(g)$ relative to $\mathbf{Stab}(g)$. In fact, we are often interested in the action of the subgroup $\mathbf{Stab}(\text{sp}(g))$. We let $\mathbf{Stab}_{\text{sp}(g)}(a)$ and $\mathbf{Orb}_{\text{sp}(g)}(a)$ denote the orbit and stabiliser of a under the action of $\mathbf{Stab}(\text{sp}(g))$. We let $\text{sp}_{\text{sp}(g)}(a)$ and $\mathbf{SP}_{\text{sp}(g)}(a)$ denote the canonical support and canonical supporting partition of $\mathbf{Stab}_{\text{sp}(g)}(a)$. In all cases when the choice of g is obvious from context we omit the subscript. The following lemma is a direct consequence of the support theorem and extends the support theorem to the elements of the universe of a gate.

► **Lemma 17.** *Let $(C_n)_{n \in \mathbb{N}}$ be a polynomial-size family of symmetric circuits with unique labels. There exists $n_0, k \in \mathbb{N}$ such that for all $n > n_0$, g a gate C_n and $a \in \text{unv}(g)$ we have that (i) $\text{Stab}_{\text{sp}(g)}(a)$ and $\text{Stab}_n(g)$ have small support, (ii) if $h \in H_g$ and a appears in $L(g)^{-1}(h)$ then $\text{sp}_{\text{sp}(g)}(a) \subseteq \text{sp}(g) \cup \text{sp}(h)$, and (iii) $|\text{sp}(g)| \leq k$ and $|\text{sp}_{\text{sp}(g)}(a)| \leq 2k$.*

5 The Translation from Formulas into Circuits

The standard translation from formulas to families of symmetric circuits does not result in a family of *transparent* circuits. We must thus define a novel translation. We do this in two parts. We first define a translation from P-uniform families of bounded-width FO+rk-formulas to equivalent P-uniform families of transparent symmetric rank-circuits. We then define a translation from formulas of FPR to P-uniform families of bounded-width FO+rk-formulas. The first of these translations is given by the following lemma.

► **Lemma 18.** *There is a function that takes as input an FO+rk-formula $\theta(\vec{x})$ and $n \in \mathbb{N}$ and outputs a transparent symmetric rank-circuit C of order n defined over the same vocabulary as $\theta(\vec{x})$ and that computes the query defined by $\theta(\vec{x})$ for structures of size n . Moreover, this function is computable and there is a polynomial p such that for an input $(\theta(\vec{x}), n)$ the algorithm computing this function terminates in at most $p(n^{\text{width}(\theta)} |\text{cl}(\theta)|)$ many steps.*

Proof Sketch. The proof follows easily once one understands why the usual translation does not produce a transparent circuit. Consider the following example. Suppose $\psi(\vec{y})$ is a subformula of $\theta(\vec{x})$ of the form $\text{rk}_p^r \vec{w} \vec{z} . \phi$ and suppose that $\phi := \phi'(w_1) \wedge \phi'(w_2)$. In this case the syntactic structure of ϕ is fixed by any permutation of the variables that fixes $\{w_1, w_2\}$ setwise. The usual translation to circuits would preserve symmetries of this form, resulting in many of the input gates of the rank gate being syntactically-equivalent.

In order to address this we first preprocess the formula $\theta(\vec{x})$, defining a new formula $\lambda(\vec{x})$ that decides the same query but is not invariant (in the sense alluded to above) under permutations of the variables. We define $\lambda(\vec{x})$ as follows. Let R be a relation symbol in the vocabulary of $\theta(\vec{x})$ (if the vocabulary is empty the translation is trivial). For a variable y let $\text{NO-OP}(y) := (R(y, y) \vee (\neg R(y, y)))$. For a sequence of variables $\vec{y} = (y_1, \dots, y_m)$ let $\text{TAG}(\vec{y}) := (\text{NO-OP}(y_1) \wedge (\text{NO-OP}(y_2) \wedge (\text{NO-OP}(y_2) \wedge (\dots \wedge (\text{NO-OP}(y_m)) \dots))))$. Let $\lambda(\vec{x})$ be the formula constructed from $\theta(\vec{x})$ by replacing each sub-formula $\psi(\vec{y})$ of the form $\text{rk}_p^r \vec{w} \vec{z} . \phi$ with the formula $\text{rk}_p^r \vec{w} \vec{z} . ((\forall u. u = u) \wedge \phi) \wedge \text{TAG}(\vec{w} \cup \vec{z})$. Since we always replace a subformula ϕ with a logically equivalent formula, it follows that $\lambda(\vec{x})$ and $\theta(\vec{x})$ are equivalent. The intuition is that $\text{TAG}(\vec{w} \cup \vec{z})$ appends a tower of conjunctions of tautologies, with each tautology referencing a unique variable from $\vec{w} \cup \vec{z}$. When we construct the circuit, this tower of tautologies acts to ‘tag’ each input to the rank gate with a unique gadget.

We now construct C using the usual approach. For each subformula $\psi(\vec{y})$ of $\lambda(\vec{x})$ and assignment $\vec{a} \in [n]^{|\vec{y}|}$ to \vec{y} we include a gate $g_{\psi, \vec{a}}$ in C . We wire the circuit such that $g_{\phi, \vec{a}}$ is an input gate to $g_{\psi, \vec{b}}$ iff ϕ is an immediate subformula of ψ and the two assignments never assign the same variable to two different values. For a complete proof see [7]. ◀

The translation from FPR to P-uniform families of bounded-width FO+rk-formulas is a concatenation of the following two translations. First, from [5], we can translate $\theta(\vec{x}) \in \text{FPR}[\tau]$ into an equivalent P-uniform family of FOR[τ]-formulas. Second, from [14], we can translate FOR[τ]-formulas into equivalent P-uniform families of FO+rk[τ]-formulas. Both of these translations increase the width by a constant factor, and so we may apply Lemma 18 to prove the following.

► **Theorem 19.** *For each FPR-formula $\theta(\vec{x})$ there exists a P-uniform family of transparent symmetric rank-circuits $(C_n)_{n \in \mathbb{N}}$ that defines the same query as $\theta(\vec{x})$.*

6 The Translation from Circuits into Formulas

We leverage the support theorem and the various polynomial-time algorithms defined for transparent circuits and circuits with unique labels in order to define a translation from P-uniform families of symmetric rank-circuits to formulas of FPR. Let $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ denote a P-uniform family of transparent symmetric $(\mathbb{B}_{\text{rk}}, \rho)$ -circuits computing a q -ary query Q .

From the Immerman-Vardi theorem [12, 16] and Lemma 9, there is a t -width interpretation Φ such that for each ρ -structure \mathcal{A} of size n the interpretation of Φ in \mathcal{A} defines a symmetric rank-circuit with unique labels (in the number universe) equivalent to C_n . We aim to show that there exists $\theta_Q \in \text{FPR}[\rho]$ that defines Q , i.e. such that $\mathcal{A} \models \theta_Q[\vec{a}]$ if, and only if, $C_n[\gamma\mathcal{A}](\Omega(\gamma\vec{a})) = 1$ for any bijection $\gamma \in [n]^U$.

Let n_0 and k be the constants in the statement of Lemma 17. Notice that for each $n \leq n_0$ there are only constantly many bijections from the universe of a structure to $[n]$, and so we can explicitly quantify over these constantly many bijections and evaluate the circuit. We thus fix $n > n_0$ and a ρ -structure \mathcal{A} with universe U of size n and show how to evaluate C_n .

It follows from Lemma 17 that each gate g has a support of size at most k and each $a \in \text{unv}(g)$ has a support of size at most $2k$. We say that two injections f and g are *compatible* if there is an injection on the union of their domains that agrees with both functions. If there is such a function we denote it by $(f|g)$. We use \sim to denote compatibility. The following result gives us that the evaluation of a gate g for a bijection $\gamma \in [n]^U$ depends only on those elements γ maps to $\text{sp}(g)$.

► **Lemma 20.** *Let g be a gate in C_n . Let $\eta \in U^{\text{sp}(g)}$ and $\gamma_1, \gamma_2 \in [n]^U$ such that $\gamma_1^{-1} \sim \eta$ and $\gamma_2^{-1} \sim \eta$. Then $L^{\gamma_1\mathcal{A}}(g)$ and $L^{\gamma_2\mathcal{A}}(g)$ are isomorphic.*

It follows from Lemma 20 that the evaluation of g is entirely determined by $\text{EV}_g := \{\eta \in U^{\text{sp}(g)} : \exists \gamma \in [n]^U \text{ s.t. } C_n[\gamma\mathcal{A}](g) = 1 \text{ and } \eta \sim \gamma^{-1}\}$. Here we see how the support theorem allows us to characterize the evaluation of a gate succinctly.

The query defined by C_n for \mathcal{A} is $Q = \{\vec{a} \in U^q : \exists g \in G, \eta \in \text{EV}_g \text{ s.t. } \Omega(\eta^{-1} \circ \vec{a}) = g\}$. In order to define Q it is thus sufficient to show that EV_g is FPR-definable. In particular, we show that there is an FPR-definable relation $V \subseteq [n]^t \times U^k$ such that $(g, \vec{x}) \in V$ if, and only if, the assignment that maps $\text{sp}(g)$ to the first $|\text{sp}(g)|$ elements of \vec{x} is in EV_g . We do this by first describing a procedure for recursively defining EV_g , i.e. defining EV_g given $\{\text{EV}_h : h \in H_g\}$, and then arguing that this definition can be implemented in FPR. This suffices as we may then use the fixed-point operator to complete the definition of V . The gate g is either a symmetric gate or a rank gate. If g is a symmetric gate then we have a FPC-definable recursive construction of EV_g from [1]. As such, we assume g is a rank gate.

As an aside, we note that the recursive construction of EV_g in [1] relies on the fact that if g is symmetric then it can be evaluated by counting the number of its inputs that evaluate to 1. Using this fact, along with a bijection between the orbit of a gate and the assignments to the support of that gate, the problem of evaluating g reduces to a counting problem on the assignments to the supports of the inputs to g . The results that underlie this counting argument fail for non-symmetric gates, and so we are forced to use a very different approach for rank gates.

We instead show that for each gate g and $\eta \in U^{\text{sp}(g)}$ there is an FPR-definable matrix M that has the same rank as $L^{\gamma\mathcal{A}}(g)$ for any $\gamma \in [n]^U$ such that $\gamma^{-1} \sim \eta$. We can then check if $\eta \in \text{EV}_g$ by applying the rank operator to M and testing against the threshold.

We introduce some notation. Let $A \times B := \text{ind}(g)$. For $h \in H_g$ let $\text{row}(h) := L(g)^{-1}(h)(1)$ and $\text{col}(h) := L(g)^{-1}(h)(2)$. Let $A_h := \{\vec{x} \in U^{\text{sp}(h)} : \eta \sim \vec{x}\}$ and for all $a \in \text{unv}(g)$ let $A_a = \{\vec{x} \in U^{\text{sp}(a)} : \eta \sim \vec{x}\}$.

We first define the index sets for the matrix M . Let $R^{\min} := \{\min(\mathbf{Orb}(\text{row}(h))) : h \in H_g\}$ and $C^{\min} := \{\min(\mathbf{Orb}(\text{col}(h))) : h \in H_g\}$. Let $I := \{(i, \vec{x}) : i \in R^{\min}, \vec{x} \in A_i\}$ and $J := \{(j, \vec{y}) : j \in C^{\min}, \vec{y} \in A_j\}$. We think of R^{\min} and C^{\min} as indexing the orbits of the row and column elements under the action of $\mathbf{Stab}(\text{sp}(g))$, with each orbit indexed by the minimal element in A (or B , respectively) that appears in it. We think of I and J as indexing the elements within an orbit instead by elements of A_i and A_j , implicitly using the bijection between these sets and the orbits of $\text{row}(h)$ and $\text{col}(h)$.

We associate with each index $((i, \vec{x}), (j, \vec{y})) \in I \times J$ a gate h and an assignment \vec{w} to the support of h as follows. It can be shown there is a function that maps a given index to a permutation $\sigma \in \mathbf{Stab}(g)$ such that $\vec{y}\sigma$ is compatible with both η and \vec{x} (see [7] for details). Let $h = L(g)(i, \sigma j)$ and let $\vec{w} = (\vec{x}|\vec{y}\sigma)$. We define the matrix $M : I \times J \rightarrow \{0, 1\}$ by $M((i, \vec{x}), (j, \vec{y})) := \vec{w} \in \text{EV}_h$.

Let x be a gate in H_g or an element of the universe of g . Let $f \in U^{\text{sp}(x)}$ and $\gamma \in [n]^U$ such that $\gamma^{-1} \sim \eta$. Let $\Pi_f^\gamma \in \mathbf{Stab}(\text{sp}(g))$ be such that $\Pi_f^\gamma(a) = \gamma(f(a))$ for all $a \in \text{sp}(x)$. It is easy to see that $\Pi_f^\gamma(x)$ is well-defined. For a fixed $h \in H_g$, the mapping $\vec{z} \mapsto \Pi_{\vec{z}}^\gamma(h)$, for $\vec{z} \in A_h$, establishes a correspondence between A_h and the orbit of h . A similar correspondence exists for a fixed $a \in \text{unv}(g)$. It follows that $\vec{z} \in \text{EV}_h$ if, and only if, $C_n[\gamma\mathcal{A}](\Pi_{\vec{z}}^\gamma(h)) = 1$. [7]

We use this correspondence to define a mapping from M to $L^{\gamma\mathcal{A}}(g)$. Let $\alpha^\gamma : I \rightarrow A$ and $\beta^\gamma : J \rightarrow B$ be defined by $\alpha^\gamma(i, \vec{x}) := \Pi_{\vec{x}}^\gamma(i)$ and $\beta^\gamma(j, \vec{y}) := \Pi_{\vec{y}}^\gamma(j)$, respectively. It is possible to show that $(\alpha^\gamma, \beta^\gamma)$ is a surjective homomorphism from M to $L^{\gamma\mathcal{A}}(g)$. It can be shown that $\alpha^\gamma(i, \vec{x}) = \alpha^\gamma(i, \vec{x}')$ if, and only if, there exists $\pi \in \mathbf{Stab}_{\text{sp}(g)}(i)$ such that $\vec{x} = \vec{x}'\pi$ – and a similar result holds for β^γ . It follows that $(\alpha^\gamma, \beta^\gamma)$ is not, in general, injective.

We resolve this problem by quotienting. Let $s \in \text{unv}(g)$ and $\vec{x}, \vec{x}' \in A_s$. We say that $\vec{x} \approx \vec{x}'$ if, and only if, there exists $\pi \in \mathbf{Stab}(s)$ such that $\vec{x} = \vec{x}'\pi$. For $(i, \vec{x}), (i', \vec{x}') \in I$ we say that $(i, \vec{x}) \approx (i', \vec{x}')$ if, and only if, $i = i'$ and $\vec{x} \approx \vec{x}'$. We similarly define \approx on J .

It is easy to see that α^γ and β^γ are constant on \approx -equivalence classes. As such, the quotient functions α^γ/\approx and β^γ/\approx are well-defined. We can also show that $M((i, \vec{x}), (j, \vec{y})) = M((i', \vec{x}'), (j', \vec{y}'))$ if $(i, \vec{x}) \approx (i', \vec{x}')$ and $(j, \vec{y}) \approx (j', \vec{y}')$. Let $M_\approx : I/\approx \times J/\approx \rightarrow \{0, 1\}$ be defined by $M_\approx((i, [\vec{x}])_\approx, (j, [\vec{y}])_\approx) := M((i, \vec{x}), (j, \vec{y}))$. It follows from the previous observation that this function is well-defined.

Since $(\alpha^\gamma, \beta^\gamma)$ is a surjective homomorphism, $(\alpha^\gamma/\approx, \beta^\gamma/\approx)$ is a surjective homomorphism from M_\approx to $L^{\gamma\mathcal{A}}(g)$. Moreover, it follows from the previous comment on the failure of injectivity that $(\alpha^\gamma/\approx, \beta^\gamma/\approx)$ is an injection. We thus have the following result.

► **Theorem 21.** *Let $\gamma \in [n]^U$ such that $\gamma^{-1} \sim \eta$. Then $L^{\gamma\mathcal{A}}(g)$ is isomorphic to M_\approx .*

It is not hard to show that the rows $M((i, \vec{x}), \cdot)$ and $M((i', \vec{x}'), \cdot)$ are equal if $(i, \vec{x}) \approx (i', \vec{x}')$, and so $\mathbf{rk}_p(M) = \mathbf{rk}_p(M_\approx)$. From this and Theorem 21 we have the following result.

► **Lemma 22.** *Let $\gamma \in U^n$ be such that $\gamma^{-1} \sim \eta$ and let $p \in \mathbb{N}$ be prime. Then $\mathbf{rk}_p(M) = \mathbf{rk}_p(M_\approx) = \mathbf{rk}_p(L^{\gamma\mathcal{A}}(g))$.*

It remains to justify our assertion that the above recursive definition of EV_g can be implemented in FPR. It is sufficient to show that there is an FPR-formula that defines M for a rank gate g and assignment $\eta \in U^{\text{sp}(g)}$. We first show that the sets $\{(g, \text{sp}(g)) : g \in G\}$, I , and J are FPR-definable. We have the following results as a consequence of Lemma 11.

► **Lemma 23.** *There is an algorithm that takes in a circuit C with unique labels and outputs if the circuit is symmetric. If it is symmetric then it outputs for each gate g and $a \in \text{unv}(g)$ the orbit $\text{Orb}(g)$ and canonical supporting partition $\text{SP}(g)$, as well as $\text{Orb}_{\text{sp}(g)}(a)$ and $\text{SP}_{\text{sp}(g)}(a)$. This algorithm runs in time polynomial in the size of the circuit.*

From Lemma 23 and the Immerman-Vardi theorem there are FPC-formulas that define the canonical support and orbit for each gate g and each $a \in \text{unv}(g)$. Moreover, it can be shown that compatibility between assignments to supports is FPR-definable. It follows that we can define A_a for each $a \in \text{unv}(g)$ and A_h for each $h \in H_g$. Combining these results we have that I and J are FPR-definable. We then define M using a relation symbol V' that denotes the value of V at a given stage in the recursive construction. This completes the FPR-definition of M and so EV_g , and hence the proof of our main result.

7 Concluding Remarks and Future Work

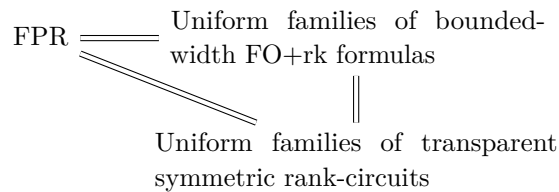
FPR is one of the most expressive logics we know that is still contained in P and understanding its expressive power is an important question. The main result of this paper establishes an equivalence between the expressive power of FPR and the computational power of uniform families of transparent symmetric rank-circuits. Not only does this establish an interesting characterization of an important logic, it also deepens our understanding of the connection between logic and circuit complexity and sheds new light on foundational aspects of the circuit model.

The circuit characterisation helps emphasise certain important aspects of the logic. Given that P-uniform families of invariant circuits (without the restriction to symmetry) express all properties on P, we can understand the inability of FPC (and, conjecturally, FPR) to express all such properties as essentially down to symmetry. As with other (machine) models of computation, the translation to circuits exposes the inherent combinatorial structure of an algorithm. In the case of logics, we find that a key property of this structure is its symmetry and the translation to circuits provides us with the tools to study it.

Still, the most significant contribution of this paper is not in the main result but in the techniques that are developed to establish it, and we highlight some of these now. The conclusion of [1] says that the support theorem is “largely agnostic to the particular [...] basis”, suggesting that it could be easily adapted to include other gates. This turns out to have been a misjudgment. Attempting to prove the support theorem for a basis that includes rank threshold gates showed us the extent to which both the proof of the theorem and, more broadly, the definitions of circuit classes, rest heavily on the assumption that all functions computed by gates are symmetric. Thus, in order to define what the “symmetry” condition might mean for circuits that include rank threshold gates, we radically generalise the circuit framework to allow for gates that take structured inputs (rather than sets of 0s and 1s) and are invariant under isomorphisms. This leads to a refined notion of circuit automorphism, which allows us to formulate a notion of symmetry and prove a version of the support theorem. Again, in that proof, substantial new methods are required.

The condition of *transparency* makes the translation of uniform circuit families into formulas of logic (which is the difficult direction of our characterisation) possible, but it complicates the other direction. Indeed, the natural translation of formulas of FPR into uniform circuit families yields circuits which are symmetric, but not transparent. This problem is addressed by introducing gadgets in the translation – which for ease of exposition, we did in formulas of FO+rk which are then translated into circuits in the natural way. Thus, the restriction to transparent circuits is sufficient to get both directions of the characterisation.

In short, we can represent the proof of our characterisation through the three equivalences in this triangle.



This highlights another interesting aspect of our result. The first translation, of FPR to uniform families of FO+rk formulas was given in [5] and used there to establish arity lower bounds. However, this was for a weaker version of the rank logic rather than the strictly more expressive one defined by Grädel and Pakusa [9]. The fact that we can complete the cycle of equivalences with the more powerful logic demonstrates that the definition of Grädel and Pakusa is the “right” formulation of FPR.

Future Work

There are many directions of work suggested by the methods and results developed in this paper. First of all, there is the question of transparency. We introduce it as a technical device that enables our characterisation to go through. Could it be dispensed with? Or are P-uniform families of transparent symmetric rank-circuits strictly weaker than families without the restriction of transparency?

The framework we have developed for working with circuits with structured inputs is very general and not specific to rank gates. It would be interesting to apply this framework to other logics. It appears to be as general a way of extending the power of circuits as Lindström quantifiers are in the context of logic. We would like to develop this link further, perhaps for specific quantifiers such as FP extended by an operator that expresses the solubility of systems of equations over rings as in [4]

At the moment, we have little by way of methods for proving inexpressibility results for FPR, whether we look at it as a logic or in the circuit model. The logical formulation lays emphasis on some parameters (the number of variables, the arity of the operators, etc.) which we can treat as resources against which to prove lower bounds. On the other hand, the circuit model brings to the fore other, more combinatorial, parameters. One such is the fan-in of gates and a promising and novel approach is to try and prove lower bounds for symmetric circuits with gates with bounded fan-in. We might ask if it is possible to compute AND[3] using a symmetric circuit with gates that have fan-in two. Perhaps we could also combine the circuit view with lower-bound methods from logic, such as pebble games. Dawar [3] has shown how the bijection games of Hella [11] can be used directly to prove lower bounds for symmetric circuits without reference to the logic. We also have pebble games for FPR [6], and it would be interesting to know if we can use these on circuits and how the combinatorial parameters of the circuit interact with the game.

Finally, we note that some of the interesting directions on the interplay between logic and symmetric circuits raised in [1] remain relevant. Can we relax the symmetry condition to something in between requiring invariance of the circuit under the full symmetric group (the case of symmetric circuits) and requiring no invariance condition at all? Can such restricted symmetries give rise to interesting logics in between FPR and P? It also remains a challenge to find a circuit characterisation of CPTC. Could the general framework for non-symmetric gates we have developed here help in this respect?

References

- 1 M. Anderson and A. Dawar. On symmetric circuits and fixed-point logics. *Theory of Computing Systems*, 60(3):521–551, 2017.
- 2 A. Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1):8–21, 2015.
- 3 A. Dawar. On symmetric and choiceless computation. In Mohammad Taghi Hajiaghayi and Mohammad Reza Mousavi, editors, *Topics in Theoretical Computer Science*, pages 23–29, Cham, 2016. Springer International Publishing.
- 4 A. Dawar, E. Grädel, B. Holm, E. Kopczynski, and W. Pakusa. Definability of linear equation systems over groups and rings. *Logical Methods in Computer Science*, 9(4), 2013.
- 5 A. Dawar, M. Grohe, B. Holm, and B. Laubner. Logics with rank operators. In *2009 24th Annual IEEE Symposium on Logic In Computer Science (LICS)*, pages 113–122, 2009.
- 6 A. Dawar and B. Holm. Pebble games with algebraic rules. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, pages 251–262, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- 7 A. Dawar and G. Wilsenach. Symmetric circuits for rank logic. *arXiv*, 2018. [arXiv:1804.02939](https://arxiv.org/abs/1804.02939).
- 8 L. Denenberg, Y. Gurevich, and S. Shelah. Definability by constant-depth polynomial-size circuits. *Information and Control*, 70(2):216–240, 1986.
- 9 E. Grädel and W. Pakusa. Rank logic is dead, long live rank logic! In *2015 24th Annual Conference on Computer Science Logic, (CSL)*, pages 390–404, 2015.
- 10 M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017. URL: <https://books.google.co.uk/books?id=RLYrDwAAQBAJ>.
- 11 L. Hella. Logical hierarchies in ptime. *Information and Computation*, 129(1):1–19, 1996.
- 12 N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1-3):86–104, 1986.
- 13 N. Immerman. *Descriptive Complexity*. Graduate texts in computer science. Springer New York, 1999.
- 14 L. Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2004.
- 15 M. Otto. The logic of explicitly presentation-invariant circuits. In *1996 10th International Workshop, Annual Conference on Computer Science Logic (CSL)*, pages 369–384. Springer, Berlin, Heidelberg, 1997.
- 16 M. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 137–146, New York, NY, USA, 1982. ACM.

Beyond Polarity: Towards a Multi-Discipline Intermediate Language with Sharing

Paul Downen

University of Oregon, Eugene, OR, USA
pdownen@cs.uoregon.edu

Zena M. Ariola

University of Oregon, Eugene, OR, USA
ariola@cs.uoregon.edu

Abstract

The study of *polarity* in computation has revealed that an “ideal” programming language combines both call-by-value and call-by-name evaluation; the two calling conventions are each ideal for half the types in a programming language. But this binary choice leaves out call-by-need which is used in practice to implement lazy-by-default languages like Haskell. We show how the notion of polarity can be extended beyond the value/name dichotomy to include call-by-need by only adding a mechanism for sharing and the extra *polarity shifts* to connect them, which is enough to compile a Haskell-like functional language with user-defined types.

2012 ACM Subject Classification Theory of computation → Type structures

Keywords and phrases call-by-need, polarity, call-by-push-value, control

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.21

Funding This work is supported by the National Science Foundation under grants CCF-1719158 and CCF-1423617.

1 Introduction

Finding a universal intermediate language suitable for compiling and optimizing both strict and lazy functional programs has been a long-sought holy grail for compiler writers. First there was continuation-passing style (CPS) [19, 2], which hard-codes the evaluation strategy into the program itself. In CPS, all the specifics of evaluation strategy can be understood just by looking at the syntax of the program. Second there were monadic languages [13, 17], that abstract away from the concrete continuation-passing into a general monadic sequencing operation. Besides moving away from continuations, making them an optional rather than mandatory part of sequencing, they make it easier to incorporate other computational effects by picking the appropriate monad for those effects. Third there were adjunctive languages [10, 23, 14], as seen in polarized logic and call-by-push-value λ -calculus, that mix both call-by-name and -value evaluation inside a single program. Like the monadic approach, adjunctive languages make evaluation order explicit within the terms and types of a program, and can easily accommodate effects. However, adjunctive languages also enable more reasoning principles, by keeping the advantages of inductive call-by-value data types, as seen in their denotational semantics. For example, the denotation of a list is just a list of values, not a list of values interspersed with computations that might diverge or cause side effects.

Each of these developments have focused only on call-by-value and -name evaluation, but there are other evaluation strategies out there. For example, to efficiently implement laziness, the Glasgow Haskell Compiler (GHC) uses a core intermediate language which is



© Paul Downen and Zena M. Ariola;
licensed under Creative Commons License CC-BY
27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 21; pp. 21:1–21:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

call-by-need [4] instead of call-by-name: the computation of named expressions is shared throughout the lifetime of their result, so that they need not be re-evaluated again. This may be seen as merely an optimization of call-by-name, but it is one that has a profound impact on the other optimizations the compiler can do. For example, full extensionality of functions (*i.e.*, the η law) does not apply in general, due to issues involving divergence and evaluation order. Furthermore, call-by-need is not just a mere optimization but a full-fledged language choice when effects are introduced [3]: call-by-need and -name are observationally different. This difference may not matter for pure functional programs, but even there, effects *become* important during compilation. For example, it is beneficial to use join points [12], which is a limited form of jump or *goto* statement, to optimize pure functional programs.

So it seems like the quest for a universal intermediate language is still ongoing. To handle all the issues involving evaluation order in modern functional compilers, the following questions, which have been unanswered so far, should also be addressed:

- (Section 3) How do you extend polarity with sharing (*i.e.*, call-by-need)? For example, how do you model the Glasgow Haskell Compiler (GHC) which mixes both call-by-need for ordinary Haskell programs and call-by-value for *unboxed* [18] machine primitives?
- (Section 4) What does a core language need to serve as a compile target for a general functional programming language with user-defined types? What are the *shifts* you need to convert between all three calling conventions? While encoding data types is routine, what do you need to fully encode *co-data types* [9]?
- (Section 5) How do you compile that general functional language to the core intermediate sub-language? And how do you know that it is robust when effects are added?

This paper answers each of these questions. The formal relationship between our intermediate language and both polarity and call-by-push-value (Appendix A). To test the robustness of this idea, we extend it in several directions in the appendix. We generalize to a dual sequent calculus framework that incorporates more calling conventions (specifically, the dual to call-by-need) and connectives not found in functional languages (Appendices B and C).

2 Polarity, data, and co-data

To begin, let's start with a basic language which is the λ -calculus extended with sums, as expressed by the following types and terms:

$$\begin{aligned} A, B, C &::= X \mid A \rightarrow B \mid A \oplus B \\ M, N, P &::= x \mid \lambda x.M \mid M N \mid \iota_1 M \mid \iota_2 M \mid \mathbf{case} M \mathbf{of} \{ \iota_1 x.N \mid \iota_2 y.P \} \end{aligned}$$

As usual, an abstraction $\lambda x.M$ is a term of a function type $A \rightarrow B$ and an injection $\iota_i M$ is a term of a sum type $A \oplus B$. Terms of function and sum types are used via application ($M N$) and **case** analysis, respectively. Variables x can be of any type, even an atomic type X .

To make this a programming language, we would need to explain how to run programs (say, closed terms of a sum type) to get results. But what should the calling convention be? We could choose to use call-by-value evaluation, wherein a function application $(\lambda x.M) N$ is reduced by first evaluating N and then plugging its value in for x , or call-by-name evaluation, wherein the same application is reduced by immediately substituting N for x without further evaluation. We might think that this choice just impacts efficiency, trading off the cost of evaluating an unneeded argument in call-by-value for the potential cost of re-evaluating the same argument many times in call-by-name. However, the choice of calling convention also impacts the properties of the language, and can affect our ability to reason about programs.

Functions are a co-data type [7], so the extensionality law for functions, known as η , expands function terms into trivial λ -abstractions as follows:

$$(\eta_{\rightarrow}) \quad M : A \rightarrow B = \lambda x.M \ x \quad (x \notin FV(M))$$

But once we allow for any computational effects in the language, this law only makes sense with respect to call-by-name evaluation. For example, suppose that we have a non-terminating term Ω (perhaps caused by general recursion) which never returns a value. Then the η_{\rightarrow} law stipulates that $\Omega = \lambda x.\Omega \ x$. This equality is fine – it does not change the observable behavior of any program – in call-by-name, but in call-by-value, $(\lambda z.5) \ \Omega$ loops forever and $(\lambda z.5) \ (\lambda x.\Omega \ x)$ returns 5. So the full η_{\rightarrow} breaks in call-by-value.

In contrast, sums are a data type, so one sensible extensionality law for sums, which corresponds to reasoning by induction on the possible cases of a free variable, is expressed by the following law stating that if x has type $A \oplus B$ then it does no harm to **case** on x first:

$$(\eta_{\oplus}) \quad M = \mathbf{case} \ x \ \mathbf{of} \ \{\iota_1 y.M[\iota_1 y/x] \mid \iota_2 z.M[\iota_2 z/x]\} \quad (x : A \oplus B)$$

Unfortunately, this law only makes sense with respect to call-by-value evaluation once we have effects. For example, consider the instance where M is $\iota_1 x$. In call-by-value, variables stand for *values* which are already evaluated because that is all that they might be substituted for. So in either case, when we plug in something like $\iota_i 5$ for x , we get the result $\iota_1(\iota_i 5)$ after evaluating the right-hand side. But in call-by-name, variables range over all terms which might induce arbitrary computation. If we substitute Ω for x , then the left-hand side results in $\iota_1 \Omega$ but the right-hand side forces evaluation of Ω with a **case**, and loops forever.

How can we resolve this conflict, where one language feature “wants” call-by-name evaluation and the other “wants” call-by-value? We just could pick one or the other as the default of the language, to the detriment of either functions or sums. Or instead we could integrate the two to get the best of both worlds, and *polarize* the language so that functions are evaluated according to call-by-name, and sums according to call-by-value. That way, both of them have their best properties in the same language, even when effects come into play. Since functions and sums are already distinguished by types, we can leverage the type system to make the call-by-value and -name distinction for us. That is to say, a type A might classify either a call-by-value term, denoted by A_+ , or a call-by-name term, denoted by A_- . Put it all together, we get the following polarized typing rules for our basic λ -calculus:

$$\begin{array}{c} A, B, C ::= A_+ \mid A_- \quad A_-, B_- ::= X^- \mid A_+ \rightarrow B_- \quad A_+, B_+ ::= X^+ \mid A_+ \oplus B_+ \\ \hline \Gamma, x : A \vdash x : A \quad \text{Var} \quad \frac{\Gamma, x : A_+ \vdash M : B_-}{\Gamma \vdash \lambda x.M : A_+ \rightarrow B_-} \rightarrow I \quad \frac{\Gamma \vdash M : A_+ \rightarrow B_- \quad \Gamma \vdash N : A_+}{\Gamma \vdash M \ N : B_-} \rightarrow E \\ \frac{\Gamma \vdash M : A_+}{\Gamma \vdash \iota_1 M : A_+ \oplus B_+} \oplus I_1 \quad \frac{\Gamma \vdash M : B_+}{\Gamma \vdash \iota_2 M : A_+ \oplus B_+} \oplus I_2 \\ \frac{\Gamma \vdash M : A_+ \oplus B_+ \quad \Gamma, x : A_+ \vdash N : C \quad \Gamma, y : B_+ \vdash P : C}{\Gamma \vdash \mathbf{case} \ M \ \mathbf{of} \ \{\iota_1 x.N \mid \iota_2 y.P\} : C} \oplus E \end{array}$$

Note that, with this polarization, injections are treated as call-by-value, in $\iota_i M$ the term M is evaluated before the tagged value is returned. More interestingly, the function call $M \ N$ has two parts: the argument N is evaluated before the function is called as in call-by-value, but this only happens once the result is demanded as in call-by-name.

But there’s a problem, just dividing up the language into two has severely restricted the ways we can compose types and terms. We can no longer inject a function into a sum, because a function is negative but a sum can only contain positive parts. Even more extreme,

21:4 Beyond Polarity

the identity function $\lambda x.x : A \rightarrow A$ no longer makes sense: the input must be a positive type and the output a negative type, and A cannot be both positive and negative at once. To get around this restriction, we need the ability to *shift* polarity between positive and negative. That way, we can still compose types and terms any way we want, just like before, and have the freedom of making the choice between call-by-name or -value instead of having the language impose one everywhere.

If we continue the data and co-data distinction that we had between sums and functions above, there are different ways of arranging the two shifts in the literature, depending on the viewpoint. In Levy’s call-by-push-value [10] the shift from positive to negative \uparrow (therein called F) can be interpreted as a data type, where the sequencing operation is subsumed by the usual notion of a **case** on values of that data type, and the reverse shift \downarrow (therein called U) can be interpreted as co-data type:¹

$$\begin{array}{l} A_-, B_- ::= \dots \mid \uparrow A_+ \\ A_+, B_+ ::= \dots \mid \downarrow A_- \end{array} \quad \frac{\Gamma \vdash M : A_+}{\Gamma \vdash \text{val } M : \uparrow A_+} \uparrow I \quad \frac{\Gamma \vdash M : \uparrow A_+ \quad \Gamma, x : A_+ \vdash N : C}{\Gamma \vdash \text{case } M \text{ of } \{\text{val } x.N\} : C} \uparrow E$$

$$\frac{\Gamma \vdash M : A_-}{\Gamma \vdash \lambda \text{enter}.M : \downarrow A_-} \downarrow I \quad \frac{\Gamma \vdash M : \downarrow A_-}{\Gamma \vdash M.\text{enter} : A_-} \downarrow E$$

$M.\text{enter}$ can be seen as sending the request `enter` to M , and $\lambda \text{enter}.M$ as waiting for that request. In contrast, Zeilberger’s calculus of unity [22] takes the opposite view, where the shift \uparrow from positive to negative is co-data and the opposite shift \downarrow is data:

$$\begin{array}{l} A_-, B_- ::= \dots \mid \uparrow A_+ \\ A_+, B_+ ::= \dots \mid \downarrow A_- \end{array} \quad \frac{\Gamma \vdash M : A_+}{\Gamma \vdash \lambda \text{eval}.M : \uparrow A_+} \uparrow I \quad \frac{\Gamma \vdash M : \uparrow A_+}{\Gamma \vdash M.\text{eval} : A_+} \uparrow E$$

$$\frac{\Gamma \vdash M : A_-}{\Gamma \vdash \text{box } M : \downarrow A_-} \downarrow I \quad \frac{\Gamma \vdash M : \downarrow A_- \quad \Gamma, x : A_- \vdash N : C}{\Gamma \vdash \text{case } M \text{ of } \{\text{box } x.N\} : C} \downarrow E$$

Here, we do not favor one form over the other and allow both forms to coexist. It turns out that with only call-by-value and -name evaluation, the two pairs of shifts amount to the same thing (more formally, we will see in Section 5 that they are *isomorphic*). But we will see next in Section 3 how extending this basic language calls both styles of shifts into play.

With the polarity shifts between positive and negative types, we can express every program that we could have in the original unpolarized language. The difference is that now since *both* call-by-value and -name evaluation is denoted by different types, the types themselves signify the calling convention. For call-by-name, this encoding is:

$$\begin{aligned} \llbracket X \rrbracket^- &= X^- & \llbracket A \rightarrow B \rrbracket^- &= (\downarrow \llbracket A \rrbracket^-) \rightarrow \llbracket B \rrbracket^- & \llbracket A \oplus B \rrbracket^- &= \uparrow((\downarrow \llbracket A \rrbracket^-) \oplus (\downarrow \llbracket B \rrbracket^-)) \\ \llbracket x \rrbracket^- &= x \\ \llbracket M N \rrbracket^- &= \llbracket M \rrbracket^- (\text{box } \llbracket N \rrbracket^-) & \llbracket \lambda x.M \rrbracket^- &= \lambda y. \text{case } y \text{ of } \{\text{box } x. \llbracket M \rrbracket^-\} \\ \llbracket \iota_i M \rrbracket^- &= \text{val}(\iota_i(\text{box } \llbracket M \rrbracket^-)) & \llbracket \text{case } M \text{ of } \{\iota_i x_i. N_i\} \rrbracket^- &= \text{case } \llbracket M \rrbracket^- \text{ of } \{\text{val}(\iota_i(\text{box } x_i)). \llbracket N_i \rrbracket^-\} \end{aligned}$$

¹ Note that this $\uparrow E$ rule is an *extension* of the elimination rule for F in call-by-push-value [10], which restricts C to be only a negative type. The impact is that, unlike call-by-push-value, this language allows for *non-value terms* of positive types, similar to SML. The extension is *conservative*, because the interpretation of A_+ values is identical to call-by-push-value, whereas the interpretation of a non-value term of type A_+ would be shifted in call-by-push-value as the computation type $\uparrow A_+$. This interpretation also illustrates how to compile the extended calculus to the lower-level call-by-push-value by \uparrow -shifting following the standard encoding of call-by-value, where positive non-value terms have an explicit `val` wherever they may return a value. More details can be found in Appendix A.

where the nested pattern $\text{val}(\iota_i(\text{box } x_i))$ is expanded in the obvious way. It converts every type into a negative one, and amounts to **boxing up** the arguments of injections and function calls. The call-by-value encoding is:

$$\begin{aligned} \llbracket X \rrbracket^+ &= X^+ & \llbracket A \rightarrow B \rrbracket^+ &= \Downarrow(\llbracket A \rrbracket^+ \rightarrow (\uparrow\llbracket B \rrbracket^+)) & \llbracket A \oplus B \rrbracket^+ &= \llbracket A \rrbracket^+ \oplus \llbracket B \rrbracket^+ \\ \llbracket x \rrbracket^+ &= x \\ \llbracket M N \rrbracket^+ &= ((\llbracket M \rrbracket^+.\text{enter}) \llbracket N \rrbracket^+).\text{eval} & \llbracket \lambda x.M \rrbracket^+ &= \lambda \text{enter}.\lambda x.\lambda \text{eval}.\llbracket M \rrbracket^+ \\ \llbracket \iota_i M \rrbracket^+ &= \iota_i \llbracket M \rrbracket^+ & \llbracket \text{case } M \text{ of } \{\iota_i x_i.N_i\} \rrbracket^+ &= \text{case } \llbracket M \rrbracket^+ \text{ of } \{\iota_i x_i.\llbracket N_i \rrbracket^+\} \end{aligned}$$

It converts every type into a positive one. As such, sum types do not have to change (because, like SML, we have not restricted positive types to only classifying values as in [14]). Instead, the shifts appear in function types: to call a function, we must first enter the abstraction, perform the call, then evaluate the result.

At a basic level, these two encodings make sense from the perspective of typability (corresponding to provability in logic) – by inspection, all of the types line up with their newly-assigned polarities. But programs are meant to be run, so we care about more than just typability. At a deeper level, the encodings are *sound* with respect to equality of terms: if two terms are equal, then their encodings are also equal. We have not yet formally defined equality, so we will return to this question later in Section 5.1.

3 Polarity and sharing

So far we have considered only call-by-value and -name calculi. What about call-by-need, which models sharing and memoization for lazy computation; what would it take to add that, too? The shifts we have are no longer enough: to complete the picture we also require shifts between call-by-need and the other polarities. We need to be able to shift into and out of the positive polarity in order for call-by-need to access data like the sum type. And we also need to be able to shift into and out of the negative polarity for call-by-need to be able to access co-data like the function type. That is a total of four more shifts to connect the ordinary polarized language to the call-by-need world. The question is, how do we align the four different shifts that we saw previously? Since call-by-need only needs access to the positive world for representing data types, we use the data forms of shifts between those two. Dually, since call-by-need only needs access to the negative world for representing co-data types, we use the co-data forms of shifts between those two. We will also need a mechanism for representing sharing. The traditional representation [4] is with **let**-bindings, and so we will do the same. In all, we have:

$$\begin{array}{c}
A, B, C ::= A_+ \mid A_- \mid A_\star \qquad A_-, B_- ::= X^- \mid A_+ \rightarrow B_- \mid \uparrow A_+ \mid \uparrow A_+ \mid \uparrow_\star A_\star \\
A_\star, B_\star ::= X^\star \mid \star\uparrow A_+ \mid \star\downarrow A_- \qquad A_+, B_+ ::= X^+ \mid A_+ \oplus B_+ \mid \downarrow A_- \mid \downarrow A_- \mid \downarrow_\star A_\star \\
\frac{\Gamma \vdash M : A_\star}{\Gamma \vdash \lambda \text{eval}_\star.M : \uparrow_\star A_\star} \uparrow I \qquad \frac{\Gamma \vdash M : \uparrow_\star A_\star}{\Gamma \vdash M.\text{eval}_\star : A_\star} \uparrow E \\
\frac{\Gamma \vdash M : A_\star}{\Gamma \vdash \text{box}_\star M : \downarrow_\star A_\star} \downarrow I \qquad \frac{\Gamma \vdash M : \downarrow_\star A_\star \quad \Gamma, x : A_\star \vdash N : C}{\Gamma \vdash \text{case } M \text{ of } \{\text{box}_\star x.N\} : C} \downarrow E \\
\frac{\Gamma \vdash M : A_+}{\Gamma \vdash \text{val}_\star M : \star\uparrow A_+} \uparrow I \qquad \frac{\Gamma \vdash M : \star\uparrow A_+ \quad \Gamma, x : A_+ \vdash N : C}{\Gamma \vdash \text{case } M \text{ of } \{\text{val}_\star x.N\} : C} \uparrow E \\
\frac{\Gamma \vdash M : A_-}{\Gamma \vdash \lambda \text{enter}_\star.M : \star\downarrow A_-} \downarrow I \qquad \frac{\Gamma \vdash M : \star\downarrow A_-}{\Gamma \vdash M.\text{enter}_\star : A_-} \downarrow E \\
\frac{\Gamma \vdash M : A \quad \Gamma, x : A \vdash N : C}{\Gamma \vdash \text{let } x = M \text{ in } N : C} \text{Let}
\end{array}$$

Now, how can a call-by-need λ -calculus with functions and sums be encoded into this polarized setting? We effectively combine both the call-by-name and -value encodings, where a shift is used for call-by-need whenever one is used for either of the other two.

$$\begin{aligned}
\llbracket X \rrbracket^\star &= X^\star & \llbracket A \rightarrow B \rrbracket^\star &= \star\downarrow((\downarrow_\star \llbracket A \rrbracket^\star) \rightarrow (\uparrow_\star \llbracket B \rrbracket^\star)) & \llbracket A \oplus B \rrbracket^\star &= \star\uparrow((\downarrow_\star \llbracket A \rrbracket^\star) \oplus (\downarrow_\star \llbracket B \rrbracket^\star)) \\
\llbracket x \rrbracket^\star &= x \\
\llbracket M N \rrbracket^\star &= ((\llbracket M \rrbracket^\star).\text{enter}_\star) (\text{box}_\star \llbracket N \rrbracket^\star).\text{eval}_\star \\
\llbracket \lambda x.M \rrbracket^\star &= \lambda \text{enter}_\star.\lambda y.\text{case } y \text{ of } \{\text{box}_\star x.\lambda \text{eval}_\star.\llbracket M \rrbracket^\star\} \\
\llbracket \iota_i M \rrbracket^\star &= \text{val}_\star(\iota_i(\text{box}_\star \llbracket M \rrbracket^\star)) \\
\llbracket \text{case } M \text{ of } \{\iota_i x_i.N_i\} \rrbracket^\star &= \text{case } \llbracket M \rrbracket^\star \text{ of } \{\text{val}_\star(\iota_i(\text{box}_\star x_i)).\llbracket N_i \rrbracket^\star\}
\end{aligned}$$

The key thing to notice here is what is shared and what is not, to ensure that the encoding correctly aligns with call-by-need evaluation. Both the shifts *into* \star , the data type $\star\uparrow A_+$ and co-data type $\star\downarrow A_-$, result in terms that can be shared by a **let**. But the shifts *out of* \star are different: the content M of $\text{box}_\star M : \downarrow_\star A_\star$ is still shared, like a data structure, but the content M of $\lambda \text{eval}_\star.M : \uparrow_\star A_\star$ is not, like a λ -abstraction. Therefore, the encoding of an injection $\llbracket \iota_i M \rrbracket^\star$ shares the computation of $\llbracket M \rrbracket^\star$ throughout the lifetime of the returned value, as for the argument of a function call:

$$\llbracket \text{case } \iota_i M \text{ of } \{\iota_i x_i.N_i\} \rrbracket^\star = \text{let } x_i = \llbracket M \rrbracket^\star \text{ in } \llbracket N_i \rrbracket^\star \qquad \llbracket (\lambda x.M)N \rrbracket^\star = \text{let } x = \llbracket N \rrbracket^\star \text{ in } \llbracket M \rrbracket^\star$$

Whereas, the encoding of a function $\llbracket \lambda x.M \rrbracket^\star$, being a value, re-computes $\llbracket M \rrbracket^\star$ every time the function is used, which is formalized by the equational theory in Section 4.4.

4 A multi-discipline intermediate language

So far, we have only considered how sharing interacts with polarity in a small language with functions and sums, but programming languages generally have more than just those two types. For example, both SML and Haskell have pairs so we should include those, too, but when do we have enough of a “representative” basis of types that serves as the core kernel language for the general source language? To define our core intermediate language, we will follow the standard practice (as in CPS) of first defining a more general source language, and then identifying the core sub-language that the entire source can be translated into.

The biggest issue is that faithfully encoding types of various disciplines into a core set of primitives is more subtle than it may at first seem. For example, using Haskell’s algebraic data type declaration mechanism, we can define both a binary and ternary sum:

<pre>data Either a b where Left : a → Either a b Right : b → Either a b</pre>	<pre>data Either3 a b c where Choice1 : a → Either3 a b c Choice2 : b → Either3 a b c Choice3 : c → Either3 a b c</pre>
--	---

But `Either a (Either b c)` does not faithfully represent `Either3 a b c` in Haskell, even though it does in SML. The two types are convertible:

<code>nest(Choice1 x) = Left x</code>	<code>unnest(Left x) = Choice1 x</code>
<code>nest(Choice2 y) = Right(Left y)</code>	<code>unnest(Right(Left y)) = Choice2 y</code>
<code>nest(Choice3 z) = Right(Right z)</code>	<code>unnest(Right(Right z)) = Choice3 z</code>

but they do not describe the same values. `Either a (Either b c)` types both the observably distinct terms Ω and `Right Ω` – which can be distinguished by pattern matching – but conversion to `Either3 a b c` collapses them both to Ω . This is not just an issue of needing nary tuples and sums, the same issue arises when pairs and sums are nested with each other.

To ensure that we model a general enough source language, we will consider one that is *extensible* (i.e., allows for user-defined types encompassing many types found in functional languages) and *multi-discipline* (i.e., allows for programs that mix call-by-value, -name, and -need evaluation). These two features interact with one another: user-defined types can combine parts with different calling conventions. But even though users can define many different types, there is still a fixed core set of types, \mathcal{F} , capable of representing them all. For example, an extensible and multi-discipline calculus encompasses both the source and target of the three encodings showed previously in Sections 2 and 3. We now look at the full core intermediate language \mathcal{F} , and how to translate general source programs into the core \mathcal{F} .

4.1 The functional core intermediate language: \mathcal{F}

Our language allows for user-defined data and co-data types. A data type introduces a number of constructors for building values of the type, a co-data type introduces a number of *observers* for observing or interacting with values of the type. Figure 1 presents some important examples that define a *core* set of types, \mathcal{F} . The calculus instantiated with just the \mathcal{F} types serves as our core intermediate language, as it contains all the needed functionality.

The data and codata declarations for \oplus and \rightarrow correspond to the polarized sum and function types from Section 2, with a slight change of notation: we write $X : +$ instead of X^+ . The data declaration of \oplus defines its two *constructors* ι_1 and ι_2 , and dually the co-data declaration for \rightarrow defines its one *observer* `call`. The terms of the resulting sum type are exactly as they were presented in Section 2. The function type uses a slightly more verbose notation than the λ -calculus for the sake of regularity: instead of $\lambda x.M$ we have $\lambda\{\text{call } x.M\}$ and instead of $M N$ we have $M.\text{call } N$. That is, dual to a **case** matching on the pattern of a data structure, a λ -abstraction matches on the co-pattern of a co-data observation like `call x`. Besides changing notation, the meaning is the same [7].

There are some points to notice about these two declarations. First, disciplines can be mixed within a single declaration, which is used to define the polarized \rightarrow function space that accepts a call-by-value (+) input and returns a call-by-name (–) result, but other

Simple (co-)data types		
data $(X:+) \oplus (Y:+) : +$ where $\iota_1 : (X:+ \vdash X \oplus Y)$ $\iota_2 : (Y:+ \vdash X \oplus Y)$	data $(X:+) \otimes (Y:+) : +$ where $(-, -) : (X:+, Y:+ \vdash X \otimes Y)$	data $0 : +$ where data $1 : +$ where $() : (\vdash 1)$
codata $(X:-) \& (Y:-) : -$ where $\pi_1 : (X \& Y \vdash X:-)$ $\pi_2 : (X \& Y \vdash Y:-)$	codata $\top : -$ where	codata $(X:+) \rightarrow (Y:-) : -$ where $\text{call} : (X:+ X \rightarrow Y \vdash Y:-)$
Quantifier (co-)data types		Polarity shift (co-)data types
data $\exists_k(X:k \rightarrow +) : +$ where $\text{pack} : (X Y:+ \vdash^{Y:k} \exists_k X)$	data $\downarrow_{\mathcal{S}}(X:\mathcal{S}) : +$ where $\text{box}_{\mathcal{S}} : (X:\mathcal{S} \vdash \downarrow_{\mathcal{S}} X)$	data $\mathcal{S}\uparrow(X:+) : \mathcal{S}$ where $\text{val}_{\mathcal{S}} : (X:+ \vdash \mathcal{S}\uparrow X)$
codata $\forall_k(X:k \rightarrow -) : -$ where $\text{spec} : (\forall_k X \vdash^{Y:k} X Y:-)$	codata $\uparrow_{\mathcal{S}}(X:\mathcal{S}) : -$ where $\text{eval}_{\mathcal{S}} : (\uparrow_{\mathcal{S}} X \vdash X:\mathcal{S})$	codata $\mathcal{S}\downarrow(X:-) : \mathcal{S}$ where $\text{enter}_{\mathcal{S}} : (\mathcal{S}\downarrow X \vdash X:-)$

■ **Figure 1** The \mathcal{F} functional core set of (co-)data declarations.

combinations are also possible. Second, instead of the function type arrow notation to assign a type to the constructors and observers, we use the turnstyle (\vdash) of a typing judgement. This avoids the issue that a function type arrow already dictates the disciplines for the argument and result, limiting our freedom of choice.

The rest of the core \mathcal{F} types exercise all the functionality of our declaration mechanism. The nullary version of sums (0) has no constructors and an empty **case** M **of** $\{\}$. We have binary and nullary tuples ($\otimes, 1$), which have terms of the form (M, N) and $()$ and are used by **case** M **of** $\{(x, y).M\}$ and **case** M **of** $\{().M\}$, respectively. We also have binary and nullary products ($\&, \top$), with two and zero observers, respectively. The terms of binary products have the form $\lambda\{\pi_1.M | \pi_2.N\}$ and can be observed as $M.\pi_i$, and the nullary product has the term $\lambda\{\}$ which cannot be observed in any way. The shifts are also generalized to operate generically over the choice of call-by-name ($-$), call-by-value ($+$), and call-by-need (\star), which we denote by \mathcal{S} . The pair of shifts between $+$ ($\downarrow_{\mathcal{S}}, \mathcal{S}\uparrow$) and $-$ ($\uparrow_{\mathcal{S}}, \mathcal{S}\downarrow$) for each \mathcal{S} has the same form as in Section 3, where we omit the annotation \mathcal{S} when it is clear from the context.

The last piece of functionality is the ability to introduce *locally quantified* types in a constructor or observer. These quantified type variables are listed as a superscript to the turnstyle, and allow user-defined types to perform type abstraction and polymorphism. Two important examples of type abstraction shown in Figure 1 are the universal (\forall_k) and existential (\exists_k) quantifiers, which apply to a type function $\lambda X:k.A$. We will use the shorthand $\forall X:k.A$ for $\forall_k(\lambda X:k.A)$ and $\exists X:k.A$ for $\exists_k(\lambda X:k.A)$. The treatment of quantified types is analogous to System F_ω , where types appear in terms as parameters. For example, the term $\lambda\{\text{spec } Y:k.M\} : \forall Y:k.A$ abstracts over the type variable Y in M , and a polymorphic $M : \forall Y:k.A$ can be observed via specialization as $M.\text{spec } B : A[B/Y]$. Dually, the term $\text{pack } B M : \exists Y:k.A$ hides the type B in the term $M : A[B/Y]$, and an existential $M : \exists Y:k.A$ can be unpacked by pattern matching as **case** M **of** $\{\text{pack } (Y:k) (x:A).N\}$.

4.2 Syntax

The syntax of our extensible and multi-discipline λ -calculus is given in Figure 2. We refer to each of the three kinds of types ($+$, $-$ and \star) as a *discipline* which is denoted by the

$$\begin{aligned}
A, B, C ::= X \mid F \mid \lambda X. A \mid A B \quad X ::= X:k \quad k, l ::= S \mid k \rightarrow l \quad \mathcal{R}, \mathcal{S}, \mathcal{T} ::= + \mid - \mid \star \\
\text{decl} ::= \mathbf{data} F(X:k).. : \mathcal{S} \mathbf{where} K : (A:\mathcal{T}.. \vdash^{X..} F X..) \\
\quad \mid \mathbf{codata} G(X:k).. : \mathcal{S} \mathbf{where} O : (A:\mathcal{T}.. \mid G X.. \vdash^{X..} B:\mathcal{R}).. \\
p ::= K X..y.. \quad q ::= O X..y.. \quad x, y, z ::= x:A \\
M, N ::= x \mid \mathbf{let} x = M \mathbf{in} N \mid M.O B..N.. \mid K B..M.. \mid \lambda\{q_i.M_i^i.\} \mid \mathbf{case} M \mathbf{of} \{p_i.M_i^i.\}
\end{aligned}$$

■ **Figure 2** Syntax of a total, pure functional calculus with (co-)data.

meta-variables \mathcal{R} , \mathcal{S} , and \mathcal{T} . A data declaration has the general form

$$\begin{aligned}
\mathbf{data} F(X_1:k_1)..(X_n:k_n) : \mathcal{S} \mathbf{where} K_1 : (A_{11} : \mathcal{T}_{11}..A_{1n} : \mathcal{T}_{1n} \vdash F X_1..X_n) \\
\quad \vdots \\
\quad K_m : (A_{m1} : \mathcal{T}_{m1}..A_{mn} : \mathcal{T}_{mn} \vdash F X_1..X_n)
\end{aligned}$$

which declares a new type constructor F and value constructors $K_1 \dots K_m$. The dual co-data declaration combines the concepts of functions and products, having the general form

$$\begin{aligned}
\mathbf{codata} G(X_1:k_1)..(X_n:k_n) : \mathcal{S} \mathbf{where} O_1 : (A_{11} : \mathcal{T}_{11}..A_{1n} : \mathcal{T}_{1n} \mid G X_1..X_n \vdash B_1 : \mathcal{R}_1) \\
\quad \vdots \\
\quad O_m : (A_{m1} : \mathcal{T}_{m1}..A_{mn} : \mathcal{T}_{mn} \mid G X_1..X_n \vdash B_m : \mathcal{R}_m)
\end{aligned}$$

Since an observer is dual to a constructor, the signature is flipped around: the signature for O_1 above can be read as “given parameters of types A_{11} to A_{1n} , O_1 can observe a value of type $G X_1..X_n$ to obtain a result of type B_1 .”²

Notice that we can *also* declare types corresponding to purely call-by-value, -name, and -need versions of sums and functions by instantiating \mathcal{S} with $+$, $-$, and \star , respectively:

$$\begin{aligned}
\mathbf{data} (X:\mathcal{S}) \oplus^{\mathcal{S}} (Y:\mathcal{S}) : \mathcal{S} \mathbf{where} \quad \mathbf{codata} (X:\mathcal{S}) \xrightarrow{\mathcal{S}} (Y:\mathcal{S}) : \mathcal{S} \mathbf{where} \\
\iota_1^{\mathcal{S}} : (X:\mathcal{S} \vdash X \oplus Y) \quad \text{call}^{\mathcal{S}} : (X:\mathcal{S} \mid X \xrightarrow{\mathcal{S}} Y \vdash Y:\mathcal{S}) \\
\iota_2^{\mathcal{S}} : (Y:\mathcal{S} \vdash X \oplus Y)
\end{aligned}$$

So the extensible language subsumes *all* the languages shown in Sections 2 and 3.

4.3 Type System

The kind and type system is given in Figure 3. In the style of system F_ω , the kind system is just the simply-typed λ -calculus at the level of types – so type variables, functions, and applications – where each connective is a constant of the kind declared in the global environment \mathcal{G} . It also includes the judgement $(\Gamma \vdash_{\mathcal{F}}^{\Theta}) \mathbf{ctx}$ for checking that a typing context is well-formed, meaning that each variable in Γ is assigned a well-kinded type with respect to the type variables in Θ and global environment \mathcal{G} .

The typing judgement for terms is $\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S}$, where \mathcal{G} is a list of declarations, $\Theta = X : k..$ assigns kinds to type variables, and $\Gamma = x : A : \mathcal{S}..$ assigns explicitly-kinded types to value variables. The interesting feature of the type system is the use of the two-level

² Both of these notions of data and co-data correspond to *finitary* types, since declarations allow for a finite number of constructors or observers for all data and co-data types, respectively. We could just as well generalize declarations with an infinite number of constructors or observers to also capture *infinitary* types at the usual cost of having infinite branching in **cases** and λ s. Since this generalization is entirely mechanical and does not enhance the main argument, we leave it out of the presentation.

$$\begin{array}{c}
 \frac{\Theta, X : k \vdash_{\mathcal{G}} A : l}{\Theta \vdash_{\mathcal{G}} \lambda X : k. A : k \rightarrow l} \quad \frac{\Theta \vdash_{\mathcal{G}} A : k \rightarrow l \quad \Theta \vdash_{\mathcal{G}} B : k}{\Theta \vdash_{\mathcal{G}} A B : l} \quad \frac{}{\Theta, X : k \vdash_{\mathcal{G}} X : k} \quad \frac{(\Theta \vdash_{\mathcal{G}} A : \mathcal{T})..}{(x : A : \mathcal{T}.. \vdash_{\mathcal{G}}^{\Theta}) \text{ctx}} \\
 \frac{(\Gamma \vdash_{\mathcal{G}}^{\Theta}) \text{ctx} \quad \Theta \vdash_{\mathcal{G}} A : \mathcal{S}}{\Gamma, x : A : \mathcal{S} \vdash_{\mathcal{G}}^{\Theta} x : A : \mathcal{S}} \quad \frac{\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S} \quad \Gamma, x : A : \mathcal{S} \vdash_{\mathcal{G}}^{\Theta} N : C : \mathcal{R}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \text{let } x : A = M \text{ in } N : C : \mathcal{R}} \quad \frac{\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S} \quad A =_{\beta\eta} B}{\Gamma \vdash_{\mathcal{G}}^{\Theta} M : B : \mathcal{S}} \\
 \text{Given } \mathbf{data} \mathbf{F}(X:k).. : \mathcal{S} \text{ where } \mathbf{K}_i : (A_{ij} : \mathcal{T}_{ij}^{\dot{i}} \vdash^{Y_{ij} : l_{ij}^{\dot{j}}} \mathbf{F}(X..))_{\dot{i}} \in \mathcal{G}, \text{ we have the rules:} \\
 \frac{}{\Theta \vdash_{\mathcal{G}} \mathbf{F} : k \rightarrow ..\mathcal{S}} \\
 \frac{(\Gamma \vdash_{\mathcal{G}}^{\Theta}) \text{ctx} \quad \Theta \vdash_{\mathcal{G}} \mathbf{F} C.. : \mathcal{S} \quad (\Theta \vdash_{\mathcal{G}} B_j : l_{ij}^{\dot{j}})_{\dot{i}} \quad (\Gamma \vdash_{\mathcal{G}}^{\Theta} M_j : A_{ij}[C/X.., B_j/Y_{ij}^{\dot{j}}] : \mathcal{T}_{ij}^{\dot{j}})_{\dot{i}}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \mathbf{K}_i B_j^{\dot{j}}. M_j^{\dot{j}} : \mathbf{F} C.. : \mathcal{S}} \mathbf{FI}_i \\
 \frac{\Theta \vdash_{\mathcal{G}} C : \mathcal{R} \quad \Gamma \vdash_{\mathcal{G}}^{\Theta} M : \mathbf{F} B.. : \mathcal{S} \quad (\Gamma, x_{ij} : A_{ij}[B/X..] : \mathcal{T}_{ij}^{\dot{j}} \vdash_{\mathcal{G}}^{\Theta, Y_{ij} : l_{ij}^{\dot{j}}} N_i : C : \mathcal{R})_{\dot{i}}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \mathbf{case } M \text{ of } \{(\mathbf{K}_i Y_{ij} : l_{ij}^{\dot{j}}. x_{ij} : A_{ij}^{\dot{j}}). N_i^{\dot{i}}\} : C : \mathcal{R}} \mathbf{FE} \\
 \text{Given } \mathbf{codata} \mathbf{G}(X:k).. : \mathcal{S} \text{ where } \mathbf{O}_i : (A_{ij} : \mathcal{T}_{ij}^{\dot{i}} \mid \mathbf{G}(X..) \vdash^{Y_{ij} : l_{ij}^{\dot{j}}} B_i : \mathcal{R}_i)_{\dot{i}} \in \mathcal{G}, \text{ we have the rules:} \\
 \frac{}{\Theta \vdash_{\mathcal{G}} \mathbf{G} : k \rightarrow ..\mathcal{S}} \\
 \frac{\Gamma \vdash_{\mathcal{G}}^{\Theta} M : \mathbf{G} C'.. : \mathcal{S} \quad (\Theta \vdash_{\mathcal{G}} C_j : l_{ij}^{\dot{j}})_{\dot{i}} \quad (\Gamma \vdash_{\mathcal{G}}^{\Theta} N_j : A_{ij}[C'/X.., C_j/Y_{ij}^{\dot{j}}] : \mathcal{T}_{ij}^{\dot{j}})_{\dot{i}}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} M. \mathbf{O}_i C_j^{\dot{j}}. N_j^{\dot{j}} : B_i : \mathcal{R}_i} \mathbf{GE}_i \\
 \frac{(\Gamma \vdash_{\mathcal{G}}^{\Theta}) \text{ctx} \quad \Theta \vdash_{\mathcal{G}} \mathbf{G} C.. : \mathcal{S} \quad (\Gamma, x_{ij} : A_{ij}[C/X..] : \mathcal{T}_{ij}^{\dot{j}} \vdash_{\mathcal{G}}^{\Theta, Y_{ij} : l_{ij}^{\dot{j}}} N_i : B_i : \mathcal{R}_i)_{\dot{i}}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \lambda \{(\mathbf{O}_i Y_{ij} : l_{ij}^{\dot{j}}. x_{ij} : A_{ij}^{\dot{j}}). N_i^{\dot{i}}\} : \mathbf{G} C.. : \mathcal{S}} \mathbf{GI}
 \end{array}$$

■ **Figure 3** Type system for the pure functional calculus.

judgement $M : A : \mathcal{S}$, which has the intended interpretation that “ M is of type A and A is of kind \mathcal{S} .” The purpose of this compound statement is to ensure that the introduction rules do not create ill-kinded types by mistake. This maintains the invariant that if $\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S}$ is derivable then so is $(\Gamma \vdash_{\mathcal{G}}^{\Theta}) \text{ctx}$ and $\Theta \vdash_{\mathcal{G}} A : \mathcal{S}$.

For example, in the \mathcal{F} environment from Figure 1, a type like $A \otimes B$ requires that both A and B are of kind $+$, so the \otimes introduction rule for closed pairs of closed types is:

$$\frac{\vdash_{\mathcal{F}} M : A : + \quad \vdash_{\mathcal{F}} N : A : +}{\vdash_{\mathcal{F}} (M, N) : A \otimes B : +} \otimes I$$

The constraint that $A : +$ and $B : +$ in the premises to $\otimes I$ ensures that $A \otimes B$ is indeed a type of $+$. This idea is also extended to variables introduced by pattern matching at a specific type by placing a two-level constraint on the variables. For example, the \rightarrow introduction rule for closed function abstractions is:

$$\frac{x : A : + \vdash_{\mathcal{F}} M : B : -}{\vdash_{\mathcal{F}} \lambda \{\text{call}(x:A).M\} : A \rightarrow B : -} \rightarrow I$$

Notice how when the variable x is added to the environment, it has the type assignment $x : A : +$ because the declared argument type of \rightarrow must be some call-by-value type. If the premise of $\rightarrow I$ holds, then $A : +$ and $B : -$, so $A \rightarrow B$ is a well-formed type of $-$.

Finally, we also need to check that a global environment \mathcal{G} is well-formed, written $\vdash_{\mathcal{G}}$, which amounts to checking that each declaration is in turn like so:

$$\frac{(X : k.., Y : l.. \vdash_{\mathcal{G}} A : \mathcal{T})..}{\vdash_{\mathcal{G}} \mathbf{data} \mathbf{F}(X:k).. : \mathcal{S} \text{ where } \mathbf{K} : (A : \mathcal{T}.. \vdash^{Y:l..} \mathbf{F} X..)..} \\
 \frac{(X : k.., Y : l.. \vdash_{\mathcal{G}} A : \mathcal{T}).. \quad (X : k.., Y : l.. \vdash_{\mathcal{G}} B : \mathcal{R})..}{\vdash_{\mathcal{G}} \mathbf{codata} \mathbf{G}(X:k).. : \mathcal{S} \text{ where } \mathbf{O} : (A : \mathcal{T}.. \mid \mathbf{G} X.. \vdash^{Y:l..} B : \mathcal{R})..}$$

$$\begin{aligned}
V &::= V_S : A : \mathcal{S} & V_+ &::= x \mid \mathbf{K} B..V.. \mid \lambda\{q_i.M_i \mid \dot{i}\} & V_- &::= M & V_\star &::= V_+ \\
F &::= \square.O B..V.. \mid \mathbf{case} \square \mathbf{of} \{p_i.M_i \dot{i}\} \mid \mathbf{let} x:A:+ = \square \mathbf{in} M \mid \mathbf{let} x:A:\star = \square \mathbf{in} H[E[x]] \\
E &::= \square \mid F[E] & U &::= \mathbf{let} x:A:\star = M \mathbf{in} \square & H &::= \square \mid U[H] \\
T &::= \mathbf{let} x = M \mathbf{in} \square \mid \mathbf{case} M \mathbf{of} \{p_i.\square \mid \dot{i}\}
\end{aligned}$$

$$\begin{aligned}
(\beta_{let}) & \quad \mathbf{let} x = V \mathbf{in} M \sim M[V/x] \\
(\beta_O) & \quad \lambda\{..\mid(O Y..x..).M\mid..\}.O B.. N.. \sim \mathbf{let} x = N.. \mathbf{in} M[B/Y..] \\
(\beta_K) & \quad \mathbf{case} \mathbf{K} B..N.. \mathbf{of} \{..\mid(\mathbf{K} Y..x..).M\mid..\} \sim \mathbf{let} x = N.. \mathbf{in} M[B/Y..] \\
(\eta_{let}) & \quad \mathbf{let} x:A = M \mathbf{in} x \sim M \\
(\eta_G) & \quad \lambda\{q_i.(x.q_i) \mid \dot{i}\} \sim x \\
(\eta_F) & \quad \mathbf{case} M \mathbf{of} \{p_i.p_i \mid \dot{i}\} \sim M \\
(\kappa_F) & \quad F[T[M_i \dot{i}]] \sim T[F[M_i] \dot{i}] \\
(\chi^S) & \quad \mathbf{let} y:B:\mathcal{S} = \mathbf{let} x:A:\mathcal{S} = M_1 \mathbf{in} M_2 \mathbf{in} N \sim \mathbf{let} x:A:\mathcal{S} = M_1 \mathbf{in} \mathbf{let} y:B:\mathcal{S} = M_2 \mathbf{in} N \\
& \quad \frac{\Gamma \vdash_{\mathcal{G}}^{\ominus} M : A : \mathcal{S} \quad M \sim M' \quad \Gamma \vdash_{\mathcal{G}}^{\ominus} M' : A : \mathcal{S}}{\Gamma \vdash_{\mathcal{G}}^{\ominus} M = M' : A : \mathcal{S}}
\end{aligned}$$

plus compatibility, reflexivity, symmetry, transitivity

■ **Figure 4** Equational theory for the pure functional calculus.

And we say that \mathcal{G}' extends \mathcal{G} if it contains all declarations in \mathcal{G} .

4.4 Equational Theory

The equational theory, given in Figure 4, equates two terms of the same type that behave the same in any well-typed context. The axioms of equality are given by the relation \sim , and the typed equality judgement is $\Gamma \vdash_{\mathcal{G}}^{\ominus} M = N : A : \mathcal{S}$. Because of the multi-discipline nature of terms, the main challenge is deciding when terms are substitutable, which controls when the β_{let} axiom can fire. For example, $\mathbf{let} x = M \mathbf{in} N$ should immediately substitute M without further evaluation if it is a call-by-name binding, but should evaluate M to a value first before substitution if it is call-by-value. And we need the ability to reason about program fragments (*i.e.*, open terms of any type) wherein a variable x acts like a value in call-by-value *only* if it stands for a value, *i.e.*, we can only substitute values and not arbitrary terms for a call-by-value variable. Thus, we link up the static and dynamic semantics of disciplines: each base kind \mathcal{S} is associated with a different set of substitutable terms V_S called *values*. The set of values for $+$ is the most strict (including only variables, λ -abstractions, and constructions $p[\rho]$ built by plugging in values for the holes in a pattern), $-$ is the most relaxed (admitting every term as substitutable), and \star shares the same notion of value as $+$. A true value, then, is a term V_S belonging to a type of kind \mathcal{S} , *i.e.*, $V_S : A : \mathcal{S}$. This way, the calling convention is aligned in both the static realm of types and dynamic realm of evaluation.

The generic β_{let} axiom relies on the fact that the left-hand side of the axiom is well-typed and every type belongs to (at most) one kind; given $\mathbf{let} x:A = V \mathbf{in} M$, then it must be that $A : \mathcal{S}$ and V is of the form $V_S : A : \mathcal{S}$ (both in the current environment). So if $x : A \& B : -$, then every well-typed binding is subject to substitution via β_{let} , but if $x : A \otimes B : +$ then only a value V_+ in the sense of call-by-value can be substituted. The corresponding extensionality axiom η_{let} eliminates a trivial \mathbf{let} binding.

The β_K and β_O axioms match against a constructor \mathbf{K} or observer \mathbf{O} , respectively, by

21:12 Beyond Polarity

selecting the matching response within a **case** or λ -abstraction and binding the parameters via a **let**. Special cases of these axioms for a sum injection and function call are:

$$\begin{aligned} \mathbf{case} \iota_i M \mathbf{of} \{ \iota_1 x_1.N_1 \mid \iota_2 x_2.N_2 \} &\sim_{\beta_{\iota_i}} \mathbf{let} x_i = M \mathbf{in} N_i \\ \lambda \{ \mathbf{call} x.N \}. \mathbf{call} M &\sim_{\beta_{\mathbf{call}}} \mathbf{let} x = M \mathbf{in} N \end{aligned}$$

The corresponding extensionality axioms η_G and η_F apply to each co-data type G and data type F to eliminate a trivial λ and **case**, respectively, and again rely on the fact that the left-hand side of the axiom is well-typed to be sensible. The special cases of these axioms for the sum (\oplus) and function (\rightarrow) connectives of \mathcal{F} are:

$$\mathbf{case} M \mathbf{of} \{ \iota_1 x:A.\iota_1 x \mid \iota_2 y:B.\iota_2 y \} \sim_{\eta_{\oplus}} M \qquad \lambda \{ \mathbf{call} y:A.(x.\mathbf{call} y) \} \sim_{\eta_{\rightarrow}} x$$

The κ_F axiom implements *commutative conversions* which permute a *frame* F of an evaluation context (E) with a *tail* context T , which brings together the frame with the return result of a block-style expression like a **let** or **case**. Frames represent the building blocks of contexts that demand a result from their hole \square . The cases for frames are an observation parameterized by values ($\square.OB.V..$), case analysis (**case** \square **of** $\{ \dots \}$), a call-by-value binding (**let** $x:A: + = \square$ **in** M), or a call-by-need binding which is needed in its body (**let** $x:A:\star = \square$ **in** $H[E[x]]$). As per call-by-need evaluation, variable x is *needed* when it appears in the eye of an evaluation context E , in the context of a *heap* H of other call-by-need bindings for different variables. Tail contexts point out where results are returned from block-style expressions, so the body of any **let** (**let** $x = M$ **in** \square) or the branches of any **case** (**case** M **of** $\{ p.\square \}$). Since a **case** can have zero or more branches, a tail context can have zero or more holes.

Finally, the χ^S axiom re-associates nested **let** bindings, so long as the discipline of their bindings match. The restriction to matching disciplines is because not all combinations are actually associative [14]; namely the following two ways of nesting call-by-value and -name **lets** are *not* necessarily the same when M_1 causes an effect:

$$(\mathbf{let} y:B:- = (\mathbf{let} x:A:+ = M_1 \mathbf{in} M_2) \mathbf{in} N) \neq (\mathbf{let} x:A:+ = M_1 \mathbf{in} \mathbf{let} y:B:- = M_2 \mathbf{in} N)$$

In the above, the right-hand side evaluates M_1 first, but the left-hand side first substitutes **let** $x:A:+ = M_1$ **in** M_2 for y , potentially erasing or duplicating the effect of M_1 . For example, when M_1 is the infinite loop Ω and N is a constant result z which does not depend on y , then the right-hand side loops forever, but the left-hand side just returns z . But when the disciplines match, re-association is sound. In particular, notice that the χ^- instance of the axiom is derivable from $\beta_{\mathbf{let}}$, and the χ^+ instance of the axiom is derivable from κ_F . The only truly novel instance of re-association is for call-by-need, which generalizes the special case of κ_F when the outer variable y happens to be needed.

Some of the axioms of this theory may appear to be weak, but nonetheless they let us derive some useful equalities. For example, the λ -calculus' full η law for functions

$$\frac{\Gamma \vdash_{\mathcal{F}}^{\ominus} M : A \rightarrow B : - \quad x \notin \Gamma}{\Gamma \vdash_{\mathcal{F}}^{\ominus} \lambda \{ \mathbf{call} x:A.(M.\mathbf{call} x) \} = M : A \rightarrow B : -}$$

is derivable from η_{\rightarrow} and $\beta_{\mathbf{let}}$. Furthermore, the sum extensionality law from Section 2, and nullary version for the void type 0

$$\begin{aligned} \Gamma, x : A_1 \oplus A_2 : + \vdash_{\mathcal{F}}^{\ominus} M = \mathbf{case} x \mathbf{of} \{ \iota_i (y_i:A_i).M[\iota_i y_i/x]^{\ddagger} \} : C : \mathcal{R} \\ \Gamma, x : 0 : + \vdash_{\mathcal{F}}^{\ominus} M = \mathbf{case} x \mathbf{of} \{ \} : C : \mathcal{R} \end{aligned}$$

are derived from the η_{\oplus} , η_0 , κ_F , and β_{let} axioms. So typed equality of this strongly-normalizing calculus captures “strong sums” (à la [15]). Additionally, the laws of monadic binding [13] (bind-and-return and bind reassociation) and the F functor of call-by-push-value [10] are instances of the generic $\beta\eta\kappa$ laws for the shift data type $\mathcal{S}\uparrow A$:

$$\begin{aligned} \Gamma \vdash_{\mathcal{F}}^{\Theta} \mathbf{case} \mathbf{box}_{\mathcal{S}} V \mathbf{of} \{ \mathbf{box}_{\mathcal{S}} \mathbf{x}. M \} &=_{\beta_{\mathcal{S}\uparrow} \beta_{let}} M[V/\mathbf{x}] : C : \mathcal{R} \\ \Gamma \vdash_{\mathcal{F}}^{\Theta} \mathbf{case} M \mathbf{of} \{ \mathbf{box}_{\mathcal{S}}(x:A). \mathbf{box}_{\mathcal{S}} x \} &=_{\eta_p} M : \mathcal{S}\uparrow A : \mathcal{S} \\ \Gamma \vdash_{\mathcal{F}}^{\Theta} \mathbf{case} (\mathbf{case} M \mathbf{of} \{ \mathbf{box}_{\mathcal{S}} \mathbf{x}. N \}) \mathbf{of} \{ \mathbf{box}_{\mathcal{T}} \mathbf{y}. N' \} & : C : \mathcal{R} \\ &=_{\kappa_F} \mathbf{case} M \mathbf{of} \{ \mathbf{box}_{\mathcal{S}} \mathbf{x}. \mathbf{case} N \mathbf{of} \{ \mathbf{box}_{\mathcal{T}} \mathbf{y}. N' \} \} \end{aligned}$$

Note that in the third equality, commuting conversions can reassociate $\mathcal{S}\uparrow A$ and $\mathcal{T}\uparrow B$ bindings for *any* combination of \mathcal{S} and \mathcal{T} , including $-$ and \star , because a **case** is *always* strict.

Note that, as usual, the equational theory collapses under certain environments and types due to the nullary versions of some connectives: we saw above that with a free variable $x : 0 : +$ all terms are equal, and so too are any two terms of type \top via η_{\top} (the nullary form of product in \mathcal{F}). Even still, there are many important cases where the equational theory is coherent. One particular sanity check is that, in the absence of free variables, the two sum injections $\iota_1()$ and $\iota_2()$ are not equal, as inherited from contextual equivalence.

► **Theorem 1** (Closed coherence). *For any global environment $\vdash \mathcal{G}$ extending \mathcal{F} , the equality $\vdash_{\mathcal{G}} \iota_1() = \iota_2() : 1 \oplus 1 : +$ is not derivable.*

4.5 Adding effects

So far, we have considered only a pure functional calculus. However, one of the features of polarity is its robustness in the face of computational effects, so let’s add some. Two particular effects we can add are *general recursion*, in the form of fixed points, and *control* in the form of μ -abstractions from Parigot’s $\lambda\mu$ -calculus [16]. To do so, we extend the calculus with the following syntax:

$$M, N ::= \dots \mid \nu \mathbf{x}. M \mid \mu \alpha. J \qquad J ::= \langle M \parallel \alpha \rangle \qquad \alpha, \beta, \gamma ::= \alpha : A$$

Fixed-point terms $\nu \mathbf{x}. A.M$ bind x to the result of M inside M itself. Because fixed points must be unrolled before evaluating their underlying term, their type is restricted to $A : -$. Control extends the calculus with *co-variables* α, β, \dots that bind to *evaluation contexts* instead of values, letting programs abstract over and manipulate their control flow. The evaluation context bound to a co-variable α of any type A can be invoked (any number of times) with a term $M : A$ via a jump $\langle M \parallel \alpha \rangle$ that never returns a result, and the co-variable α of type A can be bound with a μ -abstraction $\mu \alpha. A.J$.

To go along with the new syntax, we have some additional type checking rules:

$$\frac{\Gamma, x : A : - \vdash_{\mathcal{G}}^{\Theta} M : A : - \mid \Delta}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \nu \mathbf{x}. A.M : A : - \mid \Delta} \qquad \frac{J : (\Gamma \vdash_{\mathcal{G}}^{\Theta} \alpha : A : \mathcal{S}, \Delta)}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \mu \alpha. A.J : A : \mathcal{S} \mid \Delta} \qquad \frac{\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S} \mid \alpha : A : \mathcal{S}, \Delta}{\langle M \parallel \alpha \rangle : (\Gamma \vdash_{\mathcal{G}}^{\Theta} \alpha : A : \mathcal{S}, \Delta)}$$

The judgements in other typing rules from Figure 3 are all generalized to $\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S} \mid \Delta$. There is also a typing judgement for jumps of the form $J : (\Gamma \vdash_{\mathcal{F}}^{\Theta} \Delta)$, where Θ , Γ , and Δ play the same roles; the only difference is that J is not given a type for its result. Unlike terms, jumps never return. As in the $\lambda\mu$ -calculus, the environment Δ is placed on the right because co-variables represent alternative return paths. For example, a term

21:14 Beyond Polarity

$x : X : -, y : Y : + \vdash_{\mathcal{F}}^{X:-, Y:+} M : Y : - \mid \beta : Y : +$ could return an X via the main path, as in $M = x$, or a Y via β by aborting the main path, as in $M = \mu\alpha.X.\langle y \parallel \beta \rangle$.

And finally, the equational theory is also extended with the following equality axioms:

$$\begin{array}{ll}
 (\nu) & \nu x.M \sim M[\nu x.M/x] \\
 (\beta_\mu^\alpha) & \langle \mu\alpha.J \parallel \beta \rangle \sim J[\beta/\alpha] \qquad (\beta_\mu^F) & F[\mu\alpha.J] : B \sim \mu\beta.B.J[\langle F \parallel \beta \rangle / \langle \square \parallel \alpha \rangle] \\
 (\eta_\mu) & \mu\alpha.A.\langle M \parallel \alpha \rangle \sim M \qquad (\kappa_\mu) & T[\mu\alpha.\langle M_i \parallel \beta \rangle^i] \sim \mu\alpha.\langle T[M_i^i] \parallel \beta \rangle
 \end{array}$$

The ν axiom unrolls a fixed point by one step. The two β_μ axioms are standard generalizations of the $\lambda\mu$ -calculus: β_μ^α substitutes one co-variable for another, and β_μ^F captures a single frame of a μ -abstraction's evaluation context via a *structural substitution* that replaces one context with another. The κ_μ is the commuting conversion that permutes a μ -abstraction with a tail context T .

5 Encoding user-defined (co-)data types into \mathcal{F}

Equipped with both the extensible source language and the fixed \mathcal{F} target language, we are now able to give an encoding of user-defined (co-)data types in terms of just the core \mathcal{F} connectives from Figure 1. Intuitively, each data type is converted to an existential \oplus -sum-of- \otimes -products and each co-data type is converted to a universal $\&$ -product-of-functions, both annotated by the necessary shifts in and out of $+$ and $-$, respectively. The encoding is parameterized by a global environment \mathcal{G} so that we know the overall shape of each declared connective. Given that \mathcal{G} contains the following data declaration of F , the encoding of F is:

$$\begin{array}{l}
 \text{Given } \mathbf{data} \ F(X:k).. : \mathcal{S} \ \mathbf{where} \ K_i : (A_{ij} : \mathcal{T}_{ij}^j \vdash^{Y_{ij}:l_{ij}^j} F(X..))^i \in \mathcal{G} \\
 \llbracket F \rrbracket_{\mathcal{G}}^{\mathcal{F}} \triangleq \lambda X:k... \mathcal{S} \uparrow ((\exists Y_{ij}:l_{ij}^j..^j. ((\downarrow_{\mathcal{T}_{ij}} A_{ij}) \otimes ^j.1)) \oplus ^i.0)
 \end{array}$$

Dually, given that \mathcal{G} contains the following co-data declaration of G , the encoding of G is:

$$\begin{array}{l}
 \text{Given } \mathbf{codata} \ G(X:k).. : \mathcal{S} \ \mathbf{where} \ O_i : (A_{ij} : \mathcal{T}_{ij}^j \mid G(X..) \vdash^{Y_{ij}:l_{ij}^j} B_i : \mathcal{R}_i)^i \in \mathcal{G} \\
 \llbracket G \rrbracket_{\mathcal{G}}^{\mathcal{F}} \triangleq \lambda X:k... \mathcal{S} \downarrow ((\forall Y_{ij}:l_{ij}^j..^j. ((\downarrow_{\mathcal{T}_{ij}} A_{ij}) \rightarrow ^j. (\uparrow_{\mathcal{R}_i} B_i))) \& ^i. \top)
 \end{array}$$

However, the previous encodings for call-by-name, -value, and -need functions and sums from Sections 2 and 3 are not exactly the same when we take the corresponding declarations of functions and sums from Section 4; the call-by-name and -value encodings are missing some of the shifts used by the generic encoding, and they all elide the terminators (0, 1, and \top). Does the difference matter? No, because the encoded types are still *isomorphic*.

► **Definition 2** (Type Isomorphism). An isomorphism between two open types of kind k , written $\Theta \vDash_{\mathcal{G}} A \approx B : k$, is defined by induction on k :

- $\Theta \vDash_{\mathcal{G}} A \approx B : k \rightarrow l$ when $\Theta, X : k \vDash_{\mathcal{G}} A \ X \approx B \ X : l$, and
- $\Theta \vDash_{\mathcal{G}} A \approx B : \mathcal{S}$ when, for any x and y , there are terms $x : A : \mathcal{S} \vdash_{\mathcal{G}}^{\Theta} N : B : \mathcal{S}$ and $y : B : \mathcal{S} \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S}$ such that $x:A:\mathcal{S} \vdash_{\mathcal{G}}^{\Theta} (\mathbf{let} \ y:B = N \ \mathbf{in} \ M = x) : A : \mathcal{S}$ and $y:B:\mathcal{S} \vdash_{\mathcal{G}}^{\Theta} (\mathbf{let} \ x:A = M \ \mathbf{in} \ N = y) : B : \mathcal{S}$.

Notice that this is an *open* form of isomorphism: in the base case, an isomorphism between types with free variables is witnessed *uniformly* by a single pair of terms. This uniformity in the face of polymorphism is used to make type isomorphism compatible with the \forall and \exists quantifiers. With this notion of type isomorphism, we can formally state how some of the

specific shift connectives are redundant. In particular, within the positive (+) and negative (−) subset, there are only two shifts of interest since the two different shifts between − and + are isomorphic, and the identity shifts on + and − are isomorphic to an identity on types.

► **Theorem 3.** *The following isomorphisms hold (under $\vDash_{\mathcal{F}}$) for all $\vdash A : +$ and $\vdash B : -$*

$$\uparrow_+ A \approx _ \uparrow A \quad \downarrow_- B \approx _ \downarrow B \quad \downarrow_+ A \approx A \approx _ \uparrow A \quad \uparrow_- B \approx B \approx _ \downarrow B$$

But clearly the shifts involving \star are not isomorphic, since none of them even share the same kind. Recognizing that sometimes the generic encoding uses unnecessary identity shifts, and given the algebraic properties of polarized types [6], the hand-crafted encodings $\llbracket A \rrbracket^+$, $\llbracket A \rrbracket^-$, and $\llbracket A \rrbracket^*$ are isomorphic to $\llbracket A \rrbracket^{\mathcal{F}}$.

5.1 Correctness of encoding

Type isomorphisms give us a helpful assurance that the encoding of user-defined (co-)data types into \mathcal{F} is actually a faithful one. In every extension of \mathcal{F} with user-defined (co-)data types, all types are isomorphic to their encoding.

► **Theorem 4.** *For all $\vdash \mathcal{G}$ extending \mathcal{F} and $\Theta \vdash_{\mathcal{G}} A : k$, $\Theta \vDash_{\mathcal{G}} A \approx \llbracket A \rrbracket_{\mathcal{G}}^{\mathcal{F}} : k$.*

Note that this isomorphism is witnessed by terms in the totally pure calculus (without fixed points or μ -abstractions); the encoding works *in spite of* recursion and control, not because of it. Because of the type isomorphism, we can extract a two-way embedding between terms of type A and terms of the encoded type $\llbracket A \rrbracket_{\mathcal{G}}^{\mathcal{F}}$ from the witnesses of the type isomorphism. By the properties of isomorphisms, this embedding respects equalities between terms; specifically it is a certain kind of adjunction called an *equational correspondence* [20].

► **Theorem 5.** *For all isomorphic types $\Theta \vDash_{\mathcal{G}} A \approx B : \mathcal{S}$, the terms of type A (i.e., $\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S} \mid \Delta$) are in equational correspondence with terms of type B (i.e., $\Gamma \vdash_{\mathcal{G}}^{\Theta} N : B : \mathcal{S} \mid \Delta$).*

This means is that, in the context of a larger program, a single sub-term can be encoded into the core \mathcal{F} connectives without the rest of the program being able to tell the difference. This is useful in optimizing compilers for functional languages which change the interface of particular functions to improve performance, without hampering further optimizations.

The possible application of this encoding in a compiler is as an intermediate language: rather than encoding just one sub-term, exhaustively encoding the whole term translates from a source language with user-defined (co-)data types into the core \mathcal{F} connectives. The essence of this translation is seen in the way patterns and co-patterns are transformed; given the same generic (co-)data declarations listed in Figure 3, the encodings of (co-)patterns are:

$$\begin{aligned} \llbracket \mathbb{K}_i Y.. x.. \rrbracket_{\mathcal{G}}^{\mathcal{F}} &\triangleq \text{val}_{\mathcal{S}} (\iota_2^i (\iota_1 (\text{pack } Y.. (\text{box}_{\mathcal{T}} x, ..)))) \\ \llbracket \mathbb{O}_i Y.. x.. \rrbracket_{\mathcal{G}}^{\mathcal{F}} &\triangleq \text{enter}_{\mathcal{S}} . \pi_2^i . \pi_1 . \text{spec } Y.. \text{call } x.. \text{eval}_{\mathcal{R}_i} \end{aligned}$$

where ι_2^i denotes i applications of the ι_2 constructor, and π_2^i denotes i projections of the π_2 observer. Using this encoding of (co-)patterns, we can encode (co-)pattern-matching as:

$$\llbracket \text{case } M \text{ of } \{p_i . N_i^i\} \rrbracket_{\mathcal{G}}^{\mathcal{F}} \triangleq \text{case } \llbracket M \rrbracket_{\mathcal{G}}^{\mathcal{F}} \text{ of } \{ \llbracket p_i \rrbracket_{\mathcal{G}}^{\mathcal{F}} . \llbracket N_i \rrbracket_{\mathcal{G}}^{\mathcal{F}} \} \quad \llbracket \lambda \{q_i . M_i^i\} \rrbracket_{\mathcal{G}}^{\mathcal{F}} \triangleq \lambda \{ \llbracket q_i \rrbracket_{\mathcal{G}}^{\mathcal{F}} . \llbracket M_i \rrbracket_{\mathcal{G}}^{\mathcal{F}} \}$$

as well as data structures and co-data observations:

$$\begin{aligned} \llbracket p[B/Y.., M/x..] \rrbracket_{\mathcal{G}}^{\mathcal{F}} &\triangleq \llbracket p \rrbracket_{\mathcal{G}}^{\mathcal{F}} [\llbracket B \rrbracket_{\mathcal{G}}^{\mathcal{F}} / Y.., \llbracket M \rrbracket_{\mathcal{G}}^{\mathcal{F}} / x..] \\ \llbracket M.(q[B/Y.., N/x..]) \rrbracket_{\mathcal{G}}^{\mathcal{F}} &\triangleq \llbracket M \rrbracket_{\mathcal{G}}^{\mathcal{F}} . (\llbracket q \rrbracket_{\mathcal{G}}^{\mathcal{F}} [\llbracket B \rrbracket_{\mathcal{G}}^{\mathcal{F}} / Y.., \llbracket N \rrbracket_{\mathcal{G}}^{\mathcal{F}} / x..]) \end{aligned}$$

Note that in the above translation, arbitrary *terms* are substituted instead of just *values* as usual. This encoding of terms with user-defined (co-)data types \mathcal{G} into the core \mathcal{F} types is sound with respect to the equational theory (where Γ and Δ are encoded pointwise).

► **Theorem 6.** *If the global environment $\vdash \mathcal{G}$ extends \mathcal{F} and $\Gamma \vdash_{\mathcal{G}}^{\ominus} M = N : A \mid \Delta$ then $\llbracket \Gamma \rrbracket_{\mathcal{G}}^{\mathcal{F}} \vdash_{\mathcal{F}}^{\ominus} \llbracket M \rrbracket_{\mathcal{G}}^{\mathcal{F}} = \llbracket N \rrbracket_{\mathcal{G}}^{\mathcal{F}} : \llbracket A \rrbracket_{\mathcal{G}}^{\mathcal{F}} \mid \llbracket \Delta \rrbracket_{\mathcal{G}}^{\mathcal{F}}$.*

Since the extensible, multi-discipline language is general enough to capture call-by-value, -name, and -need functional languages – or any combination thereof – this encoding establishes a uniform translation from both ML-like and Haskell-like languages into a common core intermediate language: the polarized \mathcal{F} .

6 Conclusion

We have showed here how the idea of polarity can be extended with other calling conventions like call-by-need, which opens up its applicability to the implementation of practical functional languages. In particular, we would like to extend GHC’s already multi-discipline intermediate language with the core types in \mathcal{F} . Since it already has unboxed types [18] corresponding to positive types, what remains are the fully extensional negative types. Crucially, we believe that negative function types would lift the idea of *call arity* – the number of arguments a function takes before “work” is done – from the level of terms to the level of types. Call arity is used to optimize curried function calls, since passing multiple arguments at once is more efficient than computing intermediate closures as each argument is passed one at a time. No work is done in a negative type until receiving an *eval* request or unpacking a *val*, so polarized types compositionally specify multi-argument calling conventions.

For example, a binary function on integers would have the type $\text{Int} \rightarrow \text{Int} \rightarrow \uparrow \text{Int}$, which only computes when both arguments are given, versus the type $\text{Int} \rightarrow \uparrow_{*} \downarrow (\text{Int} \rightarrow \uparrow \text{Int})$ which specifies work is done after the first argument, breaking the call into two steps since a closure must be evaluated and followed. This generalizes the existing treatment of function closures in call-by-push-value to call-by-need closures. The advantage of lifting this information into types is so that call arity can be taken advantage of in higher order functions. For example, the *zipWith* function takes a binary function to combine two lists, pointwise, and has the type $\forall X:*. \forall Y:*. \forall Z:*. (X \rightarrow Y \rightarrow Z) \rightarrow [X] \rightarrow [Y] \rightarrow [Z]$. The body of *zipWith* does not know the call arity of the function it’s given, but in the polarized type built with negative functions: $\forall X:*. \forall Y:*. \forall Z:*. \downarrow (\downarrow X \rightarrow \downarrow Y \rightarrow \uparrow Z) \rightarrow \downarrow [X] \rightarrow \downarrow [Y] \rightarrow \uparrow [Z]$ the interface in the type spells out that the higher-order function uses the faster two-argument calling convention.

References

- 1 Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992. doi:10.1093/logcom/2.3.297.
- 2 Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, New York, NY, USA, 1992.
- 3 Zena M. Ariola, Hugo Herbelin, and Alexis Saurin. Classical call-by-need and duality. In *Typed Lambda Calculi and Applications: 10th International Conference, TLCA’11*, pages 27–44, Berlin, Heidelberg, jun 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-21691-6_6.
- 4 Zena M. Ariola, John Maraist, Martin Odersky, Matthias Felleisen, and Philip Wadler. A call-by-need lambda calculus. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL ’95*, pages 233–246, New York, NY, USA, 1995. ACM. doi:10.1145/199448.199507.

- 5 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, ICFP '00, pages 233–243, New York, NY, USA, 2000. ACM. doi:10.1145/351240.351262.
- 6 Paul Downen. *Sequent Calculus: A Logic and a Language for Computation and Duality*. PhD thesis, University of Oregon, 2017.
- 7 Paul Downen and Zena M. Ariola. The duality of construction. In Zhong Shao, editor, *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*, volume 8410 of *Lecture Notes in Computer Science*, pages 249–269. Springer Berlin Heidelberg, Berlin, Heidelberg, apr 2014. doi:10.1007/978-3-642-54833-8_14.
- 8 Paul Downen and Zena M. Ariola. A tutorial on computational classical logic and the sequent calculus. *Journal of Functional Programming*, 28:e3, 2018. doi:10.1017/S0956796818000023.
- 9 Tatsuya Hagino. A typed lambda calculus with categorical type constructors. In David H. Pitt, Axel Poigné, and David E. Rydeheard, editors, *Category Theory and Computer Science*, pages 140–157, Berlin, Heidelberg, sep 1987. Springer Berlin Heidelberg. doi:10.1007/3-540-18508-9_24.
- 10 Paul Blain Levy. *Call-By-Push-Value*. PhD thesis, Queen Mary and Westfield College, University of London, 2001.
- 11 Paul Blain Levy. *Jumbo λ -Calculus*, pages 444–455. Springer Berlin Heidelberg, Berlin, Heidelberg, jul 2006. doi:10.1007/11787006_38.
- 12 Luke Maurer, Paul Downen, Zena M. Ariola, and Simon Peyton Jones. Compiling without continuations. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '17, pages 482–494, New York, NY, USA, jun 2017. ACM. doi:10.1145/3062341.3062380.
- 13 Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 14–23, Piscataway, NJ, USA, 1989. IEEE Press. URL: <http://dl.acm.org/citation.cfm?id=77350.77353>.
- 14 Guillaume Munch-Maccagnoni. *Syntax and Models of a non-Associative Composition of Programs and Proofs*. PhD thesis, Université Paris Diderot, 2013.
- 15 Guillaume Munch-Maccagnoni and Gabriel Scherer. Polarised intermediate representation of lambda calculus with sums. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS 2015, pages 127–140. IEEE, jul 2015. doi:10.1109/LICS.2015.22.
- 16 Michel Parigot. $\lambda\mu$ -calculus: An algorithmic interpretation of classical natural deduction. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning: International Conference*, LPAR '92, pages 190–201, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. doi:10.1007/BFb0013061.
- 17 Simon Peyton Jones and Erik Meijer. Henk: a typed intermediate language. In *Proceedings of the First International Workshop on Types in Compilation*, 1997.
- 18 Simon L. Peyton Jones and John Launchbury. Unboxed values as first class citizens in a non-strict functional language. In John Hughes, editor, *Functional Programming Languages and Computer Architecture: 5th ACM Conference*, pages 636–666, Berlin, Heidelberg, aug 1991. Springer Berlin Heidelberg. doi:10.1007/3540543961_30.
- 19 Gordon D. Plotkin. Call-by-name, call-by-value and the λ -calculus. *Theoretical Computer Science*, 1:125–159, 1975. doi:10.1016/0304-3975(75)90017-1.
- 20 Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3-4):289–360, nov 1993. doi:10.1007/BF01019462.
- 21 Philip Wadler. Call-by-value is dual to call-by-name. In *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, pages 189–201, New York, NY, USA, 2003. ACM. doi:10.1145/944705.944723.

- 22 Noam Zeilberger. On the unity of duality. *Annals of Pure and Applied Logic*, 153(1):660–96, 2008. doi:10.1016/j.apal.2008.01.001.
- 23 Noam Zeilberger. *The Logical Basis of Evaluation Order and Pattern-Matching*. PhD thesis, Carnegie Mellon University, 2009.

A Related Work

There have been several polarized languages [10, 23, 14], each with subtly different and incompatible restrictions on which programs are allowed to be written. The most common such restriction corresponds to *focusing* in logic [1]; focusing means that the parameters to constructors and observers *must* be values. Rather than impose a static focusing restriction on the syntax of programs, we instead imply a dynamic focusing behavior – evaluate the parameters of constructors and observers before (co-)pattern matching – during execution. Both static and dynamic notions of focusing are two sides of the same coin [8].

Other restrictions vary between different frameworks. First, where computation can happen? In Levy’s call-by-push-value (CBPV) [10], value types (corresponding to positive types) only describe values and computation can only occur at computation types (corresponding to negative types), but in Munch-Maccagnoni’s system L [14] computation can occur at *any* type. Zeilberger’s calculus of unity (CU) [22], which is based on the classical sequent calculus, isolates computation in a separate syntactic category of *statements* which do not have a return type. But both CU and CBPV only deal with *substitutable* entities, to the exclusion of named computations which may not be duplicated or deleted. Second, what types can variables have? In CBPV variables always have positive types, but in CU variables have negative types or positive *atomic* types (and dually co-variables have positive types or negative atomic types). These restrictions explain why the two frameworks chose their favored shifts: \uparrow introduces a positive variable and \downarrow introduces a negative one, and in the setting of the sequent calculus \downarrow introduces a negative co-variable and \uparrow introduces a positive one. They also explain CU’s pattern matching: if there cannot be positive variables, then pattern matching *must* continue until it reaches something non-decomposable like a λ -abstraction. In contrast, system L has no restrictions on the types of (co-)variables.

In both of these ways, the language presented here is spiritually closest to system L. One reason is that call-by-need forces more generality into the system: if there is neither computation nor variables of call-by-need types, then there is no point of sharing work. However, the call-by-value and -name sub-language can still be reduced down to the more restrictive style of CBPV and CU. We showed here that the two styles of positive and negative shifts are isomorphic, so the only difference is reduction to the appropriate normal form. Normalizing the dynamic focusing reductions – originally named ς [21] – along with commuting conversions (κ) and let substitution (β_{let}) is a transformation into a focused term of negative type (where a shift can be applied for positive terms). Negative variables $x:A:-$ are eliminated by substituting $y.enter$ for x where $y:\downarrow A:+$, and the (co-)variables forbidden in CU can be eliminated by type-directed η -expansion into nested (co-)patterns.

The data and co-data mechanism used here extends the “jumbo” connectives of Levy’s jumbo λ -calculus [11] to include a treatment of call-by-need as well the move from mono-discipline to multi-discipline. Our notion of (co-)data is also similar to Zeilberger’s [23] definition of types via (co-)patterns, which is fully dual, extended with sharing.

Simple (co-)data types	
data $(X:+) \oplus (Y:+) : +$ where $\iota_1 : (X:+ \vdash X \oplus Y)$ $\iota_2 : (Y:+ \vdash X \oplus Y)$	data $0 : +$ where
data $(X:+) \otimes (Y:+) : +$ where $(-, -) : (X:+, Y:+ \vdash X \otimes Y)$	data $1 : +$ where $() : (\vdash 1)$
codata $(X:-) \& (Y:-) : -$ where $\pi_1 : (X \& Y \vdash X:-)$ $\pi_2 : (X \& Y \vdash Y:-)$	codata $\top : -$ where
codata $(X:-) \wp (Y:-) : -$ where $[-, -] : (X \wp Y \vdash X:-, Y:-)$	codata $\perp : -$ where $[] : (\perp \vdash)$
data $\ominus(X:-) : +$ where $\text{cont} : (\vdash \ominus X X:-)$	codata $\neg(X:+) : -$ where $\text{throw} : (X:+ \neg X \vdash)$
Quantifier (co-)data types	
data $\exists_k(X:k \rightarrow +) : +$ where $\text{pack} : (X Y:+ \vdash^{Y:k} \exists_k X)$	codata $\forall_k(X:k \rightarrow -) : -$ where $\text{spec} : (\forall_k X \vdash^{Y:k} X Y:-)$
Polarity shift (co-)data types	
data $\downarrow_S(X:S) : +$ where $\text{box}_S : (X:S \vdash \downarrow_S X)$	data $\uparrow_S(X:+) : S$ where $\text{val}_S : (X:+ \vdash \uparrow_S X)$
codata $\uparrow_S(X:S) : -$ where $\text{eval}_S : (\uparrow_S X \vdash X:S)$	codata $\downarrow_S(X:-) : S$ where $\text{enter}_S : (\downarrow_S X \vdash X:-)$

■ **Figure 5** The \mathcal{D} dual core set of (co-)data declarations.

B A dual multi-discipline sequent calculus

So far, we have seen how the extensible functional calculus enables multi-discipline programming and can represent many user-defined types with mixed disciplines via encodings. The advantage of this calculus is that it's close to an ordinary core calculus for functional programs, but the disadvantage is its incomplete *symmetries*. Most \mathcal{F} types have a dual counterpart ($\&$ and \oplus , \forall and \exists , *etc.*,) but types like \otimes and \rightarrow do not. The disciplines $+$ and $-$ represent opposite calling conventions, but the opposite of call-by-need (\star) is missing. To complete the picture, we now consider a fully *dual* calculus, which is based on the symmetric setting of the classical sequent calculus.

B.1 The dual core intermediate language: \mathcal{D}

In contrast with functional (co-)data declarations, dual calculus allows for symmetric data and co-data type declarations that are properly dual to one another: they can have multiple inputs to the left (of \vdash) and multiple outputs to the right (of \vdash). This dual notion of (co-)data

$$\begin{aligned}
A, B, C &::= X \mid F \mid \lambda X.A \mid A B & \mathbf{X}, \mathbf{Y}, \mathbf{Z} &::= X:k \quad k, l ::= S \mid k \rightarrow l & \mathcal{R}, \mathcal{S}, \mathcal{T} &::= + \mid - \mid * \mid \star \\
\text{decl} &::= \mathbf{data} F X:k.. : S \mathbf{where} K : (A : \mathcal{T}.. \vdash^Y.. F X.. \mid B : \mathcal{R}..) \\
&\quad \mid \mathbf{codata} G X:k.. : S \mathbf{where} O : (A : \mathcal{T}.. \mid G X.. \vdash^Y.. B : \mathcal{R}..) \\
c &::= \langle v \parallel e \rangle \\
v &::= x \mid \mu \alpha.c \mid \nu x.v \mid \lambda \{q_i.c_i \mid \dot{i}\} \mid K A..e..v.. & p &::= K Y.. \alpha.. x.. & \mathbf{x}, \mathbf{y}, \mathbf{z} &::= x:A \\
e &::= \alpha \mid \tilde{\mu} x.c \mid \tilde{\nu} \alpha.e \mid \tilde{\lambda} \{p_i.c_i \mid \dot{i}\} \mid O A..v..e.. & q &::= O Y.. x.. \alpha.. & \alpha, \beta, \delta &::= \alpha:A
\end{aligned}$$

■ **Figure 6** Syntax of the dual calculus.

is strictly more expressive, and lets us declare the new connectives like so:

$$\begin{array}{ll}
\mathbf{codata} (X:-) \wp (Y:-) : - \mathbf{where} & \mathbf{codata} \perp : - \mathbf{where} \\
[-, -] : (\mid X \wp Y \vdash X : -, Y : -) & [] : (\mid \perp \vdash) \\
\mathbf{data} \ominus (X:-) : + \mathbf{where} & \mathbf{codata} \neg (X:+) : - \mathbf{where} \\
\mathbf{cont} : (\vdash \ominus X \mid X : -) & \mathbf{throw} : (X : + \mid \neg X \vdash)
\end{array}$$

Note how these types rely on the newfound flexibility of having zero outputs (for \perp and \neg) and more than one output (for \wp and \ominus). These four types generalize \mathcal{F} , and decompose function types into the more primitive negative disjunction and negation types, analogous to the encoding of functions in classical logic: $A \rightarrow B \approx (\neg A) \wp B$. The full set of dual core \mathcal{D} connectives is given in Figure 5.

B.2 Syntax

The syntax of the dual calculus is given in Figure 6 which is split in two: dual to *terms* (v) which give an answer are *co-terms* (e) which ask a question. Each of the features from the functional language are divided into one of two camps. Variables x , μ -abstractions $\mu \alpha.c$, fixed points $\nu x.v$, objects of co-data types $\lambda \{ \dots \}$, and data structures like $\iota_i v$ are all terms. Dually, co-variables α , $\tilde{\mu}$ -abstractions $\tilde{\mu} x.c$ (analogous to **let** and dual to μ), co-fixed points $\tilde{\nu} \alpha.e$, case analysis of data structures $\tilde{\lambda} \{ \dots \}$ (dual to co-data objects) and co-data observations like $\pi_i e$ (dual to data structures) are all co-terms. A command c is analogous to a jump, and puts together an answer (v) with a question (e). The dual calculus can be seen as inverting elimination forms to the other side of a jump $\langle M \parallel \alpha \rangle$, expanding the role of α . By giving a body to observations themselves, co-patterns q introduce names for *all* sub-components of observations dual to patterns p : for example, the co-pattern of a projection $\pi_i[\alpha:A_i] : A_1 \& A_2$ is perfectly symmetric to the pattern of an injection $\iota_i(x:A_i) : A_1 \oplus A_2$.

In types, there is a dual set of disciplines and connectives. The base kind \star signifies the dual to call-by-need (\star); it shares delayed questions the same way call-by-need shares delayed answers. The negative co-data type constructors \wp and \perp of \mathcal{D} are dual to the positive connectives \otimes and 1 , respectively: they introduce a co-pair $[e, e'] : A \wp B$, which is a pair of co-terms $e : A$ and $e' : B$ accepting inputs of type A and B , and the co-unit $[] : \perp$. Objects of co-data types respond to observations by inverting their *entire* structure and then running a command. For $\&$ this looks like $\lambda \{ \pi_1[\alpha:A].c_1 \mid \pi_2[\beta:B].c_2 \} : A \& B$ and for \wp like $\lambda \{ [x:A, \beta:B].c \} : A \wp B$. In lieu of a non-symmetric function type, we instead have two dual negations: the data type constructor $\ominus : - \rightarrow +$ and the co-data type constructor $\neg : + \rightarrow -$ which introduce the (co-)patterns $\mathbf{cont}(\alpha:A) : \ominus A$ and $\mathbf{throw}[x:A] : \neg A$. These particular

$$\begin{array}{c}
\frac{\Theta, X:k \vdash_{\mathcal{G}} A:l \quad \Theta \vdash_{\mathcal{G}} A:k \rightarrow l \quad \Theta \vdash_{\mathcal{G}} B:k}{\Theta \vdash_{\mathcal{G}} \lambda X:k.A:k \rightarrow l} \quad \frac{(\Theta \vdash_{\mathcal{G}} A:\mathcal{T}).. \quad (\Theta \vdash_{\mathcal{G}} B:\mathcal{R})..}{(x:A.. \vdash_{\mathcal{G}}^{\Theta} \beta:B..) \text{ctx}}}{\Gamma \vdash_{\mathcal{D}}^{\Theta} v:A \mid \Delta \quad \Theta \vdash A:\mathcal{S} \quad \Gamma \mid e:A \vdash_{\mathcal{D}}^{\Theta} \Delta} \text{Cut} \\
\frac{\Gamma \vdash_{\mathcal{D}}^{\Theta} v:A \mid \Delta \quad \Theta \vdash A:\mathcal{S} \quad \Gamma \mid e:A \vdash_{\mathcal{D}}^{\Theta} \Delta}{\langle v \parallel e \rangle : (\Gamma \vdash_{\mathcal{D}}^{\Theta} \Delta)} \\
\frac{\Gamma, x:A \vdash_{\mathcal{D}}^{\Theta} x:A \mid \Delta}{\Gamma \vdash_{\mathcal{D}}^{\Theta} \mu\alpha:A.c:A \mid \Delta} \text{VR} \quad \frac{c:(\Gamma \vdash_{\mathcal{D}}^{\Theta} \alpha:A, \Delta)}{\Gamma \vdash_{\mathcal{D}}^{\Theta} \mu\alpha:A.c:A \mid \Delta} \text{AR} \quad \frac{c:(\Gamma, x:A \vdash_{\mathcal{D}}^{\Theta} \Delta)}{\Gamma \mid \bar{\mu}x:A.c:A \vdash_{\mathcal{D}}^{\Theta} \Delta} \text{AL} \quad \frac{}{\Gamma \vdash_{\mathcal{D}}^{\Theta} \alpha:A \mid \alpha:A, \Delta} \text{VL} \\
\frac{\Gamma, x:A \vdash_{\mathcal{D}}^{\Theta} v:A \mid \Delta \quad \Theta \vdash_{\mathcal{D}} A:-}{\Gamma \vdash_{\mathcal{D}}^{\Theta} \nu x:A.v:A \mid \Delta} \text{RR} \quad \frac{\Gamma \mid e:A \vdash_{\mathcal{D}}^{\Theta} \alpha:A, \Delta \quad \Theta \vdash_{\mathcal{D}} A:+}{\Gamma \mid \bar{\nu}\alpha:A.e:A \vdash_{\mathcal{D}}^{\Theta} \Delta} \text{RL} \\
\frac{\Gamma \mid e:A \vdash_{\mathcal{D}}^{\Theta} \Delta \quad \Theta \vdash_{\mathcal{D}} A=\beta\eta B:\mathcal{S}}{\Gamma \mid e:B \vdash_{\mathcal{D}}^{\Theta} \Delta} \text{TCR} \quad \frac{\Gamma \vdash_{\mathcal{D}}^{\Theta} v:A \mid \Delta \quad \Theta \vdash_{\mathcal{D}} A=\beta\eta B:\mathcal{S}}{\Gamma \vdash_{\mathcal{D}}^{\Theta} v:B \mid \Delta} \text{TCL}
\end{array}$$

Given **data** $F(X:k)..:\mathcal{S}$ where $K_i:(A_{ij}:\mathcal{T}_{ij}^i \vdash^{Y_{ij}:l_{ij}^i} F(X..) \mid B_{ij}:\mathcal{R}_{ij}^i)^i \in \mathcal{G}$, we have the rules:

$$\begin{array}{c}
\frac{}{\Theta \vdash_{\mathcal{G}} F:k.. \rightarrow \mathcal{S}} \\
\frac{(\Theta \vdash_{\mathcal{G}} C_j:l_{ij}^j)^j \quad (\Gamma \mid e_j:B_{ij}[C'/X.., C_j/Y_{ij}^j] \vdash_{\mathcal{G}}^{\Theta} \Delta)^j \quad (\Gamma \vdash_{\mathcal{G}}^{\Theta} v_j:A_{ij}[C'/X.., C_j/Y_{ij}^j] \mid \Delta)^j}{\Gamma \vdash_{\mathcal{G}}^{\Theta} K_i C_j^j \cdot e_j^j \cdot v_j^j : F C'.. \mid \Delta} \text{FR}_i \\
\frac{c_i:(\Gamma, x_{ij}:A_{ij}[C'/X..]^j \vdash_{\mathcal{G}}^{\Theta, Y_{ij}:l_{ij}^j} \alpha_{ij}:B_{ij}[C'/X..]^j, \Delta)^j}{\Gamma \mid \bar{\lambda}\{(K_i Y_{ij}:l_{ij}^j \cdot x_{ij}:A_{ij}^j \cdot x_{ij}:A_{ij}^j \cdot c_i^i)\}: F C'.. \vdash_{\mathcal{G}}^{\Theta} \Delta} \text{FL}
\end{array}$$

Given **codata** $G(X:k)..:\mathcal{S}$ where $O_i:(A_{ij}:\mathcal{T}_{ij}^j \mid G(X..) \vdash^{Y_{ij}:l_{ij}^j} B_{ij}:\mathcal{R}_{ij}^j)^i \in \mathcal{G}$, we have the rules:

$$\begin{array}{c}
\frac{}{\Theta \vdash_{\mathcal{G}} G:k.. \rightarrow \mathcal{S}} \\
\frac{(\Theta \vdash_{\mathcal{G}} C_j:l_{ij}^j)^j \quad (\Gamma \vdash_{\mathcal{G}}^{\Theta} v_j:A_{ij}[C'/X.., C_j/Y_{ij}^j] \mid \Delta)^j \quad (\Gamma \mid e_j:B_{ij}[C'/X.., C_j/Y_{ij}^j] \vdash_{\mathcal{G}}^{\Theta} \Delta)^j}{\Gamma \mid O_i C_j^j \cdot v_j^j \cdot e_j^j : F C'.. \vdash_{\mathcal{G}}^{\Theta} \Delta} \text{GL}_i \\
\frac{c_i:(\Gamma, x_{ij}:A_{ij}[C'/X..]^j \vdash_{\mathcal{G}}^{\Theta, Y_{ij}:l_{ij}^j} \alpha_{ij}:B_{ij}[C'/X..]^j, \Delta)^j}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \lambda\{[O_i Y_{ij}:l_{ij}^j \cdot x_{ij}:A_{ij}^j \cdot \alpha_{ij}:B_{ij}^j] \cdot c_i^i\}: G C'.. \mid \Delta} \text{GR}
\end{array}$$

■ **Figure 7** Type system for the dual calculus.

forms of negation are chosen because they are *involutive* up to isomorphism (as defined next in Appendix C); their two compositions are identities on types: $\ominus(\neg A) \approx A$ and $\neg(\ominus B) \approx B$ for any $A : +$ and $B : -$. Function types can be faithfully represented as $A \rightarrow B \approx (\neg A) \wp B$.

B.3 Type system

The type system of \mathcal{D} is given in Figure 7. One change from the functional calculus' type system is the use of the single-level typing judgement $v : A$ instead of the two-level $M : A : \mathcal{S}$. This is possible because of the sequent calculus' *sub-formula property* – *Cut* is the only inference rule that introduces arbitrary new types in the premises. By just checking that the type of a *Cut* makes sense in the current environment, well-formedness can be separated from typing: if the conclusion of a derivation is well-formed (*i.e.*, $(\Gamma \vdash_{\mathcal{D}}^{\Theta} \Delta) \text{ctx}$), then every judgement in the derivation is too. There is also a typing judgement for co-terms; $\Gamma \mid e : A \vdash_{\mathcal{D}}^{\Theta} \Delta$ means that e works with a term of type A in the environments Θ, Γ, Δ .

$$\begin{aligned}
V_+ &::= x \mid KB..E..V.. \mid \lambda\{q.c..\} & V_* &::= V_+ \mid \mu\alpha.H[\langle V_* \parallel \alpha \rangle] & V_- &::= v & V_* &::= V_+ \\
E_- &::= \alpha \mid \mathbf{O}B..V..E.. \mid \tilde{\lambda}\{p.c..\} & E_* &::= E_- \mid \tilde{\mu}\mathbf{x}.H[\langle x \parallel E_* \rangle] & E_+ &::= e & E_* &::= E_- \\
H &::= \square \mid \langle v \parallel \tilde{\mu}\mathbf{x}.A:*.H \rangle \mid \langle \mu\alpha:A:*.H \parallel e \rangle
\end{aligned}$$

$$\begin{aligned}
(\beta_\mu) \quad & \langle \mu\alpha.c \parallel E \rangle \sim c[E/\alpha] & (\eta_{\tilde{\mu}}) \quad & \tilde{\mu}\mathbf{x}.A.\langle x \parallel e \rangle \sim e & (\nu) \quad & \nu\mathbf{x}.v \sim v[\nu\mathbf{x}.v/\mathbf{x}] \\
(\beta_{\tilde{\mu}}) \quad & \langle V \parallel \tilde{\mu}\mathbf{x}.c \rangle \sim c[V/\mathbf{x}] & (\eta_\mu) \quad & \mu\alpha:A.\langle v \parallel \alpha \rangle \sim v & (\tilde{\nu}) \quad & \tilde{\nu}\alpha.e \sim e[\nu\alpha.e/\alpha] \\
(\beta_{\mathbf{O}}) \quad & \langle \lambda\{.. \mid [\mathbf{O}Y..x..\alpha..].c \mid ..\} \parallel \mathbf{O}B..v..e..\rangle \sim \langle v..\parallel \tilde{\mu}\mathbf{x}...\langle \mu\alpha...c[B/Y..] \parallel e..\rangle \rangle \\
(\beta_{\mathbf{K}}) \quad & \langle \mathbf{O}B..e..v..\parallel \tilde{\lambda}\{.. \mid (\mathbf{O}Y..x..\alpha..).c \mid ..\} \rangle \sim \langle \mu\alpha...\langle v..\parallel \tilde{\mu}\mathbf{x}...c[B/Y..] \parallel e..\rangle \rangle \\
(\eta_{\mathbf{G}}) \quad & \lambda\{q_i.\langle x \parallel q_i \rangle^i.\} \sim x & (\chi^*) \quad & \langle \mu\alpha:A:*. \langle v \parallel \tilde{\mu}y:B:*.c \rangle \parallel e \rangle \sim \langle v \parallel \tilde{\mu}y:B:*. \langle \mu\alpha:A:*.c \parallel e \rangle \rangle \\
(\eta_{\mathbf{F}}) \quad & \tilde{\lambda}\{p_i.\langle p_i \parallel \alpha \rangle^i.\} \sim \alpha & (\chi^*) \quad & \langle v \parallel \tilde{\mu}y:B:*. \langle \mu\alpha:A:*.c \parallel e \rangle \rangle \sim \langle \mu\alpha:A:*. \langle v \parallel \tilde{\mu}y:B:*.c \rangle \parallel e \rangle
\end{aligned}$$

$$\frac{c_i : (\Gamma \vdash_{\mathcal{D}}^{\ominus} \Delta) \quad c_1 \sim c_2}{c_1 = c_2 : (\Gamma \vdash_{\mathcal{D}}^{\ominus} \Delta)} \quad \frac{\Gamma \vdash_{\mathcal{D}}^{\ominus} v_i : A \mid \Delta \quad v_1 \sim v_2}{\Gamma \vdash_{\mathcal{D}}^{\ominus} v_1 = v_2 : A \mid \Delta} \quad \frac{\Gamma \vdash_{\mathcal{D}}^{\ominus} e_i : A \mid \Delta \quad e_1 \sim e_2}{\Gamma \mid e_1 = e_2 : A \vdash_{\mathcal{D}}^{\ominus} \Delta}$$

plus compatibility, reflexivity, symmetry, transitivity

■ **Figure 8** Equational theory for the dual calculus.

B.4 Equational theory

Lastly, we have the equational theory in Figure 8. The dualities of evaluation – between variable and co-variable bindings, data and co-data, values (answers) and evaluation contexts (questions) – are more readily apparent than \mathcal{F} . In particular, the notion of substitution discipline for \mathcal{S} is now fully dual as in [7]: a subset of terms (*values* $V_{\mathcal{S}}$) and a subset of co-terms (*co-values* $E_{\mathcal{S}}$) which are substitutable, giving the known dualities between call-by-value (+) and -name (−) [5] and \star and \star [3]. The χ axioms reassociate variable and co-variable bindings, and the important cases are for \star (corresponding to χ^* of **lets**) and \star . Also note the lack of commuting conversions κ ; these follow from the μ axiom.

C Encoding fully dual (co-)data types into \mathcal{D}

Now let's look at the fully dual version of the functional encoding from Section 5. Thanks to the generic notion of shifts, the encoding of dual (co-)data into the core \mathcal{D} connectives is similar to the functional encoding, except that in place of the function type $A \rightarrow B$ we use the classical representation $(\ominus A) \wp B$. For the generic (co-)data declarations in Figure 7, we have the following definition:

$$\begin{aligned}
[[\mathbf{F}]]_{\mathcal{G}}^{\mathcal{D}} &\triangleq \lambda\mathbf{X}...s\uparrow((\exists\mathbf{Y}_{ij}..i.((\ominus(\uparrow\mathcal{R}_{ij} B_{ij})) \otimes i.((\downarrow\mathcal{T}_{ij} A_{ij}) \otimes i.1))) \oplus i.0) \\
[[\mathbf{G}]]_{\mathcal{G}}^{\mathcal{D}} &\triangleq \lambda\mathbf{X}...s\downarrow((\forall\mathbf{Y}_{ij}..i.((\downarrow\mathcal{T}_{ij} A_{ij})) \wp i.((\uparrow\mathcal{R}_{ij} B_{ij}) \wp i.\perp))) \& i.\top
\end{aligned}$$

The encoding of multi-output data types places a \ominus -negates every additional output of a constructor, and the encoding of multi-output co-data is now exactly dual to the data encoding. The encodings of (co-)patterns, (co-)pattern-matching objects, and (co-)data

structures follow the above type encoding like so:

$$\begin{aligned}
\llbracket K_i Y.. \alpha.. x.. \rrbracket_{\mathcal{G}}^{\mathcal{D}} &\triangleq \text{val}_{\mathcal{S}} (\iota_2^i (\iota_1 (\text{pack } Y.. (\text{cont}[\text{eval}_{\mathcal{R}} \alpha], .. (\text{box}_{\mathcal{T}} x, .. ()))))) \\
\llbracket O_i Y.. x.. \alpha.. \rrbracket_{\mathcal{G}}^{\mathcal{D}} &\triangleq \text{enter}_{\mathcal{S}} [\pi_2^i [\pi_1 [\text{spec } Y.. [\text{throw}[\text{box}_{\mathcal{T}} x], .. [\text{eval}_{\mathcal{R}} \alpha, .. (]]]]]] \\
\llbracket \lambda\{q_i.c_i^i.\} \rrbracket_{\mathcal{G}}^{\mathcal{D}} &\triangleq \lambda\{\llbracket q_i \rrbracket_{\mathcal{G}}^{\mathcal{D}}.\llbracket c_i \rrbracket_{\mathcal{G}}^{\mathcal{D}.i}\} \\
\llbracket \tilde{\lambda}\{p_i.c_i^i.\} \rrbracket_{\mathcal{G}}^{\mathcal{D}} &\triangleq \tilde{\lambda}\{\llbracket p_i \rrbracket_{\mathcal{G}}^{\mathcal{D}}.\llbracket c_i \rrbracket_{\mathcal{G}}^{\mathcal{D}.i}\} \\
\llbracket p[C/Y.., e/\alpha.., v/x..] \rrbracket_{\mathcal{G}}^{\mathcal{D}} &= \llbracket p \rrbracket_{\mathcal{G}}^{\mathcal{D}} [\llbracket C \rrbracket_{\mathcal{G}}^{\mathcal{D}}/Y.., \llbracket e \rrbracket_{\mathcal{G}}^{\mathcal{D}}/\alpha.., \llbracket v \rrbracket_{\mathcal{G}}^{\mathcal{D}}/x..] \\
\llbracket q[C/Y.., v/x..], e/\alpha.. \rrbracket_{\mathcal{G}}^{\mathcal{D}} &= \llbracket q \rrbracket_{\mathcal{G}}^{\mathcal{D}} [\llbracket C \rrbracket_{\mathcal{G}}^{\mathcal{D}}/Y.., \llbracket v \rrbracket_{\mathcal{G}}^{\mathcal{D}}/x.., \llbracket e \rrbracket_{\mathcal{G}}^{\mathcal{D}}/\alpha..]
\end{aligned}$$

We also have an analogous notion of type isomorphism. The case for higher kinds is the same, and base isomorphism $\Theta \vDash_{\mathcal{G}} A \approx B : \mathcal{S}$ is witnessed by a pair of inverse commands $c : (x : A \vdash_{\mathcal{G}}^{\Theta} \beta : B)$ and $c' : (y : B \vdash_{\mathcal{G}}^{\Theta} \alpha : A)$ such that both compositions are identities:

$$\langle \mu\beta : B.c \parallel \tilde{\mu}y : B.c' \rangle = \langle x \parallel \alpha \rangle : (x : A \vdash_{\mathcal{G}}^{\Theta} \alpha : A) \quad \langle \mu\alpha : A.c' \parallel \tilde{\mu}x : A.c \rangle = \langle y \parallel \beta \rangle : (y : B \vdash_{\mathcal{G}}^{\Theta} \beta : B)$$

Using type isomorphisms in \mathcal{D} , the analogous local and global encodings are sound for fully dual data and co-data types utilizing any combination of $+$, $-$, \star , and \star evaluation.

► **Theorem 7.** *For all $\vdash_{\mathcal{G}}$ extending \mathcal{D} and $\Theta \vDash_{\mathcal{G}} A : k$, $\Theta \vDash_{\mathcal{G}} A \approx \llbracket A \rrbracket_{\mathcal{G}}^{\mathcal{D}} : k$.*

► **Theorem 8.** *For all $\vdash_{\mathcal{G}}$ extending \mathcal{D} , (co-)terms of type A are in equational correspondence with (co-)terms of type $\llbracket A \rrbracket_{\mathcal{G}}^{\mathcal{D}}$, respectively.*


► **Theorem 9.** *If $\vdash_{\mathcal{G}}$ extends \mathcal{D} and $c = c' : (\Gamma \vdash_{\mathcal{G}}^{\Theta} \Delta)$ then $\llbracket c \rrbracket_{\mathcal{G}}^{\mathcal{D}} = \llbracket c' \rrbracket_{\mathcal{G}}^{\mathcal{D}} : (\llbracket \Gamma \rrbracket_{\mathcal{G}}^{\mathcal{D}} \vdash_{\mathcal{F}}^{\Theta} \llbracket \Delta \rrbracket_{\mathcal{G}}^{\mathcal{D}})$.*

Expressivity Within Second-Order Transitive-Closure Logic

Flavio Ferrarotti

Software Competence Center Hagenberg, Hagenberg, Austria


flavio.ferrarotti@scch.at

 <https://orcid.org/0000-0003-2278-8233>

Jan Van den Bussche

Hasselt University, Hasselt, Belgium


jan.vandenbussche@uhasselt.be

 <https://orcid.org/0000-0003-0072-3252>

Jonni Virtema

Hasselt University, Hasselt, Belgium

jonni.virtema@uhasselt.be

 <https://orcid.org/0000-0002-1582-3718>

Abstract

Second-order transitive-closure logic, $SO(TC)$, is an expressive declarative language that captures the complexity class PSPACE. Already its monadic fragment, $MSO(TC)$, allows the expression of various NP-hard and even PSPACE-hard problems in a natural and elegant manner. As $SO(TC)$ offers an attractive framework for expressing properties in terms of declaratively specified computations, it is interesting to understand the expressivity of different features of the language. This paper focuses on the fragment $MSO(TC)$, as well on the purely existential fragment $SO(2TC)(\exists)$; in $2TC$, the TC operator binds only tuples of relation variables. We establish that, with respect to expressive power, $SO(2TC)(\exists)$ collapses to existential first-order logic. In addition we study the relationship of $MSO(TC)$ to an extension of $MSO(TC)$ with counting features ($CMSO(TC)$) as well as to order-invariant MSO . We show that the expressive powers of $CMSO(TC)$ and $MSO(TC)$ coincide. Moreover we establish that, over unary vocabularies, $MSO(TC)$ strictly subsumes order-invariant MSO .

2012 ACM Subject Classification Theory of computation \rightarrow Finite Model Theory

Keywords and phrases Expressive power, Higher order logics, Descriptive complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.22

Funding The research reported in this paper results from the joint project *Higher-Order Logics and Structures* supported by the Austrian Science Fund (FWF: [I2420-N31]) and the Research Foundation Flanders (FWO: [G0G6516N]).

1 Introduction

Second-order transitive-closure logic, $SO(TC)$, is an expressive declarative language that captures the complexity class PSPACE [21]. It extends second-order logic with a transitive closure operator over relations of relations, i.e., over super relations among relational structures. The super relations are defined by means of second-order logic formulae with free relation variables. Already its monadic fragment, $MSO(TC)$, allows the expression of NP-complete problems in a natural and elegant manner. Consider, for instance, the well known Hamiltonian cycle query over the standard vocabulary of graphs, which is not expressible in monadic second-order logic [13].



© Flavio Ferrarotti, Jan Van den Bussche, and Jonni Virtema;
licensed under Creative Commons License CC-BY

27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 22; pp. 22:1–22:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

► **Example 1.** A graph $G = (V, E)$ has a Hamiltonian cycle if the following holds:

- a. There is a relation \mathcal{R} such that $(Z, z, Z', z') \in \mathcal{R}$ iff $Z' = Z \cup \{z'\}$, $z' \notin Z$, and $(z, z') \in E$.
 - b. The tuple $(\{x\}, x, V, y)$ is in the transitive closure of \mathcal{R} , for some $x, y \in V$ s.t. $(y, x) \in E$.
- In the language of MSO(TC) this can be written as follows:

$$\exists XYxy(X(x) \wedge \forall z(z \neq x \rightarrow \neg X(z)) \wedge \forall z(Y(z)) \wedge E(y, x) \wedge [\text{TC}_{Z,z,Z',z'}\varphi](X, x, Y, y)),$$

where $\varphi := \neg Z(z') \wedge \forall x(Z'(x) \leftrightarrow (Z(x) \vee z' = x)) \wedge E(z, z')$.

Even some well-known PSPACE-complete problems such as deciding whether a given quantified Boolean formula QBF is valid [27] can be expressed in MSO(TC) (see Section 3).

In general, SO(TC) offers an attractive framework for expressing properties in terms of declaratively specified computations at a high level of abstraction. There are many examples of graph computation problems that involve complex conditions such as graph colouring [4], topological subgraph discovery [19], recognition of hypercube graphs [18], and many others (see [9, 16, 17]). Such graph algorithms are difficult to specify, even by means of rigorous methods such as Abstract State Machines (ASMs) [10], B [2] or Event-B [3], because the algorithms require the definition of characterising conditions for particular subgraphs that lead to expressions beyond first-order logic. Therefore, for the sake of easily comprehensible and at the same time fully formal high-level specifications, it is reasonable to explore languages such as SO(TC). Let us see an example that further supports these observations.

► **Example 2.** Self-similarity of complex networks [37] (aka scale invariance) has practical applications in diverse areas such as the world-wide web [14], social networks [20], and biological networks [32]. Given a network represented as a finite graph G , it is relevant to determine whether G can be built starting from some graph pattern G_p by recursively replacing nodes in the pattern by new, “smaller scale”, copies of G_b . If this holds, then we say that G is self-similar.

Formally, a graph G is *self-similar* w.r.t. a graph pattern G_p of size k , if there is a sequence of graphs G_0, G_1, \dots, G_n such that $G_0 = G_p$, $G_n = G$ and, for every pair (G_i, G_{i+1}) of consecutive graphs in the sequence, there is a partition $\{P_1, \dots, P_k\}$ of the set of nodes of G_{i+1} which satisfies the following:

- a. For every $j = 1, \dots, k$, the sub-graph induced by P_j in G_{i+1} is isomorphic to G_i .
- b. There is a graph G_t isomorphic to G_p with set of nodes $V_t = \{a_1, \dots, a_k\}$ for some $a_1 \in P_1, \dots, a_k \in P_k$ and set of edges

$$E_t = \{(a_i, a_j) \mid \text{there is an edge } (x, y) \text{ of } G_{i+1} \text{ such that } P_i(x) \text{ and } P_j(y)\}.$$

- c. For very $1 \leq i < j \leq k$, the closed neighborhoods $N_{G_{i+1}}[P_i]$ and $N_{G_{i+1}}[P_j]$ of P_i and P_j in G_{i+1} , respectively, are isomorphic.

It is straightforward to write this definition of self-similarity in SO(TC), for we can clearly write a second-order logic formula which defines such a super relation \mathcal{R} on graphs and then simply check whether the pair of graphs (G, G_p) is in the transitive closure of \mathcal{R} .

Highly expressive query languages are gaining relevance in areas such as knowledge representation (KR), rigorous methods and provers. There are several examples of highly expressive query languages related to applications in KR. See for instance the monadically defined queries in [36], the Monadic Disjunctive SNP queries in [5] or the guarded queries in [11]. See also [33] where a query language with transitive closure for graph databases is considered. All of them can be considered fragments of Datalog. Regarding rigorous methods, the TLA⁺ language [28] is able to deal with higher-order formulations, and tools such as the

TLA⁺ Proof System¹ and the TLA⁺ Model-Checker (TLC)² can handle them (provided a finite universe of values for TLC). Provers such as Coq³ and Isabelle⁴ can already handle some high-order expression. Moreover, the success with solvers for the Boolean satisfiability problem (SAT) has encouraged researchers to target larger classes of problems, including PSPACE-complete problems, such as satisfiability of Quantified Boolean formulas (QBF). Note the competitive evaluations of QBF solvers (QBFEVAL) held in 2016 and 2017 and recent publications on QBF solvers such as [8, 31, 22] among several others.

We thus think it is timely to study which features of highly expressive query languages affect their expressive power. In this sense, SO(TC) provides a good theoretical base since, apart from been a highly expressive query language (recall that it captures PSPACE), it enables natural and precise high-level definitions of complex practical problems, mainly due to its ability to express properties in terms of declaratively specified computations. While second-order logic extended with the standard partial fixed-point operator, as well as first-order logic closed under taking partial fixed-points and under an operator for non-deterministic choice, also capture the class of PSPACE queries over arbitrary finite structure [34], relevant computation problems such as that in Example 2 are clearly more difficult to specify in these logics. The same applies to the extension of first-order logic with the partial fixed-point operator, which is furthermore subsumed by SO(TC) since it captures PSPACE only on the class of ordered finite structures [1]. Note that SO(TC) coupled with hereditary finite sets and set terms, could be considered as a kind of declarative version of Blass, Gurevich, and Shelah (BGS) model of abstract state machine [7], which is a powerful language in which all computable queries to relational databases can be expressed [6].

Our results can be summarized as follows.

1. We investigate to what extent universal quantification and negation are important to the expressive power of SO(TC). Specifically, we consider the case where TC-operators are applied only to second-order variables. Of course, a second-order variable can simulate a first-order variable, since we can express already in first-order logic (FO) that a set is a singleton. This, however, requires universal quantification.

We define a “purely existential” fragment of SO(TC), SO(2TC)(\exists), as the fragment without universal quantifiers and in which TC-operators occur only positively and bind only tuples of relation variables. We show that the expressive power of this fragment collapses to that of existential FO.

For SO alone, this collapse is rather obvious and was already remarked by Rosen in the introduction of his paper [35]. Our result generalizes this collapse to include TC operators, where it is no longer obvious.

2. We investigate the expressive power of the monadic fragment, MSO(TC). On strings, this logic is equivalent to the complexity class NLIN. Already on unordered structures, however, we show that MSO(TC) can express counting terms and numeric predicates in NLOGSPACE. In particular, MSO(TC) can express queries not expressible in the fixpoint logic FO(LFP). We also discuss the fascinating open question whether the converse holds as well.
3. We compare the expressive power of MSO(TC) to that of order-invariant MSO. Specifically, we show that MSO(TC) can express queries not expressible in order-invariant MSO; over monadic vocabularies, we show that order-invariant MSO is subsumed by MSO(TC). Again, what happens over higher-arity relations is an interesting open question.

¹ <https://tla.msr-inria.inria.fr/tlaps>

² <https://lampport.azurewebsites.net/tla/tlc.html>

³ <https://coq.inria.fr/>

⁴ <https://isabelle.in.tum.de/>

This paper is organized as follows. In Section 2 definitions and basic notions related to SO(TC) are given. In Section 3 the complexity of model checking is studied. Section 4 is dedicated to establishing the collapse of SO(2TC)(\exists) to existential first-order logic. Sections 5 and 6 concentrate on the relationships between MSO(TC) and the counting extension CMSO(TC) and order-invariant MSO, respectively. We conclude with a discussion of open questions in Section 7.

2 Preliminaries

We assume that the reader is familiar with finite model theory, see e.g., [15] for a good reference. For a tuple \vec{a} of elements, we denote by $\vec{a}[i]$ the i th element of the tuple. We recall from the literature, the syntax and semantics of first-order (FO) and second-order (SO) logic, as well as their extensions with the transitive closure operator (TC). We assume a sufficient supply of *first-order* and *second-order variables*. The natural number $\text{ar}(R) \in \mathbb{N}$, is the *arity* of the second-order variable X . By *variable*, we mean either a first-order or second-order variable. Variables χ and χ' have the same *sort* if either both χ and χ' are first-order variables, or both are second-order variables of the same arity. Tuples $\vec{\chi}$ and $\vec{\chi}'$ of variables have the same *sort*, if the lengths of $\vec{\chi}$ and $\vec{\chi}'$ are the same and, for each i , the sort of $\vec{\chi}[i]$ is the same as the sort of $\vec{\chi}'[i]$.

► **Definition 3.** The formulas of SO(TC) are defined by the following grammar:

$$\varphi ::= x = y \mid X(x_1, \dots, x_k) \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \exists x\varphi \mid \exists Y\varphi \mid [\text{TC}_{\vec{X}, \vec{X}'}\varphi](\vec{Y}, \vec{Y}'),$$

where X and Y are second-order variables, $k = \text{ar}(X)$, x, y, x_1, \dots, x_k are first-order variables, \vec{X} and \vec{X}' are disjoint tuples of variables of the same sort, and \vec{Y} and \vec{Y}' are also tuples of variables of that same sort (but not necessarily disjoint).

The set of free variables of a formula φ , denoted by $\text{FV}(\varphi)$ is defined as usual. For the TC operator, we define

$$\text{FV}([\text{TC}_{\vec{X}, \vec{X}'}\varphi](\vec{Y}, \vec{Y}')) := (\text{FV}(\varphi) - (\vec{X} \cup \vec{X}')) \cup \vec{Y} \cup \vec{Y}'.$$

Above in the right side, in order to avoid cumbersome notation, we use \vec{X} , \vec{X}' , \vec{Y} and \vec{Y}' to denote the sets of variables occurring in the tuples.

A *vocabulary* is a finite set of variables. A (finite) *structure* \mathfrak{A} over a vocabulary τ is a pair (A, I) , where A is a finite nonempty set called the *domain* of \mathfrak{A} , and I is an *interpretation* of τ on A . By this we mean that whenever $x \in \tau$ is a first-order variable, then $I(x) \in A$, and whenever $X \in \tau$ is a second-order variable of arity m , then $I(X) \subseteq A^m$. In this article, structures are always finite. We denote $I(X)$ also by $X^{\mathfrak{A}}$. For a variable X and a suitable value R for that variable, $\mathfrak{A}[R/X]$ denotes the structure over $\tau \cup \{X\}$ equal to \mathfrak{A} except that X is mapped to R . We extend the notation also to tuples of variables and values, $\mathfrak{A}[\vec{R}/\vec{X}]$, in the obvious manner. We say that a vocabulary τ is *appropriate* for a formula φ if $\text{FV}(\varphi) \subseteq \tau$.

► **Definition 4.** Let \mathfrak{A} be a structure over τ and φ an SO(TC)-formula such that τ is appropriate for φ . The satisfaction of φ by \mathfrak{A} , denoted by $\mathfrak{A} \models \varphi$, is defined as follows. We only give the cases for second-order quantifiers and transitive closure operator; the remaining cases are defined as usual.

- For second-order variable X : $\mathfrak{A} \models \exists X\varphi$ iff $\mathfrak{A}[R/X] \models \varphi$, for some $R \subseteq A^{\text{ar}(X)}$.

- For the case of the TC-operator, consider a formula ψ of the form $[\text{TC}_{\vec{X}, \vec{X}'}\varphi](\vec{Y}, \vec{Y}')$ and let $\mathfrak{A} = (A, I)$. Define $\mathcal{J}_{\vec{X}}$ to be the following set

$$\{J(\vec{X}) \mid J \text{ is an interpretation of } \vec{X} \text{ on } A\} = \{J(\vec{X}') \mid J \text{ is an interpretation of } \vec{X}' \text{ on } A\}$$

and consider the binary relation \mathcal{B} on $\mathcal{J}_{\vec{X}}$ defined as follows:

$$\mathcal{B} := \{(\vec{R}, \vec{R}') \in \mathcal{J}_{\vec{X}} \times \mathcal{J}_{\vec{X}'} \mid \mathfrak{A}[\vec{R}/\vec{X}, \vec{R}'/\vec{X}'] \models \varphi\}.$$

We set $\mathfrak{A} \models \psi$ to hold if $(I(\vec{Y}), I(\vec{Y}'))$ belongs to the transitive closure of \mathcal{B} . Recall that, for a binary relation \mathcal{B} on any set \mathcal{J} , the transitive closure of \mathcal{B} is defined by

$$\text{TC}(\mathcal{B}) := \{(a, b) \in \mathcal{J} \times \mathcal{J} \mid \exists n > 0 \text{ and } e_0, \dots, e_n \in \mathcal{J} \\ \text{such that } a = e_0, b = e_n, \text{ and } (e_i, e_{i+1}) \in \mathcal{B} \text{ for all } i < n\}.$$

By TC^m we denote the variant of TC in which the quantification of n above is restricted to natural numbers $\leq m$. That is, $\text{TC}^m(\mathcal{B})$ consists of pairs (\vec{a}, \vec{b}) such that \vec{b} is reachable from \vec{a} by \mathcal{B} in at most m steps. Moreover, by 2TC and 2TC^m we denote the syntactic restrictions of TC and TC^m of the form

$$[\text{TC}_{\vec{X}, \vec{X}'}\varphi](\vec{Y}, \vec{Y}') \text{ and } [\text{TC}_{\vec{X}, \vec{X}'}^m\varphi](\vec{Y}, \vec{Y}'),$$

where $\vec{X}, \vec{X}', \vec{Y}, \vec{Y}'$ are tuples of second-order variables (i.e. without first-order variables). The logic $\text{SO}(2\text{TC})$ then denotes the extension of second-order logic with 2TC -operator. Analogously, by $\text{FO}(1\text{TC})$, we denote the extension of first-order logic with applications of such transitive-closure operators that bind only first-order variables.⁵

3 Complexity of MSO(TC)

The descriptive complexity of different logics with the transitive closure operator has been thoroughly studied by Immerman. Let $\text{SO}(\text{arity } k)(\text{TC})$ denote the fragment of $\text{SO}(\text{TC})$ in which second-order variables are all of arity $\leq k$.

► **Theorem 5** ([23, 24]).

- *On finite ordered structures, first-order transitive-closure logic $\text{FO}(1\text{TC})$ captures non-deterministic logarithmic space NLOGSPACE .*
 - *On strings (word structures), $\text{SO}(\text{arity } k)(\text{TC})$ captures the complexity class $\text{NSPACE}(n^k)$.*
- See also the discussion in the conclusion section.

By the above theorem, $\text{MSO}(\text{TC})$ captures nondeterministic linear space NLIN over strings. Deciding whether a given *quantified Boolean formula* is valid (QBF) is a well-known PSPACE -complete problem [27]. Observe that there are PSPACE -complete problems already in NLIN ; in fact QBF is such a problem. Thus, we can conclude the following. The inclusion in PSPACE is clear.

► **Proposition 6.** *Data complexity of model checking of $\text{MSO}(\text{TC})$ is PSPACE -complete.*

We next turn to combined complexity of model checking. By the above proposition, this is at least PSPACE -hard. However, the straightforward algorithm for model checking $\text{MSO}(\text{TC})$ clearly has polynomial-space combined complexity. We thus conclude:

⁵ In the literature $\text{FO}(1\text{TC})$ is often denoted by $\text{FO}(\text{TC})$.

► **Proposition 7.** *Combined complexity of model checking of MSO(TC) is PSPACE-complete.*

For combined complexity, we can actually sharpen the PSPACE-hardness; already a very simple fragment of MSO(TC) is PSPACE-complete.

Specifically, we give a reduction from the *corridor tiling* problem, which is a well-known PSPACE-complete problem. Instance of the corridor tiling problem is a tuple $P = (T, H, V, \vec{b}, \vec{t}, n)$, where $n \in \mathbb{N}$ is a positive natural number, $T = \{1, \dots, k\}$, for some $k \in \mathbb{N}$, is a finite set of *tiles*, $H, V \subseteq T \times T$ are *horizontal* and *vertical constraints*, and \vec{b}, \vec{t} are n -tuples of tiles from T . A *corridor tiling for P* is a function $f : \{1, \dots, n\} \times \{1, \dots, m\} \rightarrow T$, for some $m \in \mathbb{N}$, such that

- $(f(1, 1), \dots, f(n, 1)) = \vec{b}$ and $(f(1, m), \dots, f(n, m)) = \vec{t}$,
- $(f(i, j), f(i + i, j)) \in H$, for $i < n$ and $j \leq m$,
- $(f(i, j), f(i, j + 1)) \in V$, for $i \leq n$ and $j < m$.

The *corridor tiling problem* is the following PSPACE-complete decision problem [12]:

Input: An instance $P = (T, H, V, \vec{b}, \vec{t}, n)$ of the corridor tiling problem.

Output: Does there exist a corridor tiling for P ?

Let *monadic 2TC[\forall FO]* denote the fragment of MSO(2TC) of the form $[\text{TC}_{\vec{X}, \vec{X}'} \varphi](\vec{Y}, \vec{Y}')$, where φ is a formula of universal first-order logic (i.e., φ is of the form $\forall \vec{x} \psi$, where ψ is a quantifier-free formula of first-order logic).

► **Theorem 8.** *Combined complexity of model checking for monadic 2TC[\forall FO] is PSPACE-complete.*

Proof. Inclusion to PSPACE follows from the corresponding result for MSO(TC). In order to prove hardness, we give a reduction from corridor tiling. Let $P = (T, H, V, \vec{b}, \vec{t}, n)$ be an instance of the corridor tiling problem and set $k := |T|$. Let $\tau = \{s, X_1, \dots, X_k, Y_1, \dots, Y_k\}$ be a vocabulary, where s is a binary second-order variable and $X_1, \dots, X_k, Y_1, \dots, Y_k$ are monadic second-order variables. Let \mathfrak{A}_P denote the structure over τ such that $A = \{1, \dots, n\}$, $I(s)$ is the canonical successor relation on A , and, for each $i \leq k$, $I(X_i) = \{j \in A \mid \vec{b}[j] = i\}$ and $I(Y_i) = \{j \in A \mid \vec{t}[j] = i\}$. Define

$$\begin{aligned} \varphi_H &:= \forall xy (s(x, y) \rightarrow \bigvee_{(i,j) \in H} Z'_i(x) \wedge Z'_j(y)), & \varphi_V &:= \forall x \bigvee_{(i,j) \in V} Z_i(x) \wedge Z'_j(x) \\ \varphi_T &:= \forall x \bigvee_{i \in T} (Z'_i(x) \wedge \bigwedge_{j \in T, i \neq j} \neg Z'_j(x)), \end{aligned}$$

where \vec{Z} and \vec{Z}' are k -tuples of distinct monadic second-order variables not in τ . We then define $\varphi_P := \text{TC}_{\vec{Z}, \vec{Z}'}[\varphi_T \wedge \varphi_H \wedge \varphi_V](\vec{X}, \vec{Y})$. We claim that $\mathfrak{A}_P \models \varphi_P$ if and only if there exists a corridor tiling for P , from which the claim follows. ◀

4 Existential positive SO(2TC) collapses to EFO

Let $\text{SO}(2\text{TC})[\exists]$ denote the syntactic fragment of $\text{SO}(2\text{TC})$ in which existential quantifiers and the TC-operator occur only positively, that is, in scope of even number of negations. In this section, we show that the expressive power of $\text{SO}(2\text{TC})[\exists]$ collapses to that of existential first-order logic $\exists\text{FO}$. In this section, TC-operators are applied only to tuples of second-order variables. As already discussed in the introduction, this restriction is vital: the formula $[\text{TC}_{x, x'} R(x, x') \vee x = x'](y, y')$ expresses reachability in directed graphs, which is not definable even in the full first-order logic.

To facilitate our proofs we start by introducing some helpful terminology.

► **Definition 9.** Let \vec{a} and \vec{b} be tuples of the same length and I a set of natural numbers. The *difference* $\text{diff}(\vec{a}, \vec{b})$ of the tuples \vec{a} and \vec{b} is defined as follows

$$\text{diff}(\vec{a}, \vec{b}) := \{i \mid \vec{a}[i] \neq \vec{b}[i]\}.$$

The *similarity* $\text{sim}(\vec{a}, \vec{b})$ of tuples \vec{a} and \vec{b} is defined as follows

$$\text{sim}(\vec{a}, \vec{b}) := \{i \mid \vec{a}[i] = \vec{b}[i]\}.$$

We say that the tuples \vec{a} and \vec{b} are *pairwise compatible* if the sets $\{\vec{a}[i] \mid i \in \text{diff}(\vec{a}, \vec{b})\}$ and $\{\vec{b}[i] \mid i \in \text{diff}(\vec{a}, \vec{b})\}$ are disjoint. The tuples \vec{a} and \vec{b} are *pairwise compatible outside* I if $\{\vec{a}[i] \mid i \in \text{diff}(\vec{a}, \vec{b}), i \notin I\}$ and $\{\vec{b}[i] \mid i \in \text{diff}(\vec{a}, \vec{b}), i \notin I\}$ are disjoint. The tuples \vec{a} and \vec{b} are *pairwise I -compatible* if \vec{a} and \vec{b} are pairwise compatible and $\text{sim}(\vec{a}, \vec{b}) = I$.

► **Definition 10.** Let $\sigma \subseteq \tau$ be vocabularies, \mathfrak{A} a τ -structure, and \vec{a} a tuple of elements of A . The (quantifier-free) σ -type of \vec{a} in \mathfrak{A} is the set of those quantifier free FO(σ)-formulae $\varphi(\vec{x})$ such that $\mathfrak{A}[\vec{a}/\vec{x}] \models \varphi$.

The following lemma establishes that 2TC-operators that are applied to \exists FO-formulas can be equivalently expressed by the finite 2TC^m-operator.

► **Lemma 11.** *Every formula φ of the form $[\text{TC}_{\vec{X}, \vec{X}'}\theta](\vec{Y}, \vec{Y}')$, where $\theta \in \exists\text{FO}$ and \vec{X}, \vec{X}' , \vec{Y}, \vec{Y}' are tuples of second-order variables, is equivalent with the formula $[\text{TC}_{\vec{X}, \vec{X}'}^k\theta](\vec{Y}, \vec{Y}')$, for some $k \in \mathbb{N}$.*

Proof. Let $\theta = \exists x_1 \dots \exists x_n \psi$, where ψ is quantifier-free, and let τ denote the vocabulary of φ . We will show that for large enough k and for all τ -structures \mathfrak{A}

$$\mathfrak{A} \models [\text{TC}_{\vec{X}, \vec{X}'}\theta](\vec{Y}, \vec{Y}') \text{ iff } \mathfrak{A} \models [\text{TC}_{\vec{X}, \vec{X}'}^k\theta](\vec{Y}, \vec{Y}').$$

From here on we consider τ and φ fixed; especially, by a constant, we mean a number that is independent of the model \mathfrak{A} ; that is, it may depend on τ and φ .

It suffices to show the left-to-right direction as the converse direction holds trivially for all k . Assume that $\mathfrak{A} \models [\text{TC}_{\vec{X}, \vec{X}'}\theta](\vec{Y}, \vec{Y}')$. By the semantics of TC there exists a natural number k_0 and tuples of relations $\vec{B}_0, \dots, \vec{B}_{k_0}$ on A such that $\vec{B}_0 = \vec{Y}^{\mathfrak{A}}$, $\vec{B}_{k_0} = \vec{Y}'^{\mathfrak{A}}$, and

$$\mathfrak{A}[\vec{B}_i/\vec{X}, \vec{B}_{i+1}/\vec{X}'] \models \theta, \text{ for } 0 \leq i < k_0. \quad (1)$$

It suffices to establish that, if k_0 is large enough, then there exists two natural numbers h and h' , $0 \leq h \leq h+3 \leq h' \leq k_0$, and an interpretation \vec{H} for \vec{X} such that

$$\mathfrak{A}[\vec{B}_h/\vec{X}, \vec{H}/\vec{X}'] \models \theta \text{ and } \mathfrak{A}[\vec{H}/\vec{X}, \vec{B}_{h'}/\vec{X}'] \models \theta.$$

For each $i < k_0$, let $\mathfrak{A}_i := \mathfrak{A}[\vec{B}_i/\vec{X}, \vec{B}_{i+1}/\vec{X}']$ and let σ denote the vocabulary of \mathfrak{A}_i . By the semantics of the existential quantifier, (1) is equivalent to saying that

$$\mathfrak{A}_i[\vec{a}_i/x_1, \dots, x_n] \models \psi, \text{ for } 0 \leq i < k_0, \quad (2)$$

for some n -tuples $\vec{a}_0, \dots, \vec{a}_{k_0-1}$ from A . We will prove the following claim.

Claim. There exists an index set I and $n + 2$ mutually pairwise I -compatible sequences in $\vec{a}_1, \dots, \vec{a}_{k_0-1}$ that have a common σ -type provided that k_0 is a large enough constant.

Proof of the claim. Let $\vec{c}^0 = (\vec{c}_0^0, \vec{c}_1^0, \dots, \vec{c}_t^0)$ denote the longest (not necessarily consecutive) subsequence of $\vec{a}_1, \dots, \vec{a}_{k_0-1}$ that have a common σ -type. Since there are only finitely many σ -types, t can be made as large as needed by making k_0 a large enough constant.

We will next show that there exists $n + 2$ mutually pairwise I -compatible sequences in \vec{c}^0 for some I (provided that t is large enough). Set $\text{SIM}_0 := \emptyset$. In the construction below we maintain the following properties for $0 \leq i \leq n$:

- For each $j \in \text{SIM}_i$ and for each tuple \vec{a} and \vec{b} in \vec{c}^i it holds that $\vec{a}[j] = \vec{b}[j]$.
- The length of \vec{c}^i is as long a constant as we want it to be.

For $l < n$, let $\vec{b}_0^l, \dots, \vec{b}_{t_l}^l$ be a maximal collection (in length) of mutually pairwise SIM_l -compatible sequences from \vec{c}^l . If $t_l \geq n + 1$ we are done. Otherwise note that, since each \vec{b}_j^l is an n -tuple, the number of different points that may occur in $\vec{b}_0^l, \dots, \vec{b}_{t_l}^l$ is $\leq n^2 + n$. By an inductive argument we may assume that the length of \vec{c}^l is as large a constant as we want, and thus we may conclude that there exists an index $i \notin \text{SIM}_l$ and an element d_l such that there are as many as we want tuples \vec{c}_j^l in \vec{c}^l such that $\vec{c}_j^l[i] = d_l$. Set $\text{SIM}_{l+1} := \text{SIM}_l \cup \{i\}$ and let \vec{c}^{l+1} be the sequence of exactly those $\vec{a} \in \vec{c}^l$ such that $\vec{a}[i] = d_l$. Notice that the length of \vec{c}^{l+1} is as large a constant as we want it to be.

Finally, the case $l = n$. Note that $\text{SIM}_n = \{0, \dots, n-1\}$ and \vec{c}^n is a sequence of n -tuples; in fact all tuples in \vec{c}^n are identical. Thus, if the length of \vec{c}^n is at least $n + 2$, the first $n + 2$ sequences of \vec{c}^n constitute a mutually pairwise SIM_n -compatible sequence of length $n + 2$. It is now straightforward but tedious to check how large k_0 has to be so that the length of \vec{c}^n is at least $n + 2$; thus the claim holds. \blacktriangleleft

Now let $\vec{a}_{i_0}, \dots, \vec{a}_{i_{n+1}}$, $0 < i_0 < \dots < i_{n+1}$, be mutually pairwise I -compatible sequences from $\vec{a}_1, \dots, \vec{a}_{k_0-1}$ with a common σ -type provided by the Claim. Let $1 \leq j \leq n + 1$ be an index such that \vec{a}_{i_0-1} and \vec{a}_{i_j} are pairwise compatible outside I and $\text{sim}(\vec{a}_{i_0-1}, \vec{a}_{i_j}) \subseteq I$. It is straightforward to check that such a j always exists, for if \vec{a}_{i_0-1} and $\vec{a}_{i_{j'}}$ are not pairwise compatible outside I or $\text{sim}(\vec{a}_{i_0-1}, \vec{a}_{i_{j'}}) \not\subseteq I$, there exists some indices $m, m' \notin I$ such that $\vec{a}_{i_0-1}[m] = \vec{a}_{i_{j'}}[m']$, and for each such $\vec{a}_{i_{j'}}$, the value of the related $\vec{a}_{i_{j'}}[m']$ has to be unique as $\vec{a}_{i_1}, \dots, \vec{a}_{i_{n+1}}$ are mutually pairwise I -compatible. Now j must exist since the length of $\vec{a}_{i_1}, \dots, \vec{a}_{i_{n+1}}$ is $n + 1$ while the length of \vec{a}_{i_0-1} is only n .

Consider the models $\mathfrak{A}_{i_0-1} = \mathfrak{A}[\vec{B}_{i_0-1}/\vec{X}, \vec{B}_{i_0}/\vec{X}']$ and $\mathfrak{A}_{i_j} = \mathfrak{A}[\vec{B}_{i_j}/\vec{X}, \vec{B}_{i_{j+i}}/\vec{X}']$ and recall that

$$\mathfrak{A}_{i_0-1}[\vec{a}_{i_0-1}/x_1, \dots, x_n] \models \psi \text{ and } \mathfrak{A}_{i_j}[\vec{a}_{i_j}/x_1, \dots, x_n] \models \psi.$$

We claim that there exists a sequence \vec{B} of relations on A such that

$$\mathfrak{A}[\vec{B}_{i_0-1}/\vec{X}, \vec{B}/\vec{X}', \vec{a}_{i_0-1}/x_1, \dots, x_n] \models \psi \text{ and } \mathfrak{A}[\vec{B}/\vec{X}, \vec{B}_{i_{j+i}}/\vec{X}', \vec{a}_{i_j}/x_1, \dots, x_n] \models \psi. \quad (3)$$

and thus that $\mathfrak{A}[\vec{B}_{i_0-1}/\vec{X}, \vec{B}/\vec{X}'] \models \theta$ and $\mathfrak{A}[\vec{B}/\vec{X}, \vec{B}_{i_{j+i}}/\vec{X}'] \models \theta$. From this the claim of the theorem follows for $k = k_0$.

It now suffices to show that such a \vec{B} exists. The idea is that \vec{B} looks exactly like \vec{B}_{i_0} with respect to points in \vec{a}_{i_0-1} and like \vec{B}_{i_j} with respect to points \vec{a}_{i_j} . Formally \vec{B} is defined as follows. For every relation $\vec{B}[m]$ and tuple $\vec{a} \in A^{\text{ar}(\vec{B}[m])}$

- if \vec{a} is completely included in neither \vec{a}_{i_0-1} nor \vec{a}_{i_j} then we set $\vec{a} \notin \vec{B}[m]$,
- if \vec{a} is completely included in \vec{a}_{i_0-1} then we set $\vec{a} \in \vec{B}[m]$ iff $\vec{a} \in \vec{B}_{i_0}[m]$,
- if \vec{a} is completely included in \vec{a}_{i_j} then we set $\vec{a} \in \vec{B}[m]$ iff $\vec{a} \in \vec{B}_{i_j}[m]$.

Note that if $\vec{a} = (a_1, \dots, a_m)$ is completely included in both \vec{a}_{i_0-1} and \vec{a}_{i_j} then there exists indices $j_1, \dots, j_m \in I$ such that, for $1 \leq l \leq m$, $a_l = \vec{a}_{i_j}[j_l] = \vec{a}_{i_0}[j_l]$. The former equality follows, with indices in I , since \vec{a}_{i_0-1} and \vec{a}_{i_j} are pairwise compatible outside I and $\text{sim}(\vec{a}_{i_0-1}, \vec{a}_{i_j}) \subseteq I$. The latter equality follows since \vec{a}_{i_0} and \vec{a}_{i_j} are pairwise I -compatible. Since \vec{a}_{i_0} and \vec{a}_{i_j} have the same σ -type $\vec{a} \in \vec{B}_{i_0}[m]$ iff $\vec{a} \in \vec{B}_{i_j}[m]$, for all m , and thus \vec{B} is well-defined. It is now immediate that (3) holds. \blacktriangleleft

► **Lemma 12.** *For every formula of vocabulary τ of the form $\exists X\theta$ or $[\text{TC}_{\vec{X}, \vec{X}', \theta}](\vec{Y}, \vec{Y}')$, where $\theta \in \exists\text{FO}$ and $\vec{X}, \vec{X}', \vec{Y}, \vec{Y}'$ are tuples of relation variables, there exists an equivalent formula $\varphi \in \exists\text{FO}$ of vocabulary τ .*

Proof. Consider first the formula $\exists X\theta$ (this collapse was remarked, but not proven, by Rosen in the introduction of his paper [35]). Define $n := \text{ar}(X)$ and let k be the number of occurrences of X in θ . The idea behind our translation is that the quantification of X can be equivalently replaced by a quantification of an n -ary relation of size $\leq k$; this can be then expressed in $\exists\text{FO}$ by quantifying k many n -tuples (content of the finite relation).

Let θ_\emptyset denote the formula obtained from θ by replacing every occurrence of the relation variable X of the form $X(\vec{x})$ in θ by the formula $\exists x(x \neq x)$. Define

$$\gamma := \exists \vec{x}_1 \dots \exists \vec{x}_k (\theta_\emptyset \vee \theta'),$$

where, for each i , $\exists \vec{x}_i$ is a shorthand for $\exists x_{1,i} \dots \exists x_{n,i}$ and θ' is the formula obtained from θ by substituting each occurrence of the relation variable X of the form $X(\vec{x})$ in θ by $\bigvee_{1 \leq i \leq k} (\vec{x} = \vec{x}_i)$. It is straightforward to check that γ is an $\exists\text{FO}$ -formula of vocabulary τ equivalent with $\exists X\theta$.

Consider then the formula $\varphi = [\text{TC}_{\vec{X}, \vec{X}', \theta}](\vec{Y}, \vec{Y}')$. In order to simplify the presentation, we stipulate that \vec{X} and \vec{X}' are of length one, that is, variables X and X' , respectively; the generalisation of the proof for arbitrary tuples of second-order variables is straightforward. By Lemma 11, we obtain $k \in \mathbb{N}$ such that φ and $\varphi' := [\text{TC}_{X, X', \theta}^k](Y, Y')$ are equivalent.

The following formulas are defined via substitution; by $\theta(A/B)$ we denote the formula obtained from θ by substituting each occurrence of the symbol B by the symbol A .

- $\theta_0^{\text{end}} := \theta(Y/X, Y'/X')$ and $\theta_i^{\text{end}} := \theta(X_i/X, Y'/X')$, for $1 \leq i < k$,
- $\theta_1^{\text{move}} := \theta(Y/X, X_1/X')$ and $\theta_i^{\text{move}} := \theta(X_{i-1}/X, X_i/X')$, for $2 \leq i < k$.

Let ψ denote the following formula of existential second-order logic

$$\exists X_1 \dots \exists X_{k-1} \bigvee_{0 \leq n < k} (\theta_n^{\text{end}} \wedge \bigwedge_{1 \leq i \leq n} \theta_i^{\text{move}}).$$

It is immediate that φ' and ψ are equivalent. Note that ψ is of the form $\exists X_1 \dots \exists X_{k-1} \psi'$, where ψ' is an $\exists\text{FO}$ -formula. By repetitively applying the first case of this lemma to subformulas of ψ , we eventually obtain an equivalent $\exists\text{FO}$ -formula over τ as required. \blacktriangleleft

The following theorem now follows by applying Lemma 12 repetitively bottom up.

► **Theorem 13.** *The expressive powers of $\text{SO}(2\text{TC})[\exists]$ and $\exists\text{FO}$ coincide.*

5 MSO(TC) and counting

We define a counting extension of $\text{MSO}(\text{TC})$ and show that the extension does not add expressive power to the logic. In this way, we demonstrate that quite a bit of queries involving counting can be expressed already in $\text{MSO}(\text{TC})$.

5.1 Syntax and semantics of CMSO(TC)

We assume a sufficient supply of *counter variables* or simply *counters*, which are a new sort of variables. We use the Greek letters μ and ν (with subscripts) to denote counter variables. The notion of a vocabulary is extended so that it may also contain counters. A structure \mathfrak{A} over a vocabulary τ is defined to be a pair (A, I) as before, where I now also maps the counters in τ to elements of $\{0, \dots, n\}$, where n is the cardinality of A .

We also assume a sufficient supply of *numeric predicates*. Intuitively numeric predicates are relations over natural numbers such as the tables of multiplication and addition. Technically, we use an approach similar to generalised quantifiers; a k -ary numeric predicate is a class $Q_p \subseteq \mathbb{N}^{k+1}$ of $k+1$ -tuples of natural numbers. For a numeric predicate Q_p , we use p as a symbol referring to the predicate. For simplicity, we often call p also numeric predicate. Note that when evaluating a k -ary numeric predicate $p(\mu_1, \dots, \mu_k)$ on a finite structure \mathfrak{A} , we let the numeric predicate Q_p access also the cardinality of the structure in question, and thus Q_p consists of $k+1$ -tuples and not k -tuples. This convention allows us, for example, to regard the modular sum $a + b \equiv c \pmod{n}$, where n refers to the cardinality of the structure, as a 3-ary numeric predicate.

We consider only those numeric predicates which can be decided in NLOGSPACE. Since, on finite ordered structures, first-order transitive closure logic captures NLOGSPACE, this boils down to being definable in first-order transitive closure logic when the counter variables are interpreted as points in an ordered structure representing an initial segment of natural numbers (see Definition 16 and Proposition 17 below for precise formulations). Note that the equality of numeric variables is also a 2-ary NLOGSPACE predicate.

- **Definition 14.** The syntax of CMSO(TC) extends the syntax of MSO(TC) as follows:
 - Let φ be a formula, μ a counter, and x a first-order variable. Then $\mu = \#\{x \mid \varphi\}$ is also a formula. The set of its free variables is defined to be $(\text{FV}(\varphi) - \{x\}) \cup \{\mu\}$.
 - If φ is a formula and μ a counter then also $\exists\mu\varphi$ is a formula with set of free variables $\text{FV}(\varphi) - \{\mu\}$.
 - Let μ_1, \dots, μ_k be counters and let p be a k -ary numeric predicate. Then $p(\mu_1, \dots, \mu_k)$ is a formula with the set of free variables $\{\mu_1, \dots, \mu_k\}$.
 - The scope of the transitive-closure operator is widened to apply as well to counters. Formally, in a formula of the form $[\text{TC}_{\vec{X}, \vec{X}'}\varphi](\vec{Y}, \vec{Y}')$, the variables in \vec{X} , \vec{X}' , \vec{Y} , and \vec{Y}' may also include counters. We still require that the tuples \vec{X} , \vec{X}' , \vec{Y} , and \vec{Y}' have the same sort, i.e., if a counter appears in some position in one of these tuples then a counter must appear in that position in each of the tuples.
- **Definition 15.** The satisfaction relation, $\mathfrak{A} \models \psi$, for CMSO(TC) formulas ψ and structures $\mathfrak{A} = (A, I)$ over a vocabulary appropriate for ψ is defined in the same way as for MSO(TC) with the following additional clauses.
 - Let ψ be of the form $\exists\mu\varphi$, where μ is a counter, and let n denote the cardinality of A . Then $\mathfrak{A} \models \psi$ iff there exists a number $i \in \{0, \dots, n\}$ such that $\mathfrak{A}[i/\mu] \models \varphi$.
 - Let ψ be of the form $\mu = \#\{x \mid \varphi\}$. Then $\mathfrak{A} \models \psi$ iff $I(\mu)$ equals the cardinality of the set $\{a \in A \mid \mathfrak{A}[a/x] \models \varphi\}$.
 - Let ψ be of the form $p(\mu_1, \dots, \mu_k)$, where μ_1, \dots, μ_k are counters and p is a k -ary numeric predicate. Then $\mathfrak{A} \models p(\mu_1, \dots, \mu_k)$ iff $(|A|, I(\mu_1), \dots, I(\mu_k)) \in Q_p$.
- **Definition 16.** A k -ary numeric predicate Q_p is *decidable in NLOGSPACE* if the membership $(n_0, \dots, n_k) \in Q_p$ can be decided by a nondeterministic Turing machine that uses logarithmic space when the numbers n_0, \dots, n_k are given in unary. Note that this is equivalent to linear space when n_0, \dots, n_k are given in binary.

From now on we restrict our attention to numeric predicates that are decidable in NLOGSPACE. The following proposition follows directly from a result of Immerman (Theorem 5) that, on ordered structures, FO(1TC) captures NLOGSPACE.

► **Proposition 17.** *For every k -ary numeric predicate Q_p decidable in NLOGSPACE there exists a formula φ_p of FO(1TC) over $\{s, x_1, \dots, x_k\}$, where s is a binary second-order variable and x_1, \dots, x_k are first-order variables, s.t. for all appropriate structures \mathfrak{A} for $p(\mu_1, \dots, \mu_k)$*

$$\mathfrak{A} \models p(\mu_1, \dots, \mu_k) \text{ iff } (|A|, I(\mu_1), \dots, I(\mu_k)) \in Q_p \text{ iff } (B, J) \models \varphi_p,$$

where $B = \{0, 1, \dots, |A|\}$, $J(s)$ is the successor relation of B , and $J(x_i) = I(\mu_i)$, for $1 \leq i \leq k$.

5.2 CMSO(TC) collapses to MSO(TC)

Let τ be a vocabulary with counters. Let τ^* denote the vocabulary without counters obtained from τ by viewing each counter variable of τ as a set variable. Let $\mathfrak{A} = (A, I)$ be a structure over τ , and let $\mathfrak{B} = (A, J)$ be a structure over τ^* with the same domain as \mathfrak{A} . We say that \mathfrak{B} *simulates* \mathfrak{A} if for every counter μ in τ , the set $J(\mu)$ has cardinality $I(\mu)$, and $J(X) = I(X)$, for each first-order or second-order variable $X \in \tau$. Let φ be a CMSO(TC)-formula over τ and ψ an MSO(TC) formula over τ^* . We say that ψ *simulates* φ if whenever \mathfrak{B} simulates \mathfrak{A} , we have that $\mathfrak{A} \models \varphi$ if and only if $\mathfrak{B} \models \psi$.

Let $\varphi(x)$ and $\psi(y)$ be formulae of some logic. The *Härtig quantifier* is defined as follows:

$$\mathfrak{A} \models \text{Hxy}(\varphi(x), \psi(y)) \Leftrightarrow \text{the sets } \{a \in A \mid \mathfrak{A}[a/x] \models \varphi\} \text{ and } \{b \in A \mid \mathfrak{A}[b/y] \models \psi\} \\ \text{have the same cardinality}$$

► **Proposition 18.** *The Härtig quantifier can be expressed in MSO(TC).*

Proof. Consider a structure (A, I) and monadic second-order variables X, Y, X' and Y' . Let $\psi_{\text{decrement}}$ denote an FO-formula expressing that $I(X') = I(X) \setminus \{a\}$ and $I(Y') = I(Y) \setminus \{b\}$, for some $a \in I(X)$ and $b \in I(Y)$. Define

$$\psi_{\text{ec}} := \exists X_\emptyset \left((\forall x \neg X_\emptyset(x)) \wedge [\text{TC}_{X, Y, X', Y'} \psi_{\text{decrement}}](Z, Z', X_\emptyset, X_\emptyset) \right).$$

It is straightforward to check that ψ_{ec} holds in (A, I) if and only if $|I(Z)| = |I(Z')|$. Therefore $\text{Hxy}(\varphi(x), \psi(y))$ is equivalent with the formula

$$\exists Z \exists Z' (\forall x (\varphi(x) \leftrightarrow Z(x)) \wedge \forall y (\psi(y) \leftrightarrow Z'(y)) \wedge \psi_{\text{ec}}),$$

assuming that Z, Z' are variable symbols that occur in neither φ nor ψ . ◀

► **Lemma 19.** *Let $\tau = \{s, x_1, \dots, x_n\}$ and $\sigma = \{X_1, \dots, X_n\}$ be vocabularies, where s is a binary second-order variable, x_1, \dots, x_n are first-order variables, and X_1, \dots, X_n are monadic second-order variables. For every FO(1TC)-formula φ over τ there exists an MSO(TC)-formula φ^+ over σ such that*

$$(A, I) \models \varphi \Leftrightarrow (B, J) \models \varphi^+,$$

for every (A, I) and (B, J) such that (A, I) is a structure over vocabulary τ , where $A = \{0, \dots, m\}$, for some $m \in \mathbb{N}$, and $I(s)$ is the canonical successor relation on A , and (B, J) is a structure over vocabulary σ such that $|B| = m$ and $|J(X_i)| = I(x_i)$, for $1 \leq i \leq n$.

Proof. We define the translation $^+$ recursively as follows. In the translation, we introduce for each first-order variable x_i a monadic second-order variable X_i by using the corresponding capital letter with the same index. Consequently, in tuples of variables, identities between the variables are maintained. The idea of the translation is that natural numbers i are simulated by sets of cardinality i . Identities between first-order variables are then simulated with the help of the Hartig quantifier, which, by Proposition 18, is definable in MSO(TC).

- For ψ of the form $x_i = x_j$, define $\psi^+ := \text{Hxy}(X_i(x), X_j(y))$.
- For ψ of the form $s(x_i, x_j)$, define $\psi^+ := \exists z (\neg X_i(z) \wedge \text{Hxy}(X_i(x) \vee x = z, X_j(y)))$.
- For ψ of the form $\neg\varphi$ and $(\varphi \wedge \theta)$, define ψ^+ as $\neg\varphi^+$ and $(\varphi^+ \wedge \theta^+)$, respectively.
- For ψ of the form $\exists x_i \varphi$, define $\psi^+ := \exists X_i \varphi^+$, where X_i is a monadic second-order variable.
- For ψ of the form $[\text{TC}_{\vec{x}, \vec{x}'} \varphi](\vec{y}, \vec{y}')$, define $\psi^+ := [\text{TC}_{\vec{X}, \vec{X}'} \varphi^+](\vec{Y}, \vec{Y}')$, where $\vec{X}, \vec{X}', \vec{Y}$, and \vec{Y}' are tuples of monadic second-order variables that correspond to the tuples \vec{x}, \vec{x}' , \vec{y} , and \vec{y}' of first-order variables.

The correctness of the translation follows by a simple inductive argument. ◀

With the help of the previous lemma, we are now ready to show how CMSO(TC)-formulas can be simulated in MSO(TC).

► **Theorem 20.** *Every CMSO(TC)-formula can be simulated by an MSO(TC)-formula.*

Proof. Let τ be a vocabulary with counters and τ^* the vocabulary without counters obtained from τ by viewing each counter as a set variable. We define recursively a translation * that maps CMSO(TC)-formulas over vocabulary τ to MSO(TC)-formulas over τ^* .

- For ψ of the form $x_i = x_j$, define $\psi^* := x_i = x_j$.
- For ψ of the form $X(x_1, \dots, x_n)$, define $\psi^* := X(x_1, \dots, x_n)$.
- For an NLOGSPACE numeric predicate Q_p and ψ be of the form $p(\mu_1, \dots, \mu_k)$, define ψ^* as $\varphi_p^+(\mu_1/X_1, \dots, \mu_k/X_k)$, where $^+$ is the translation defined in Lemma 19 and φ_p the defining formula of Q_p obtained from Proposition 17.
- For ψ of the form $\mu = \#\{x \mid \varphi\}$, define ψ^* as the MSO(TC)-formula $\text{Hxy}(\varphi^*, \mu(y))$.
- For ψ of the form $\neg\varphi$ and $(\varphi \wedge \theta)$, define ψ^* as $\neg\varphi^*$ and $(\varphi^* \wedge \theta^*)$, respectively.
- For ψ of the form $\exists x_i \varphi$, $\exists \mu_i \varphi$, and $\exists X_i \varphi$, define ψ^* as $\exists x_i \varphi^*$, $\exists \mu_i \varphi^*$, and $\exists X_i \varphi^*$. Remember that, on the right, μ_i is treated as a monadic second-order variable.
- For ψ of the form $[\text{TC}_{\vec{X}, \vec{X}'} \varphi](\vec{Y}, \vec{Y}')$, define $\psi^* := [\text{TC}_{\vec{X}, \vec{X}'} \varphi^*](\vec{Y}, \vec{Y}')$.

We claim that, for every CMSO(TC)-formula ψ over τ , ψ^* is an MSO(TC)-formula over τ^* that simulates ψ . Correctness of the simulation follows by induction using Lemma 19 and Proposition 17.

We show the case for the numeric predicates. Let $\mathfrak{A} = (A, I)$ be a τ -structure and \mathfrak{A}^* a τ^* -structure that simulates \mathfrak{A} . Let Q_p be a k -ary NLOGSPACE numeric predicate, μ_1, \dots, μ_k counters from τ , and φ_p the defining FO(1TC)-formula of Q_p given by Proposition 17. Then, by Proposition 17,

$$\mathfrak{A} \models p(\mu_1, \dots, \mu_k) \text{ iff } (B, J) \models \varphi_p,$$

where $B = \{0, 1, \dots, |A|\}$, $J(s)$ is the successor relation of B , and $J(x_i) = I(\mu_i)$, for $1 \leq i \leq k$. Let $^+$ denote the translation from FO(1TC) to MSO(TC) defined in Lemma 19. Then, by Lemma 19, it follows that $(B, J) \models \varphi_p$ iff $\mathfrak{A} \models \varphi_p^+$. ◀

In the next example, we introduce notation for some MSO(TC)-definable numeric predicates that are used in the following sections.

► **Example 21.** Let k be a natural number, X, Y, Z, X_1, \dots, X_n monadic second-order variables, and $\mathfrak{A} = (A, I)$ an appropriate structure. The following numeric predicates are clearly NLOGSPACE-definable and thus, by Theorem 20, definable in MSO(TC):

- $\mathfrak{A} \models \text{size}(X, k)$ iff $|I(X)| = k$,
- $\mathfrak{A} \models \times(X, Y, Z)$ iff $|I(X)| \times |I(Y)| = |I(Z)|$,
- $\mathfrak{A} \models +(X_1, \dots, X_n, Y)$ iff $|I(X_1)| + \dots + |I(X_n)| = |I(Y)|$.

6 Order-invariant MSO

Order-invariance plays an important role in finite model theory. In descriptive complexity theory many characterisations rely on the existence of a linear order. However the particular order in a given structure is often not important. Related to applications in computer science, it is often possible to access an ordering of the structure that is not controllable and thus a use of the ordering should be such that change in the ordering should not make a difference. Consequently, in both cases order can be used, but in a way that the described properties are *order-invariant*.

Let $\tau_{\leq} := \tau \cup \{\leq\}$ be a finite vocabulary, where \leq is a binary relation symbol. A formula $\varphi \in \text{MSO}$ over τ_{\leq} is *order-invariant*, if for every τ -structure \mathfrak{A} and expansions \mathfrak{A}' and \mathfrak{A}^* of \mathfrak{A} to the vocabulary τ_{\leq} , in which $\leq^{\mathfrak{A}'}$ and $\leq^{\mathfrak{A}^*}$ are linear orders of A , we have that $\mathfrak{A}' \models \varphi$ if and only if $\mathfrak{A}^* \models \varphi$. A class \mathcal{C} of τ -structures is *definable in order-invariant MSO* if and only if the class $\{(\mathfrak{A}, \leq) \mid \mathfrak{A} \in \mathcal{C} \text{ and } \leq \text{ is a linear order of } A\}$ is definable by some order-invariant MSO-formula.

We call a vocabulary τ a *unary vocabulary* if it consists of only monadic second-order variables. In this section we establish that on unary vocabularies MSO(TC) is strictly more expressive than order-invariant MSO. The separation holds already for the empty vocabulary.

6.1 Separation on empty vocabulary

First note that over vocabulary $\{\leq\}$ there exists only one structure, up to isomorphism, of size k , for each $k \in \mathbb{N}$, in which \leq is interpreted as a linear order of the domain. Consequently, every MSO-formula of vocabulary $\{\leq\}$ is order-invariant. Also note that, in fact, $\{\leq\}$ -structures interpreted as word models correspond to finite strings over some fixed unary alphabet. Thus, via Büchi's theorem, we obtain that, over the empty vocabulary, order-invariant MSO captures essentially regular languages over unary alphabets. Hence, to separate MSO(TC) from order-invariant MSO over the empty vocabulary, it suffices to observe that not all NLOGSPACE properties of unary strings are regular (recall Theorem 5 and Lemma 19). The following example gives a concrete example of the separation.

► **Example 22.** Consider the class $\mathcal{C} = \{\mathfrak{A} \mid |A| \text{ is a prime number}\}$ of \emptyset -structures. Clearly the language of prime length words over some unary alphabet is not regular and thus it follows via Büchi's theorem that \mathcal{C} is not definable in order-invariant MSO. However the following formula of MSO(TC) defines \mathcal{C} . We use MSO(TC)-definable numeric predicates introduced in Example 21.

$$\exists X \forall Y \forall Z (\forall x (X(x)) \wedge (\text{size}(Y, 1) \vee \text{size}(Z, 1) \vee \neg \times(Y, Z, X))) \wedge \exists x \exists y \neg x = y.$$

► **Corollary 23.** *For any vocabulary τ , there exists a class \mathcal{C} of τ -structures such that \mathcal{C} is definable in MSO(TC) but it is not definable in order-invariant MSO.*

6.2 Inclusion on unary vocabularies

We will show that every class of structures over a unary vocabulary τ that is definable in order-invariant MSO is also definable in MSO(TC).

► **Definition 24.** For a finite word w of some finite alphabet $\Sigma = \{a_1 \dots, a_k\}$, a *Parikh vector* $p(w)$ of w is the k -tuple $(|w|_{a_1}, \dots, |w|_{a_k})$ where $|w|_{a_i}$ denotes the number of a_i s in w . A *Parikh image* $P(L)$ of a language L is the set $\{p(w) \mid w \in L\}$ of Parikh vectors of the words in the language.

A subset S of \mathbb{N}^k is a *linear set* if $S = \{\vec{v}_0 + \sum_{i=1}^m a_i \vec{v}_i \mid a_1, \dots, a_m \in \mathbb{N}\}$ for some *offset* $\vec{v}_0 \in \mathbb{N}^k$ and *generators* $\vec{v}_1, \dots, \vec{v}_m \in \mathbb{N}^k$.

► **Theorem 25** (Parikh's theorem, [30]). *For every regular language L its Parikh image $P(L)$ is a finite union of linear sets.*

We use the following improved version of Parikh's theorem:

► **Theorem 26** ([26]). *For every regular language L over alphabet of size k its Parikh image $P(L)$ is a finite union of linear sets with at most k generators.*

► **Definition 27.** Let $\tau = \{X_1, \dots, X_k\}$ be a finite unary vocabulary and let Y_1, \dots, Y_{2^k} denote the Boolean combinations of the variables in τ in some fixed order. For every structure $\mathfrak{A} = (A, I)$ over τ , we extend the scope of I to include also Y_1, \dots, Y_{2^k} in the obvious manner. The *Parikh vector* $p(\mathfrak{A})$ of \mathfrak{A} is the 2^k -tuple $(|I(Y_1)|, \dots, |I(Y_{2^k})|)$. A *Parikh image* $P(\mathcal{C})$ of a class of τ -structures \mathcal{C} is the set $\{p(\mathfrak{A}) \mid \mathfrak{A} \in \mathcal{C}\}$.

► **Theorem 28.** *Over finite unary vocabularies MSO(TC) is strictly more expressive than order-invariant MSO.*

Proof. Strictness follows directly from Corollary 23 and thus it suffices to establish inclusion. Let $\tau = \{X_1, \dots, X_k\}$ be a finite unary vocabulary and φ an order-invariant MSO-formula of vocabulary τ_{\leq} . Let \mathcal{C} be the class of τ structures that φ defines. We will show that \mathcal{C} is definable in MSO(TC). Set $n := 2^k$ and let Y_1, \dots, Y_n denote the Boolean combinations of the variables in τ in some fixed order; we regard these combinations also as fresh monadic second-order variables and set $\sigma := \{Y_1, \dots, Y_n\}$. For each X_i , let χ_i denote the disjunction of those variables Y_j in which X_i occurs positively. Let \mathcal{C}_{\leq} denote the class of τ_{\leq} -structures that φ defines. We may view \mathcal{C}_{\leq} also as a language L over the alphabet σ and as the class L_w of σ_{\leq} -structures corresponding to the word models of the language L . Let φ^* denote the order-invariant MSO-formula over σ_{\leq} obtained from φ by substituting each variable X_i by the formula χ_i . Since φ^* clearly defines L_w , by Büchi's Theorem, L is regular. Consequently, by the improved version of Parikh's Theorem (Theorem 26), the Parikh image $P(L)$ of L is a finite union of linear sets with at most n generators.

Observe that if two τ -structures have the same Parikh image, the structures are isomorphic. Thus \mathcal{C} is invariant under Parikh images. Hence \mathcal{C} is uniquely characterised by its Parikh image $P(\mathcal{C})$, which, since $P(L) = P(\mathcal{C})$, is a finite union of linear sets with at most n generators.

Claim. For every linear set $A \subseteq \mathbb{N}^n$, where $n = 2^k$, there exists a formula φ_A of MSO(TC) of vocabulary $\tau = \{X_1, \dots, X_k\}$ such that φ_A defines the class of τ -structures that have A as their Parikh image.

With the help of the above claim, the theorem follows in a straightforward manner. Let A_1, \dots, A_m be a finite collection of linear sets such that $P(\mathcal{C}) = A_1 \cup \dots \cup A_m$ and let $\varphi_{A_1}, \dots, \varphi_{A_m}$ be the related MSO(TC)-formulas of vocabulary τ provided by the claim. Clearly $\psi := \varphi_{A_1} \vee \dots \vee \varphi_{A_m}$ defines \mathcal{C} .

Proof of the Claim. Let $A \subseteq \mathbb{N}^n$ be a linear set with n generators, i.e.,

$$A = \left\{ \vec{v}_0 + \sum_{j=1}^n a_j \vec{v}_j \mid a_1, \dots, a_n \in \mathbb{N} \right\}, \text{ for some } \vec{v}_0, \vec{v}_1, \dots, \vec{v}_n \in \mathbb{N}^n.$$

For each tuple $\vec{v} \in \mathbb{N}^n$ and n -tuple of monadic second-order variables \vec{Z} , let $\text{size}(\vec{Z}, \vec{v})$ denote the FO-formula stating that, for each i , the size of the extension of $\vec{Z}[i]$ is $\vec{v}[i]$. For $0 \leq i \leq n$, we introduce fresh distinct n -tuples of monadic variable symbols \vec{Z}_i and define

$$\varphi_{\text{gen}} := \bigwedge_{0 \leq i \leq n} \text{size}(\vec{Z}_i, \vec{v}_i).$$

Let $\vec{R}_1, \dots, \vec{R}_n$ be fresh distinct n -tuples of monadic second-order variables and let S_1, \dots, S_n be fresh distinct monadic second-order variables. Define

$$\varphi_A^* := \exists \vec{Z}_0 \dots \vec{Z}_n \vec{R}_1 \dots \vec{R}_n S_1 \dots S_n \varphi_{\text{gen}} \wedge \bigwedge_{1 \leq i, j \leq n} \times(\vec{Z}_i[j], S_i, \vec{R}_i[j]) \wedge \bigwedge_{1 \leq i \leq n} +(\vec{Z}_0[i], \vec{R}_1[i], \dots, \vec{R}_n[i], Y_i), \quad (4)$$

where \times and $+$ refer to the MSO(TC)-formulas defined in Example 21. Finally define $\varphi_A := \exists Y_1 \dots Y_n \varphi_{BC} \wedge \varphi_A^*$, where φ_{BC} is an FO-formula stating that, for each i , the extension of Y_i is the extension of the Boolean combination of the variables in τ that Y_i represents. A τ -structure \mathfrak{B} satisfies φ_A if and only if the Parikh image of \mathfrak{B} is A . \blacktriangleleft

7 Conclusion

There are quite a number of interesting challenging questions regarding the expressive power within second-order transitive-closure logic.

1. We have shown that MSO(TC) can do counting, and thus can certainly express some queries not expressible in fixpoint logic FO(LFP). A natural question is whether MSO(TC) can also be separated from the counting extension of FO(LFP). Note that MSO(TC) can express numerical predicates in NLOGSPACE, while counting fixpoint logic can express numerical predicates in PTIME. Thus, over the empty vocabulary, the question seems related to a famous open problem from complexity theory. Note however, that it is not even clear that MSO(TC) can *only* express numerical predicates in NLOGSPACE. Over graphs, the answer is probably affirmative as the CFI query can probably be expressed in MSO(TC).
2. The converse question, whether there is a fixpoint logic query not expressible in MSO(TC), is fascinating. On ordered structures, this would show that there are problems in PTIME that are not in NLIN, which is open (we only know that the two classes are different [29]). On unordered structures, however, we actually conjecture that the query about a binary relation (transition system) R and two nodes a and b , that asks whether a and b are *bisimilar* w.r.t. R , is not expressible in MSO(TC).
3. In stating Theorem 5 we recalled that $\text{SO}(\text{arity } k)(\text{TC})$ captures the complexity class $\text{NSPACE}(n^k)$, on strings. What about ordered structures in general? Using the standard adjacency matrix encoding of a relational structure as a string [25], it follows that on ordered structures over vocabularies with maximal arity a , $\text{SO}(\text{arity } k \cdot a)(\text{TC})$ can express

all queries in $\text{NSPACE}(n^k)$. Can we show that this blowup in arity is necessary? For example, can we show that $\text{MSO}(\text{TC})$ does *not* capture NLIN over ordered graphs (binary relations)?

4. In the previous section we have clarified the relationship between $\text{MSO}(\text{TC})$ and order-invariant MSO , over unary vocabularies. What about higher arities?

References

- 1 Serge Abiteboul and Victor Vianu. Fixpoint extensions of first-order logic and datalog-like languages. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 71–79. IEEE Computer Society, 1989. doi:10.1109/LICS.1989.39160.
- 2 Jean-Raymond Abrial. *The B-book - Assigning programs to meanings*. Cambridge University Press, 2005.
- 3 Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.
- 4 Faisal N. Abu-Khzam and Michael A. Langston. Graph coloring and the immersion order. In *Computing and Combinatorics, 9th Annual International Conference (COCOON 2003)*, pages 394–403, 2003.
- 5 Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '13*, pages 213–224, New York, NY, USA, 2013. ACM. doi:10.1145/2463664.2465223.
- 6 Andreas Blass, Yuri Gurevich, and Jan Van den Bussche. Abstract state machines and computationally complete query languages. *Information and Computation*, 174(1):20–36, 2002. doi:10.1006/inco.2001.3067.
- 7 Andreas Blass, Yuri Gurevich, and Saharon Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100(1):141–187, 1999. doi:10.1016/S0168-0072(99)00005-6.
- 8 Joshua Blinkhorn and Olaf Beyersdorff. Shortening QBF proofs with dependency schemes. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT 2017*, pages 263–280, Cham, 2017. Springer International Publishing.
- 9 Béla Bollobás. *Modern Graph Theory*, volume 184 of *Graduate Texts in Mathematics*. Springer, 2002.
- 10 E. Börger and R. F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, 2003.
- 11 Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable highly expressive query languages - IJCAI-15 distinguished paper (honorary mention). In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2826–2832. AAAI Press, 2015.
- 12 Bogdan S Chlebus. Domino-tiling games. *J. Comput. Syst. Sci.*, 32(3):374–392, 1986. doi:10.1016/0022-0000(86)90036-X.
- 13 Bruno Courcelle. The monadic second-order logic of graphs VIII: orientations. *Ann. Pure Appl. Logic*, 72(2):103–143, 1995. doi:10.1016/0168-0072(95)94698-V.
- 14 Stephen Dill, Ravi Kumar, Kevin S. Mccurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the web. *ACM Trans. Internet Technol.*, 2(3):205–223, 2002.
- 15 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory. Perspectives in Mathematical Logic*. Springer, 1995.

- 16 Flavio Ferrarotti. *Expressibility of Higher-Order Logics on Relational Databases: Proper Hierarchies*. PhD thesis, Massey University, Wellington, New Zealand, 2008. URL: <http://hdl.handle.net/10179/799>.
- 17 Flavio Ferrarotti, Senén González, and José Maria Turull Torres. On fragments of higher order logics that on finite structures collapse to second order. In Juliette Kennedy and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information, and Computation - 24th International Workshop, WoLLIC 2017, London, UK, July 18-21, 2017, Proceedings*, volume 10388 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2017. doi:10.1007/978-3-662-55386-2_9.
- 18 Flavio Ferrarotti, Wei Ren, and Jose Maria Turull Torres. Expressing properties in second- and third-order logic: hypercube graphs and SATQBF. *Logic Journal of the IGPL*, 22(2):355–386, 2014. doi:10.1093/jigpal/jzt025.
- 19 Martin Grohe, Kenichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011)*, pages 479–488. ACM, 2011.
- 20 R. Guimerà, L. Danon, A. Díaz-Guilera, F. Giralt, and A. Arenas. Self-similar community structure in a network of human interactions. *Phys. Rev. E*, 68:065103, Dec 2003.
- 21 David Harel and David Peleg. On static logics, dynamic logics, and complexity classes. *Information and Control*, 60(1-3):86–102, 1984. doi:10.1016/S0019-9958(84)80023-6.
- 22 Marijn J. H. Heule, Martina Seidl, and Armin Biere. Solution validation and extraction for QBF preprocessing. *J. Autom. Reasoning*, 58(1):97–125, 2017. doi:10.1007/s10817-016-9390-4.
- 23 Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, aug 1987. doi:10.1137/0216051.
- 24 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988. doi:10.1137/0217058.
- 25 Neil Immerman. *Descriptive Complexity*. Springer, 1998.
- 26 E. Kopczynski and A. W. To. Parikh images of grammars: Complexity and applications. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 80–89, July 2010. doi:10.1109/LICS.2010.21.
- 27 Richard Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.*, 6:467–480, 1977.
- 28 Leslie Lamport. *Specifying Systems, The TLA⁺ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.
- 29 C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- 30 Rohit J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.
- 31 Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency learning for QBF. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT 2017*, pages 298–313, Cham, 2017. Springer International Publishing.
- 32 Albert Réka. Scale-free networks in cell biology. *Journal of Cell Science*, 118(21):4947–4957, 2005.
- 33 Juan L. Reutter, Miguel Romero, and Moshe Y. Vardi. Regular queries on graph databases. In *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, pages 177–194, 2015. doi:10.4230/LIPIcs.ICDT.2015.177.
- 34 David Richerby. Logical characterizations of PSPACE. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 20-24, 2004, Proceedings*, volume 3210 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2004. doi:10.1007/978-3-540-30124-0_29.

22:18 Expressivity Within Second-Order Transitive-Closure Logic

- 35 E. Rosen. An existential fragment of second order logic. *Archive for Mathematical Logic*, 38(4–5):217–234, 1999.
- 36 Sebastian Rudolph and Markus Krötzsch. Flag & check: Data access with monadically defined queries. In *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '13, pages 151–162, New York, NY, USA, 2013. ACM. doi:10.1145/2463664.2465227.
- 37 Chaoming Song, Shlomo Havlin, and Hernán A. Makse. Self-similarity of complex networks. *Nature*, 433:392–395, 2005.


Quantifying Bounds in Strategy Logic

Nathanaël Fijalkow¹

CNRS, LaBRI, Bordeaux, France

Alan Turing Institute of data science, London, United Kingdom


nfijalkow@turing.ac.uk

 <https://orcid.org/0000-0002-6576-4680>

Bastien Maubert²

University of Naples “Federico II”, Naples, Italy

bastien.maubert@gmail.com

 <https://orcid.org/0000-0002-9081-2920>

Aniello Murano

University of Naples “Federico II”, Naples, Italy

murano@na.infn.it

Sasha Rubin

University of Naples “Federico II”, Naples, Italy

rubin@unina.it

Abstract

Program synthesis constructs programs from specifications in an automated way. Strategy Logic (SL) is a powerful and versatile specification language whose goal is to give theoretical foundations for program synthesis in a multi-agent setting. One limitation of Strategy Logic is that it is purely qualitative. For instance it cannot specify quantitative properties of executions such as “every request is quickly granted”, or quantitative properties of trees such as “most executions of the system terminate”. In this work, we extend Strategy Logic to include quantitative aspects in a way that can express bounds on “how quickly” and “how many”. We define Prompt Strategy Logic, which encompasses Prompt LTL (itself an extension of LTL with a prompt eventuality temporal operator), and we define Bounded-Outcome Strategy Logic which has a bounded quantifier on paths. We supply a general technique, based on the study of automata with counters, that solves the model-checking problems for both these logics.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Prompt LTL, Strategy Logic, Model checking, Automata with counters

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.23

1 Introduction

In order to reason about strategic aspects in distributed systems, temporal logics of programs (such as LTL [38], CTL [5] and CTL* [19]) have been extended with operators expressing the existence of strategies for coalitions of components. Among the most successful proposals are Alternating-time Temporal Logic (ATL) [3] and, more recently, the more expressive Strategy Logic (SL) [13, 36]. Both logics can express the existence of strategies for coalitions that

¹ This project has received funding from the Alan Turing Institute under EPSRC grant EP/N510129/1 and the DeLTA project (ANR-16-CE40-0007).

² This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 709188.



ensure some temporal properties against all possible behaviours of the remaining components. Moreover, if such strategies exist, one can also obtain witnessing finite-state strategies. As a result, synthesizing reactive systems from temporal specifications [39, 30, 31] can be reduced to model checking such strategic logics.

Although quite expressive, for instance Strategy Logic can express important game-theoretic concepts such as the existence of Nash equilibria, such logics can only express qualitative properties. On the other hand important properties of distributed systems, such as bounding the maximal number of steps between an event and its reaction, are quantitative. Parametric extensions of temporal logics have been introduced to capture such properties.

A simple way to extend temporal operators is to annotate them with constant bounds, e.g., $\mathbf{F}^{\leq k}\varphi$ says that φ holds within k steps where $k \in \mathbb{N}$ is a constant. However, one may not know such bounds or care for their exact value when writing the specification (or it may not be practical to compute the bound). Instead, one may replace the constants by variables N and ask about the possible valuations of the variables that make the formula true. For instance, PROMPT-LTL [2, 32] is an extension of LTL with the operator $\mathbf{F}^{\leq N}$ where N is a (unique) variable. The model-checking problem asks if there exists a valuation of the variable N such that the formula holds. In order to reason about and synthesize strategies that ensure such parametric properties, we introduce “Prompt Strategy Logic”, an extension of SL with the $\mathbf{F}^{\leq N}$ operator. For instance, the formula $\exists s_1(a_1, s_1)\forall s_2(a_2, s_2)\exists N\mathbf{AGF}^{\leq N}p$ expresses that there exists a strategy for agent a_1 such that for all strategies of agent a_2 there is a bound N (that can depend on the strategy for a_2) such that in all outcomes (generated by the remaining agents) the atom p holds at least once every N steps.

Another way to parameterise temporal logics is to bound the path quantifiers, expressing, for instance, that at least k different paths satisfy ψ , or all but k paths satisfy ψ [8]. Such operators can bound, for instance, how well a linear-time temporal property holds, thus giving a measure of “coverage”. We introduce “Bounding-Outcome Strategy Logic” which extends Strategy Logic with a *bounded outcome quantifier* $\mathbf{A}^{\leq N}$ which allows one to express that all but N outcomes satisfy some property. For instance, the formula $\exists s(a, s)\exists N\mathbf{A}^{\leq N}\mathbf{GF}p$ expresses that there exists a strategy for agent a such that for all but finitely many outcomes, the atom p holds infinitely often. The algorithmic contribution of this paper is a solution to the model-checking problem for both these logics (and their combination). We do this by applying the theory of regular cost functions. A cost function is an equivalence class of mappings from the domain (e.g., infinite words) to $\mathbb{N} \cup \{\infty\}$ with an equivalence relation that, intuitively speaking, forgets the precise values and focuses on boundedness [14, 16].

Our results allow us to solve a problem left open in [10] that considers games with two players and a third player called “nature” (indicating that it is uncontrollable), and asks whether there is a strategy for player 1 (having very general linear-time objectives) such that for all strategies of player 2, in the resulting tree (i.e., where nature’s strategy is not fixed), the number of plays in which player 1 does not achieve her objective is “small”. In particular, in case the linear-time objective is the LTL formula ψ and “small” is instantiated to mean “finite”, our main result allows one to solve this problem by reducing to model checking Bounding-outcome Strategy Logic formula $\exists s_1(a_1, s_1)\forall s_2(a_2, s_2)\exists N\mathbf{A}^{\leq N}\neg\psi$. In fact our automata construction can be adapted to deal with all omega-regular objectives.

Related work. Parametric-LTL [2] extends LTL with operators of the form $\mathbf{F}^{\leq x}$ and $\mathbf{G}^{\leq x}$, where x is a variable. The interpretation of $\mathbf{F}^{\leq x}\psi$ is that ψ holds within x steps, and the interpretation of $\mathbf{G}^{\leq x}$ is that ψ holds for at least the next x steps. That paper studies variations on the classic decision problems, e.g., model checking asks to decide if there is a

valuation of the variables x_1, \dots, x_k such that the formula $\varphi(x_1, \dots, x_k)$ holds in the given structure. Note that for this problem, the formula is equivalent to one in which all variables are replaced by a single variable. The complexity of these problems is no worse than for ordinary LTL, i.e., PSPACE. The technique used in this paper to prove these upper-bounds is a pumping lemma that allows one to reduce/enlarge the parameters.

Parametric-LTL has been studied in the context of open systems and games. For instance, [41] studies the problem of synthesizing a strategy for an agent with a parametric-LTL objective in a turn-based graph-game against an adversarial environment. A number of variations are studied, e.g., decide whether there exists a valuation (resp. for all valuations) of the variables such that there exists a strategy for the agent that enforces the given parametric-LTL formula. The complexity of these problems is, again, no worse than that of solving ordinary LTL games, i.e., 2EXPTIME. The technique used to prove these upper bounds is the alternating-colour technique of [32] that allows one to replace a prompt formula by an LTL formula, and was originally introduced to reason about PROMPT-LTL, the fragment of parametric-LTL without $\mathbf{G}^{\leq x}$. We remark that Church's synthesis for PROMPT-LTL formulas was shown in [32] to have complexity no worse than that of LTL, i.e., 2EXPTIME.

Promptness was first studied in the context of multi-agent systems in [4]. They study the model-checking problem for the logic PROMPT-ATL* and its fragments, for memoryless and memoryful strategies. Again, one finds that the complexity of model checking prompt variations is no worse than the non-prompt ones. That paper also studies the case of systems with imperfect information. We remark that the formula of Prompt Strategy Logic mentioned above is not a formula of PROMPT-ATL* because the bound N can depend on the strategy of agent a_2 , which is not possible in PROMPT-ATL*.

Promptness has also been studied in relation with classic infinitary winning conditions in games on graphs. In bounded parity games, the even colors represent requests and odd colors represent grants, and the objective of the player is to ensure that every request is promptly followed by a larger grant [12, 37]. We discuss this in Example 6. Such winning conditions have been generalised to games with costs in [21, 22], leading to the construction of efficient algorithms for synthesizing controllers with prompt specifications.

Promptness in automata can be studied using various notions of automata with counters that only affect the acceptance condition. For instance, a run in a prompt Büchi-automaton is successful if there is a bound on the time between visits to the Büchi set. The expressive power, the cost of translating between such automata, and complexity of decision problems (such as containment) have been studied in [1, 12].

The theory of regular cost functions [14, 16] defines automata and logics able to express boundedness properties in various settings. For instance, the logics PROMPT-LTL, PLTL and kTL are in some precise sense subsumed by the LTL^{\leq} logic from [27], which extends LTL with a bounded until $\varphi \mathbf{U}^{\leq N} \varphi'$ allowing φ not to hold in at most N (possibly non-consecutive) places before φ' holds. A decision procedure for this logic has been given through the compilation into cost automata on words. In this paper, we rely on several results from the theory of regular cost functions, and develop some new ones for the study of Prompt Strategy Logic and Bounding-outcome Strategy Logic. A major open problem in the theory of regular cost functions over infinite trees is the equivalence between general cost automata. To handle the bounded until operator in branching-time logics one would need to first prove this equivalence, which has been proved to be beyond our reach today [20]. In this work we rely on a weaker version of this equivalence for distance automata.

To the best of our knowledge, the only previous works on quantitative extensions of Strategy Logic consider games with counters and allow for the expression of constraints on their values in formulas. The model-checking problem for these logics is undecidable, even

when restricted to the case of *energy constraints*, which can only state that the counters remain above certain thresholds [24]. For the Boolean Goal fragment of Strategy Logic in the case of one counter, the problem is still open [9, 24]. The present work thus provides the first decidable quantitative extension of Strategy Logic.

Plan. In Section 2 we recall Branching-time Strategy Logic. We introduce and motivate our two quantitative extensions, PROMPT-SL and BOSL, in Section 3 and Section 4 respectively. In Section 5 we solve their model-checking problem by introducing the intermediary logic BOUND-QCTL* and developing an automata construction based on automata with counters.

2 Branching-time Strategy Logic

In this section we recall Branching-time Strategy Logic [26], a variant of Strategy Logic [36].

For the rest of the paper we fix a number of parameters: AP is a finite set of *atomic propositions*, Ag is a finite set of *agents* or *players*, Act is a finite set of *actions*, and Var is a finite set of *strategy variables*. The alphabet is $\Sigma = 2^{\text{AP}}$.

Notations. A *finite* (resp. *infinite*) *word* over Σ is an element of Σ^* (resp. Σ^ω). The *length* of a finite word $w = w_0w_1 \dots w_n$ is $|w| = n + 1$, and $\text{last}(w) = w_n$ is its last letter. Given a finite (resp. infinite) word w and $0 \leq i < |w|$ (resp. $i \in \mathbb{N}$), we let w_i be the letter at position i in w , $w_{\leq i}$ is the prefix of w that ends at position i and $w_{\geq i}$ is the suffix of w that starts at position i . We write $w \preceq w'$ if w is a prefix of w' . The cardinal of a set S is written $\text{Card}(S)$.

2.1 Games

We start with classic notions related to concurrent games on graphs.

► **Definition 1 (Game).** A *concurrent game structure* (or *game* for short) is a structure $\mathcal{G} = (V, v_0, \Delta, \ell)$ where V is the set of *vertices*, $v_0 \in V$ is the *initial vertex*, $\Delta : V \times \text{Act}^{\text{Ag}} \rightarrow V$ is the *transition function*, and $\ell : V \rightarrow \Sigma$ is the *labelling function*.

Joint actions. In a vertex $v \in V$, each player $a \in \text{Ag}$ chooses an action $c(a) \in \text{Act}$, and the game proceeds to the vertex $\Delta(v, \mathbf{c})$, where $\mathbf{c} \in \text{Act}^{\text{Ag}}$ stands for the *joint action* $(c(a))_{a \in \text{Ag}}$. Given a joint action $\mathbf{c} = (c(a))_{a \in \text{Ag}}$ and $a \in \text{Ag}$, we let $\mathbf{c}(a)$ denote $c(a)$.

Plays and strategies. A *finite* (resp. *infinite*) *play* is a finite (resp. infinite) word $\rho = v_0 \dots v_n$ (resp. $\pi = v_0v_1 \dots$) such that for every i such that $0 \leq i < |\rho| - 1$ (resp. $i \geq 0$), there exists a joint action \mathbf{c} such that $\Delta(v_i, \mathbf{c}) = v_{i+1}$. A *strategy* is a partial function $\sigma : V^+ \rightarrow \text{Act}$ mapping each finite play to an action, and Strat is the set of all strategies.

Assignments. An *assignment* is a partial function $\chi : \text{Ag} \cup \text{Var} \rightarrow \text{Strat}$, assigning to each player and variable in its domain a strategy. For an assignment χ , a player a and a strategy σ , $\chi[a \mapsto \sigma]$ is the assignment of domain $\text{dom}(\chi) \cup \{a\}$ that maps a to σ and is equal to χ on the rest of its domain, and $\chi[s \mapsto \sigma]$ is defined similarly, where s is a variable; also, $\chi[a \mapsto ?]$ is the assignment of domain $\text{dom}(\chi) \setminus \{a\}$, on which it is equal to χ .

Outcomes. For assignment χ and finite play ρ , $\text{Out}(\chi, \rho)$ is the set of infinite plays that start with ρ and are then extended by letting players follow the strategies assigned by χ . Formally, $\text{Out}(\chi, \rho)$ is the set of plays $\rho \cdot v_1 v_2 \dots$ such that for all $i \geq 0$, there exists \mathbf{c} such that for all $a \in \text{dom}(\chi) \cap \text{Ag}$, $\mathbf{c}_a \in \chi(a)(\rho \cdot v_1 \dots v_i)$ and $v_{i+1} = \Delta(v_i, \mathbf{c})$, with $v_0 = \text{last}(\rho)$.

2.2 BSL syntax

The core of Branching-time Strategy Logic, on which we build Prompt Strategy Logic and Bounding-outcome Strategy Logic, is the full branching-time temporal logic CTL*. This differs from usual variants of Strategy Logic which are based on the linear-time temporal logic LTL. The main difference is the introduction of an *outcome quantifier* which quantifies on outcomes of the currently fixed strategies. While in SL temporal operators could only be evaluated in contexts where all agents were assigned a strategy, this outcome quantifier allows for evaluation of (branching-time) temporal properties on partial assignments of strategies to agents. We recall Branching-time Strategy Logic, introduced in [26], which has the same expressive power as SL but allows to express branching-time properties without resorting to computationally expensive strategy quantifications.

At the syntax level, in addition to usual boolean connectives and temporal operators, we have four constructs:

- strategy quantification: $\exists s\varphi$, which means “there exists a strategy s such that φ holds”,
- assigning a strategy to a player: $(a, s)\varphi$, which is interpreted as “when the agent a plays according to s , φ holds”,
- unbinding a player: $(a, ?)\varphi$, which is interpreted as “ φ holds after agent a has been unbound from her strategy, if any”, and
- quantifying over outcomes: $\mathbf{A}\psi$, which reads as “ ψ holds in all outcomes of the strategies currently assigned to agents”.

The difference between BSL and SL lies in the last two constructs. Note that unbinding agents was irrelevant in linear-time SL, where assignments need to be total to evaluate temporal properties.

► **Definition 2** (BSL syntax). The set of BSL formulas is the set of state formulas given by the following grammar:

$$\begin{aligned} \text{State formulas:} \quad \varphi &::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists s\varphi \mid (a, s)\varphi \mid (a, ?)\varphi \mid \mathbf{A}\psi \\ \text{Path formulas:} \quad \psi &::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi \mathbf{U}\psi, \end{aligned}$$

where $p \in \text{AP}$, $a \in \text{Ag}$ and $s \in \text{Var}$.

We use classic abbreviations $\top = p \vee \neg p$, $\mathbf{F}\psi = \top \mathbf{U}\psi$, $\mathbf{G}\psi = \neg \mathbf{F}\neg\psi$ and $\forall s\varphi = \neg \exists s\neg\varphi$.

A variable s appears *free* in a formula φ if it appears in a binding operator (a, s) that is not in the scope of any strategy quantifier $\langle\langle s \rangle\rangle$.

2.3 BSL semantics

Given a formula $\varphi \in \text{BSL}$, an assignment is *variable-complete for φ* if its domain contains all free strategy variables of φ .

► **Definition 3** (BSL semantics). The semantics of a state formula is defined on a game \mathcal{G} , an assignment χ that is variable-complete for φ , and a finite play ρ . For a path formula

23:6 Quantifying Bounds in Strategy Logic

ψ , the finite play is replaced with an infinite play π and an index $i \in \mathbb{N}$. The definition by mutual induction is as follows:

$\mathcal{G}, \chi, \rho \models p$	if	$p \in \ell(\text{last}(\rho))$
$\mathcal{G}, \chi, \rho \models \neg\varphi$	if	$\mathcal{G}, \chi, \rho \not\models \varphi$
$\mathcal{G}, \chi, \rho \models \varphi \vee \varphi'$	if	$\mathcal{G}, \chi, \rho \models \varphi$ or $\mathcal{G}, \chi, \rho \models \varphi'$
$\mathcal{G}, \chi, \rho \models \exists s\varphi$	if	there exists $\sigma \in \text{Strat}$ s.t. $\mathcal{G}, \chi[s \mapsto \sigma], \rho \models \varphi$
$\mathcal{G}, \chi, \rho \models (a, s)\varphi$	if	$\mathcal{G}, \chi[a \mapsto \chi(s)], \rho \models \varphi$
$\mathcal{G}, \chi, \rho \models (a, ?)\varphi$	if	$\mathcal{G}, \chi[a \mapsto ?], \rho \models \varphi$
$\mathcal{G}, \chi, \rho \models \mathbf{A}\psi$	if	for all $\pi \in \text{Out}(\chi, \rho)$, $\mathcal{G}, \chi, \pi, \rho - 1 \models \psi$
$\mathcal{G}, \chi, \pi, i \models \varphi$	if	$\mathcal{G}, \chi, \pi_{\leq i} \models \varphi$
$\mathcal{G}, \chi, \pi, i \models \neg\psi$	if	$\mathcal{G}, \chi, \pi, i \not\models \psi$
$\mathcal{G}, \chi, \pi, i \models \psi \vee \psi'$	if	$\mathcal{G}, \chi, \pi, i \models \psi$ or $\mathcal{G}, \chi, \pi, i \models \psi'$
$\mathcal{G}, \chi, \pi, i \models \mathbf{X}\psi$	if	$\mathcal{G}, \chi, \pi, i + 1 \models \psi$
$\mathcal{G}, \chi, \pi, i \models \psi \mathbf{U} \psi'$	if	$\exists j \geq i$ s.t. $\mathcal{G}, \chi, \pi, j \models \psi'$ and $\forall k$ s.t. $i \leq k < j$, $\mathcal{G}, \chi, \pi, k \models \psi$

BSL has the same expressivity as SL, and there are linear translations in both directions [26]. More precisely, the translation from BSL to SL is linear in the size of the formula times the number of players; indeed, the outcome quantifier is simulated in SL by a strategy quantification and a binding for each player who is not currently bound to a strategy. This translation may thus increase the nesting and alternation depth of strategy quantifiers in the formula, which is known to increase the complexity of the model-checking problem [13, 36].

3 Prompt Strategy Logic

In this section we introduce PROMPT-SL, an extension of both BSL and PROMPT-LTL.

3.1 Prompt-SL syntax

The syntax of PROMPT-SL extends that of branching-time strategy logic BSL with two additional constructs, where N is a variable over natural numbers:

- a bounded version of the classical “eventually” operator written $\mathbf{F}^{\leq N}$, and
- an existential quantification on the values of variable N , written $\exists N$.

As in PROMPT-LTL, the formula $\mathbf{F}^{\leq N}\psi$ states that ψ will hold at the latest within N steps from the present. For a formula φ of PROMPT-SL there is a unique *bound variable* N : indeed, in the spirit of PROMPT-LTL where a unique bound must exist for all prompt- eventualities, formulas of our logic cannot use more than one bound variable. However, in PROMPT-SL, existential quantification on N is part of the syntax, which allows to freely combine quantification on the (unique) bound variable N with other operators of the logic. In particular one can express the existence of a unique bound that should work for all strategies, or instead that the bound may depend on the strategy (see Example 6).

► **Definition 4** (PROMPT-SL syntax). The syntax of PROMPT-SL formulas is defined by the following grammar:

State formulas:	$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists s\varphi \mid (a, s)\varphi \mid (a, ?)\varphi \mid \mathbf{A}\psi \mid \exists N\varphi$
Path formulas:	$\psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi \mathbf{U} \psi \mid \mathbf{F}^{\leq N}\psi$

where $p \in \text{AP}$, $s \in \text{Var}$, $a \in \text{Ag}$ and N is a fixed bounding variable. A PROMPT-SL *sentence* is a state formula with no free strategy variable, in which every $\mathbf{F}^{\leq N}$ is in the scope of some $\exists N$, and $\mathbf{F}^{\leq N}$ and $\exists N$ always appear positively, i.e. under an even number of negations.

3.2 Prompt-SL semantics

We now define the semantics of PROMPT-SL.

► **Definition 5** (PROMPT-SL semantics). The semantics is defined inductively as follows, where φ (resp. ψ) is a cost-SL state (resp. path) formula, \mathcal{G} is a game, χ is an assignment variable-complete for φ (resp. ψ), ρ is a finite play, π an infinite one, $i \in \mathbb{N}$ is a point in time and $n \in \mathbb{N}$ is a bound.

$\mathcal{G}, \chi, \rho, n \models p$	if	$p \in \ell(\text{last}(\rho))$
$\mathcal{G}, \chi, \rho, n \models \neg\varphi$	if	$\mathcal{G}, \chi, \rho, n \not\models \varphi$
$\mathcal{G}, \chi, \rho, n \models \varphi \vee \varphi'$	if	$\mathcal{G}, \chi, \rho, n \models \varphi$ or $\mathcal{G}, \chi, \rho, n \models \varphi'$
$\mathcal{G}, \chi, \rho, n \models \exists s\varphi$	if	there exists $\sigma \in \text{Strat}$ s.t. $\mathcal{G}, \chi[s \mapsto \sigma], \rho, n \models \varphi$
$\mathcal{G}, \chi, \rho, n \models (a, s)\varphi$	if	$\mathcal{G}, \chi[a \mapsto \chi(s)], \rho, n \models \varphi$
$\mathcal{G}, \chi, \rho, n \models (a, ?)\varphi$	if	$\mathcal{G}, \chi[a \mapsto ?], \rho, n \models \varphi$
$\mathcal{G}, \chi, \rho, n \models \mathbf{A}\psi$	if	for all $\pi \in \text{Out}(\chi, \rho)$, $\mathcal{G}, \chi, \pi, \rho - 1, n \models \psi$
$\mathcal{G}, \chi, \rho, n \models \exists N\varphi$	if	there exists $n' \in \mathbb{N}$ such that $\mathcal{G}, \chi, \rho, n' \models \varphi$
$\mathcal{G}, \chi, \pi, i, n \models \varphi$	if	$\mathcal{G}, \chi, \pi_{\leq i}, n \models \varphi$
$\mathcal{G}, \chi, \pi, i, n \models \neg\psi$	if	$\mathcal{G}, \chi, \pi, i, n \not\models \psi$
$\mathcal{G}, \chi, \pi, i, n \models \psi \vee \psi'$	if	$\mathcal{G}, \chi, \pi, i, n \models \psi$ or $\mathcal{G}, \chi, \pi, i, n \models \psi'$
$\mathcal{G}, \chi, \pi, i, n \models \mathbf{X}\psi$	if	$\mathcal{G}, \chi, \pi, i + 1, n \models \psi$
$\mathcal{G}, \chi, \pi, i, n \models \psi \mathbf{U}\psi'$	if	$\exists j \geq i$ s.t. $\mathcal{G}, \chi, \pi, j, n \models \psi'$ and $\forall k$ s.t. $i \leq k < j$, $\mathcal{G}, \chi, \pi, k, n \models \psi$
$\mathcal{G}, \chi, \pi, i, n \models \mathbf{F}^{\leq N}\psi$	if	there exists $j \in [i, n]$ such that $\mathcal{G}, \chi, \pi, j, n \models \psi$.

The semantics of a sentence Φ does not depend on the bound n , and we may write $\mathcal{G}, \chi, \rho \models \Phi$ if $\mathcal{G}, \chi, \rho, n \models \Phi$ for some n . In addition a sentence does not require an assignment for its evaluation. Given a game \mathcal{G} with initial vertex v_0 and a sentence Φ , we write $\mathcal{G} \models \Phi$ if $\mathcal{G}, \emptyset, v_0 \models \Phi$, where \emptyset is the empty assignment.

► **Example 6.** In bounded parity games [12, 37] the odd colours represent requests and even colours represent grants, and the objective of the player a_1 is to ensure against player a_2 that every request is promptly followed by a larger grant. Solving such games can be cast as a model-checking problem of the PROMPT-SL formula

$$\exists s_1(a_1, s_1)\forall s_2(a_2, s_2)\exists N\mathbf{AG} \left[\bigwedge_{c \text{ odd}} c \rightarrow \mathbf{F}^{\leq N} \bigvee_{d > c \text{ even}} d \right]$$

on the structure in which every vertex is labelled by its color. The finitary parity condition relaxes the constraint by only requiring requests that appear infinitely often to be promptly granted, and solving such games can be reduced to model checking the PROMPT-SL formula

$$\exists s_1(a_1, s_1)\forall s_2(a_2, s_2)\exists N\mathbf{AG} \left[\bigwedge_{c \text{ odd}} (c \wedge \mathbf{GF}c) \rightarrow \mathbf{F}^{\leq N} \bigvee_{d > c \text{ even}} d \right].$$

Observe that in both these definitions, the bound on the delay between requests and grants can depend on the outcome, i.e. on the opponent's strategy. We can also express *uniform* variants of these objectives by moving the quantification on the bound $\exists N$ before the quantification on opponent's strategies $\forall s_2$. Such games are studied in the context of the theory of regular cost functions [14, 16, 15], and their relationship to the non-uniform variants has been investigated in [11]. The solution to the model-checking problem for PROMPT-SL that we present here allows us to solve both types of games, uniform and non-uniform.

4 Bounding-outcomes Strategy Logic

We now define our second quantitative extension of Strategy Logic, which we call Bounding-outcomes Strategy Logic, or BOSL.

4.1 BOSL syntax

The syntax of BOSL extends that of strategy logic BSL with two additional constructs:

- a bounded version of the outcome quantifier written $\mathbf{A}^{\leq N}$,
- an existential quantification on the values of variable N , written $\exists N$.

BOSL can also be seen as PROMPT-SL without the bounded eventually $\mathbf{F}^{\leq N}$ but with the novel bounded outcome quantifier $\mathbf{A}^{\leq N}$. While formula $\mathbf{A}\psi$ states that ψ holds in all outcomes of the current assignment, $\mathbf{A}^{\leq N}\psi$ states that ψ holds in all of these outcomes *except for at most N of them*.

► **Definition 7** (BOSL syntax). The syntax of BOSL formulas is given by the following grammar:

$$\begin{array}{ll} \text{State formulas:} & \varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists s\varphi \mid (a, s)\varphi \mid (a, ?)\varphi \mid \mathbf{A}\psi \mid \mathbf{A}^{\leq N}\psi \mid \exists N\varphi \\ \text{Path formulas:} & \psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi\mathbf{U}\psi \end{array}$$

where $p \in \text{AP}$, $s \in \text{Var}$, $a \in \text{Ag}$ and N is a fixed bounding variable. A BOSL *sentence* is a state formula with no free strategy variable, in which every $\mathbf{A}^{\leq N}$ is in the scope of some $\exists N$, and where $\mathbf{A}^{\leq N}$ and $\exists N$ always appear positively, i.e. under an even number of negations.

4.2 BOSL semantics

► **Definition 8** (BOSL semantics). We only give the definition for the new operator $\mathbf{A}^{\leq N}$, the others are as in Definition 5.

$$\mathcal{G}, \chi, \rho, n \models \mathbf{A}^{\leq N}\psi \quad \text{if} \quad \text{Card}(\{\pi \in \text{Out}(\rho, \chi) : \mathcal{G}, \chi, \pi, |\rho| - 1, n \not\models \psi\}) \leq n$$

The full semantics can be found in Appendix A.1. Once again, for a sentence Φ we write $\mathcal{G} \models \Phi$ if $\mathcal{G}, \emptyset, v_0, n \models \Phi$ for some $n \in \mathbb{N}$, where \emptyset is the empty assignment.

► **Example 9.** As an example we consider the framework of Carayol and Serre [10] that considers games with two players and a third player called “nature”. The usual semantics is for nature to be a random player, in which case we are interested in whether player 1 has a strategy ensuring to win almost all paths. The paper [10] suggests other formalisations for the third player, of topological, measure-theoretic, and combinatorial nature, and provides general reductions. For instance, one may fix a constant N and write the following formula $\exists s_1(a_1, s_1)\forall s_2(a_2, s_2)\mathbf{A}^{\leq N}\psi$, stating that player a_1 has a strategy ensuring to win all but N paths. If N is a constant the above question is solved in [10]. However the latter work leaves open the question of ensuring that player a_1 wins all but a bounded number of paths, which is expressible by the Bounding-outcome Strategy Logic formula $\exists N\exists s_1(a_1, s_1)\forall s_2(a_2, s_2)\mathbf{A}^{\leq N}\psi$. One could also consider the variant where the bound can depend on the opponent’s strategy, which can be expressed by the formula $\exists s_1(a_1, s_1)\forall s_2\exists N(a_2, s_2)\mathbf{A}^{\leq N}\psi$. In this paper we show that the model-checking problem for Bounding-outcome Strategy Logic is decidable, thereby giving a solution to both variants of this question.

5 Model checking

In this section we solve the model-checking problem for both PROMPT-SL and BOSL with a uniform approach which, in fact, works also for the combination of the two logics. As done in [35] for ATL with strategy context, in [6] for an extension of it with imperfect information and in [7] for Strategy Logic with imperfect information, we go through an adequate extension of QCTL*, which itself extends CTL* with second-order quantification. This approach makes automata constructions and their proof of correctness easier and clearer. In our case we define an extension of QCTL* called BOUND-QCTL*, which contains the bounded eventually $\mathbf{F}^{\leq N}$ from PROMPT-LTL and PROMPT-SL, a bounded path quantifier $\mathbf{A}^{\leq N}$ similar to the bounded outcome quantifier from BOSL, and the quantifier on bounds $\exists N$ present in both PROMPT-SL and BOSL. We then recall definitions and results about cost automata, that we use to solve the model-checking problem for BOUND-QCTL*. We finally solve the model-checking problem for both PROMPT-SL and BOSL by reducing them to model checking BOUND-QCTL*.

5.1 Bound Quantified CTL*

In this section we define Bound Quantified CTL*, or BOUND-QCTL*, which extends PROMPT-LTL to the branching-time setting and adds quantification on atomic propositions. One can also see it as an extension of Quantified CTL* [40, 29, 30, 23, 34] with the bounded eventually operator and a bounded version of the universal path quantifier. Unlike PROMPT-LTL, but similarly to our PROMPT-SL and BOSL, an existential quantification on the bound for the bounded eventually and bounded outcome quantifier is also part of the syntax.

5.1.1 Bound-QCTL* syntax

► **Definition 10.** The syntax of BOUND-QCTL* is defined by the following grammar:

$$\begin{aligned} \varphi &= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{A}\psi \mid \mathbf{A}^{\leq N}\psi \mid \exists p\varphi \mid \exists N\varphi \\ \psi &= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi\mathbf{U}\psi \mid \mathbf{F}^{\leq N}\psi \end{aligned}$$

where $p \in \text{AP}$, and N is a fixed bounding variable.

As usual, formulas of type φ are called *state formulas*, those of type ψ are called *path formulas*, and QCTL* consists of all the state formulas defined by the grammar. We further distinguish between *positive formulas*, in which operators $\mathbf{F}^{\leq N}$, $\mathbf{A}^{\leq N}$ and $\exists N$ appear only positively (under an even number of negations), and *negative formulas*, in which operators $\mathbf{F}^{\leq N}$, $\mathbf{A}^{\leq N}$ and $\exists N$ appear only negatively (under an odd number of negations). A BOUND-QCTL* *sentence* is a positive formula such that all operators $\mathbf{F}^{\leq N}$ and $\mathbf{A}^{\leq N}$ in the formula are in the scope of some $\exists N$. Note that we will be interested in model checking sentences, and every subformula of a sentence is either positive or negative.

5.1.2 Bound-QCTL* semantics

BOUND-QCTL* formulas are evaluated on (unfoldings of) Kripke structures.

► **Definition 11.** A (finite) *Kripke structure* is a tuple $\mathcal{S} = (S, s_0, R, \ell)$, where S is a finite set of *states*, $s_0 \in S$ is an *initial state*, $R \subseteq S \times S$ is a left-total *transition relation*³, and $\ell : S \rightarrow \Sigma$ is a *labelling function*.

³ *i.e.*, for all $s \in S$, there exists s' such that $(s, s') \in R$.

23:10 Quantifying Bounds in Strategy Logic

A *path* in \mathcal{S} is a finite word λ over S such that for all i , $(\lambda_i, \lambda_{i+1}) \in R$. For $s \in S$, we let $\text{Paths}(s) \subseteq S^+$ be the set of all paths that start in s .

Trees. Let S be a finite set of *directions* and Σ a set of *labels*. A (Σ, S) -*tree* (or simply *tree*) is a pair $t = (\tau, \ell)$ where $\ell : \tau \rightarrow \Sigma$ is a *labelling* and $\tau \subseteq S^+$ is the *domain* such that:

- there exists $r \in S^+$, called the *root* of τ , such that each $u \in \tau$ starts with r , i.e. $r \preceq u$,
- if $u \cdot s \in \tau$ and $u \cdot s \neq r$, then $u \in \tau$,
- if $u \in \tau$ then there exists $s \in S$ such that $u \cdot s \in \tau$.

The elements of τ are called *nodes*. If $u \cdot s \in \tau$, we say that $u \cdot s$ is a *child* of u . A *branch* λ in t is an infinite sequence of nodes such that $\lambda_0 \in \tau$ and for all i , λ_{i+1} is a child of λ_i , and $\text{Branches}(t, u)$ is the set of branches that start in node u . We let $\text{Branches}(t)$ denote the set of branches that start in the root. If S is a singleton, a tree becomes an infinite word.

► **Definition 12.** The *tree unfolding* of a Kripke structure \mathcal{S} from state s is the tree $t_{\mathcal{S}}(s) = (\text{Paths}(s), \ell')$, where for every $u \in \text{Paths}(s)$, we have $\ell'(u) = \ell(\text{last}(u))$. We may write $t_{\mathcal{S}}$ for $t_{\mathcal{S}}(s_0)$, the unfolding from the initial state.

Projection and subtrees. Given two trees t, t' and a proposition p , we write $t \equiv_p t'$ if they have same domain τ and for all p' in AP such that $p' \neq p$, for all u in τ , we have $p' \in \ell(u)$ if, and only if, $p' \in \ell'(u)$. Given a tree $t = (\tau, \ell)$ and a node $u \in \tau$, we define the *subtree of t rooted in u* as the tree $t_u = (\tau_u, \ell')$ where $\tau_u = \{v \in S^+ : u \preceq v\}$ and ℓ' is ℓ restricted to τ_u .

► **Definition 13.** The semantics $t, u, n \models \varphi$ and $t, \lambda, n \models \psi$ are defined inductively, where φ is a BOUND-QCTL* state formula, ψ is a BOUND-QCTL* path formula, $t = (\tau, \ell)$ is a tree, u is a node, λ is a branch in t , and n in \mathbb{N} a bound (the inductive cases for classic CTL* operators can be found in Appendix A.2):

$$\begin{aligned} t, u, n \models \mathbf{A}^{\leq N} \psi & \quad \text{if} & \quad \text{Card}(\{\lambda \in \text{Branches}(t, u) : t, \lambda, n \not\models \psi\}) \leq n \\ t, u, n \models \exists p \varphi & \quad \text{if} & \quad \exists t' \equiv_p t \text{ such that } t', u, n \models \varphi \\ t, u, n \models \exists N \varphi & \quad \text{if} & \quad \exists n' \in \mathbb{N} \text{ such that } t, u, n' \models \varphi, \\ t, \lambda, n \models \mathbf{F}^{\leq N} \psi & \quad \text{if} & \quad \exists j \text{ such that } 0 \leq j \leq n \text{ and } t, \lambda_{\geq j}, n \models \psi \end{aligned}$$

The *value* $\llbracket \varphi \rrbracket_{\text{inf}}(t)$ (resp. $\llbracket \varphi \rrbracket_{\text{sup}}(t)$) of a positive (resp. negative) state formula φ on a tree t with root r is defined as

$$\llbracket \varphi \rrbracket_{\text{inf}}(t) = \inf \{n \in \mathbb{N} : t, r, n \models \varphi\} \quad \text{and} \quad \llbracket \varphi \rrbracket_{\text{sup}}(t) = \sup \{n \in \mathbb{N} : t, r, n \models \varphi\},$$

with the usual convention that $\inf \emptyset = \infty$ and $\sup \emptyset = 0$. In case it is not a positive or negative formula, its value is undefined. We remark that $\{n \in \mathbb{N} : t, r, n \models \varphi\}$ is downward (resp. upward) closed if φ is negative (resp. positive). The value of a sentence Φ is always either 0 or ∞ (recall that sentences are necessarily positive formulas and N is always quantified), and given a Kripke structure \mathcal{S} , we write $\mathcal{S} \models \Phi$ if $\llbracket \Phi \rrbracket_{\text{inf}}(t_{\mathcal{S}}) = 0$.

5.2 Regular cost functions

In this section we develop the theory of regular cost functions over trees for distance automata. To this end we define and study the two dual models of **distance** and **distance**-automata for recognising cost functions [14], referred to as cost automata.

Let E be a set of structures (such as infinite words or trees). We define an equivalence relation \approx on functions $E \rightarrow \mathbb{N} \cup \{\infty\}$ by $f \approx g$ if for all $X \subseteq E$, $f(X)$ is bounded if, and only if, $g(X)$ is bounded. A *cost function* over E is an equivalence class of the relation \approx .

In Section 5.2.1 we define cost games whose objectives may refer to a single counter that, in each step, can be incremented or left unchanged. In Section 5.2.2 we define automata whose semantics are given using cost games. We introduce **distance**-automata and their duals **distance**-automata that compute functions $E \rightarrow \mathbb{N} \cup \{\infty\}$. In Section 5.2.3 we focus on automata over infinite words and the notion of history-deterministic automata.

The novel technical contribution of this section is an extension of the classical property of history-deterministic automata: the original result says that given a history-deterministic automaton over infinite words, one can simulate it along every branch of a tree. This is the key argument to handle the **A** operator in PROMPT-SL. In Section 5.2.4 we extend this result by allowing the automaton to skip a bounded number of paths, which will allow us to capture the bounded-outcome operator $\mathbf{A}^{\leq N}$ in BOSL.

5.2.1 Cost games

The semantics of cost automata are given by turn-based two-player games, which are essentially a special case of the general notion of games given in Section 3.2. We give here a slightly modified definition better fitting the technical developments.

► **Definition 14.** A *game* is given by $G = (V, V_E, V_A, v_0, E, c)$, where $V = V_E \uplus V_A$ is a set of *vertices* divided into the vertices V_E controlled by Eve and the vertices V_A controlled by Adam, $v_0 \in V$ is an *initial vertex*, $E \subseteq V \times V$ is a left-total *transition relation*, $c : V \rightarrow \Omega$ is a labelling function.

A *finite* (resp. *infinite*) *play* is a finite (resp. infinite) word $\rho = v_0 \dots v_n$ (resp. $\pi = v_0 v_1 \dots$) such that for every i such that $0 \leq i < |\rho| - 1$ (resp. $i \geq 0$), $(v_i, v_{i+1}) \in E$. A *strategy* for Eve (resp. for Adam) is a function $\sigma : V^* \cdot V_E \rightarrow V$ (resp. $\sigma : V^* \cdot V_A \rightarrow V$) such that for all finite play $\rho \in V^* \cdot V_E$ (resp. $\rho \in V^* \cdot V_A$), we have $(\text{last}(\rho), \sigma(\rho)) \in E$. Given a strategy σ for Eve and σ' for Adam, we let $\text{Outcome}(\sigma, \sigma')$ be the unique infinite play that starts in v_0 and is consistent with σ and σ' .

An *objective* is a set $W \subseteq \Omega^\omega$. To make the objective explicit we speak of W -games, which are games with objective W . A strategy σ for Eve *ensures* $W \subseteq \Omega^\omega$ if for all strategy σ' of Adam, the infinite word obtained by applying c to each position of the play $\text{Outcome}(\sigma, \sigma')$ is in W . Eve *wins* the W -game G if there exists a strategy for her that ensures W . The same notions apply to Adam. We now introduce the objectives we will be using.

- Given $d \in \mathbb{N}^*$, the *parity objective* $\mathbf{parity} \subseteq \{1, \dots, d\}^\omega$ is the set of infinite words in which the maximum label appearing infinitely many times is even.
- The *distance objective* uses the set of labels $\{\epsilon, \mathbf{i}\}$ acting on a counter taking values in the natural numbers and initialised to 0. The labels ϵ and \mathbf{i} are seen as actions on the counter: the action ϵ leaves the counter unchanged and \mathbf{i} increments the counter by 1. For $n \in \mathbb{N}$, the distance objective $\mathbf{distance}(n) \subseteq \{\epsilon, \mathbf{i}\}^\omega$ is the set of infinite words such that the counter is bounded by n .
- The *regular distance objective* $\mathbf{fininc} \subseteq \{\epsilon, \mathbf{i}\}^\omega$ is the set of infinite words such that the counter is incremented finitely many times.
- The *co-distance objective* uses set of labels $\{\epsilon, \mathbf{i}\}$, where ϵ and \mathbf{i} have the same interpretation as in $\mathbf{distance}(n)$. For $n \in \mathbb{N}$, the objective $\overline{\mathbf{distance}}(n) \subseteq \{\epsilon, \mathbf{i}\}^\omega$ is the set of infinite words such that the counter eventually reaches value n .
- The objectives can be combined: $\mathbf{parity} \cap \mathbf{distance}(n) \subseteq (\{1, \dots, d\} \times \{\epsilon, \mathbf{i}\})^\omega$ is the Cartesian product of the parity and the distance objective (where a pair of infinite words is assimilated with the infinite word formed of the pairs of letters at same position).

The following result, proven in [11], relates **distance** and **fininc** in the context of games.

► **Lemma 15.** *Let G be a finite game. There exists $n \in \mathbb{N}$ such that Eve wins for **parity** \cap **distance**(n) iff Eve wins for **parity** \cap **fininc**.*

5.2.2 Cost automata

We now define automata over (Σ, S) -trees.

► **Definition 16.** A (non-deterministic) *automaton* is a tuple $\mathcal{A} = (Q, q_0, \delta, c)$ where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta \subseteq Q \times \Sigma \times Q^S$ is a transition relation, and $c : Q \rightarrow \Omega$ is a labelling function.

When an automaton is equipped with an objective $W \subseteq \Omega^\omega$ we speak of an W -automaton. To define the semantics of W -automata, we define acceptance games. Given an W -automaton \mathcal{A} and a (Σ, S) -tree $t = (\tau, \ell)$, we define the acceptance W -game $G_{\mathcal{A}, t}$ as follows.

- The set of vertices is $(Q \times \tau) \cup (Q \times \tau \times Q^S)$. The vertices of the form $Q \times \tau$ are controlled by Eve, the others by Adam.
- The initial vertex is (q_0, r) , where r is the root of t .
- The transition relation relates the vertex (q, u) to (q, u, h) if $(q, \ell(u), h) \in \delta$, and (q, u, h) is related to $(h(s), u \cdot s)$ for every $s \in S$.
- The label of a vertex (q, u) is $c(q)$, and the other vertices are not labelled.

We say that t is accepted by \mathcal{A} if Eve wins the acceptance W -game $G_{\mathcal{A}, t}$.

An equivalent point of view is to say that t is accepted by \mathcal{A} if there exists a (Q, S) -tree with same domain as t respecting the transition relation δ with respect to t , such that all branches satisfy W .

We instantiate this definition for cost automata: the objective **parity** \cap **distance** gives rise to the notion of **distance**-automata. A **distance**-automaton \mathcal{A} *computes* the function $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}$ over trees defined by

$$\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(t) = \inf \{n \in \mathbb{N} : t \text{ is accepted by } \mathcal{A} \text{ with objective } \mathbf{parity} \cap \mathbf{distance}(n)\},$$

and it *recognises* the \approx -equivalence class of the function $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}$.

Dually, the objective **parity** \cap $\overline{\mathbf{distance}}$ (n) gives rise to $\overline{\mathbf{distance}}$ -automata. A $\overline{\mathbf{distance}}$ -automaton \mathcal{A} *computes* the function $\llbracket \mathcal{A} \rrbracket_{\overline{\mathbf{d}}}$ over trees defined by

$$\llbracket \mathcal{A} \rrbracket_{\overline{\mathbf{d}}}(t) = \sup \{n \in \mathbb{N} : t \text{ is accepted by } \mathcal{A} \text{ with objective } \mathbf{parity} \cap \overline{\mathbf{distance}}(n)\}$$

and *recognises* the \approx -equivalence class of the function $\llbracket \mathcal{A} \rrbracket_{\overline{\mathbf{d}}}$.

If \mathcal{A} recognises the \approx -equivalence class of the function $f : E \rightarrow (\mathbb{N} \cup \{\infty\})$ we abuse notation and say that \mathcal{A} *recognises the function* f .

To illustrate the definition of **distance**-automata, we now give an example that will be useful later on to capture the bounded path quantifier $\mathbf{A}^{\leq N}$.

► **Lemma 17.** *Let $p \in AP$. There exists a **distance**-automaton recognising the function that counts the number of paths with infinitely many p 's.*

Proof. Let us say that a path is bad if it contains infinitely many p . The **distance**-automaton \mathcal{A} has four states:

- $q_{0,\epsilon}$, whose intuitive semantics is “the tree contains one bad path”,
- $q_{0,i}$, meaning “the tree contains at least two bad paths”,
- $q_{1,p}$ and $q_{1,\neg p}$, which mean “the tree does not contain any bad path”.

All states are initial (note that this is an inconsequential abuse because we defined automata with a single initial state). We use the set of labels $\Omega = \{2, 3\} \times \{\epsilon, i\}$. The transitions are as follows, where $q_0 = \{q_{0,\epsilon}, q_{0,i}\}$ and $q_1 = \{q_{1,p}, q_{1,\neg p}\}$.

$$\delta = \begin{cases} (q_{0,\epsilon}, a, h) & \text{if } h \text{ contains at most one } q_0 \\ (q_{0,i}, a, h) & \text{if } h \text{ contains at least two } q_0 \\ (q_{1,\neg p}, a, h) & \text{if } p \notin a \text{ and } h \text{ contains only } q_1 \\ (q_{1,p}, a, h) & \text{if } p \in a \text{ and } h \text{ contains only } q_1 \end{cases}$$

The labelling function is $c(q_{0,\epsilon}) = (2, \epsilon)$, $c(q_{0,i}) = (2, i)$, $c(q_{1,\neg p}) = (2, \epsilon)$, and $c(q_{1,p}) = (3, \epsilon)$. We claim that the following two properties hold, which implies Lemma 17.

- if t contains n bad paths, then $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(t) \leq n - 1$,
- if $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(t) \leq n$, then t contains at most $\text{Card}(S)^n$ bad paths.

Assume that t contains n bad paths, we construct a run for \mathcal{A} (i.e., a labelling of t with states of \mathcal{A}) as follows. A node u of the tree is labelled by:

- $q_{0,\epsilon}$ if exactly one $t_{u,s}$ contains a bad path for some direction $s \in S$,
- $q_{0,i}$ if $t_{u,s}$ contain a bad path for at least two different directions $s \in S$,
- $q_{1,\neg p}$ if t_u does not contain a bad path and $p \notin \ell(u)$,
- $q_{1,p}$ if t_u does not contain a bad path and $p \in \ell(u)$.

This yields a valid run whose branches all satisfy the parity condition. Along a branch the counter is incremented each time there are at least two subtrees with a bad path, which can happen at most $n - 1$ times because there are n bad paths. Hence the maximal value of the counter on a branch is $n - 1$, implying that $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(t) \leq n - 1$.

We show the second point by induction on n . If $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(t) = 0$, then t contains at most one bad path. If $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(t) = n + 1$, consider a (Q, S) -tree representing a run of value $n + 1$. Because $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(t) \geq 1$, there is at least one node labelled $q_{0,i}$. By definition of the transition relation, if there are two nodes on the same level labelled $q_{0,i}$, then they must descend from another node $q_{0,i}$ higher in the tree. Thus there is a unique node u labelled $q_{0,i}$ that is closest to the root (it may be the root itself). Except for u 's ancestors, which are labelled with $q_{0,\epsilon}$, all nodes outside of the subtree rooted in u are necessarily labelled with q_1 . The subtrees rooted in u 's children have a run with value at most n . By induction hypothesis each of these subtrees contains at most $\text{Card}(S)^n$ bad paths, so the tree rooted in u contains at most $\text{Card}(S)^{n+1}$ bad paths. Since nodes labelled by q_1 cannot contain a bad path, this means that t contains at most $\text{Card}(S)^{n+1}$ bad paths. ◀

The objective **parity** gives rise to parity automata. The following lemma follows from the observation that **fininc** is an ω -regular objective.

► **Lemma 18.** *For every automaton with objective $\text{parity} \cap \text{fininc}$ one can construct an equivalent parity automaton.*

5.2.3 Regular cost functions over words

The definitions of cost-automata can be applied to infinite words, which is the particular case where S is a singleton. A central notion in the theory of regular cost functions is that of history-deterministic automata over infinite words. Informally, a non-deterministic automaton is history-deterministic if its non-determinism can be resolved by a function considering only the input read so far. This notion has been introduced for studying ω -automata in [25]. We specialise it here to the case of cost functions, involving a relaxation on the values allowing for a good interplay with the definition of equivalence for cost functions.

To give a formal definition we introduce the notation \mathcal{A}_σ for \mathcal{A} a W -automaton and a strategy $\sigma : \Sigma^* \rightarrow \delta$, where δ is the transition relation of \mathcal{A} : \mathcal{A}_σ is a (potentially infinite) deterministic W -automaton $(Q \times \Sigma^*, (q_0, \varepsilon), \delta_\sigma, c_\sigma)$ where $((q, w), a, (q', wa)) \in \delta_\sigma$ just if $\sigma(w) = (q, a, q')$, and $c_\sigma(q, w) = c(q)$. The automaton \mathcal{A}_σ is infinite but deterministic, as for each situation the strategy σ chooses the transition to follow.

► **Definition 19** ([14, 17]). We say that a **distance-automaton** \mathcal{A} over infinite words is *history-deterministic* if there exists a function $\alpha : \mathbb{N} \rightarrow \mathbb{N}$ such that for every n there exists a strategy σ such that for all words w we have $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(w) \leq n \implies \llbracket \mathcal{A}_\sigma \rrbracket_{\mathbf{d}}(w) \leq \alpha(n)$.

We now explain the usefulness of the notion of history-deterministic automata. The situation is the following: we consider a language L over infinite words, and we want to construct an automaton for the language of trees “all branches are in L ”. Given a deterministic automaton for L one can easily solve this problem by constructing an automaton running the deterministic automaton on all branches.

In the quantitative setting we consider here, we have a function $f : \Sigma^\omega \rightarrow \mathbb{N} \cup \{\infty\}$ instead of L , and we wish to construct an automaton computing the function over trees $t \mapsto \sup \{f(\lambda) : \lambda \in \text{Branches}(t)\}$. Unfortunately, **distance-automata** do not determinise, so the previous approach needs to be refined. The construction fails for non-deterministic automata, because two branches may have very different accepting runs even on their shared prefix. The notion of history-deterministic automata yields a solution to this problem, as stated in the following theorem.

► **Theorem 20** ([18]). *Let \mathcal{A} be a history-deterministic **distance-automaton** over infinite words. One can construct a **distance-automaton** recognising the function over trees*

$$t \mapsto \sup \{ \llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(\lambda) : \lambda \in \text{Branches}(t) \}$$

We present an extension of this result where the function can remove a bounded number of paths in the computation. The proof is in Appendix A.3.

► **Theorem 21.** *Let \mathcal{A} be a history-deterministic **distance-automaton** over infinite words. One can construct a **distance-automaton** recognising the function over trees*

$$t \mapsto \inf \{ \max(\text{Card}(B), \sup \{ \llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(\lambda) : \lambda \notin B \}) : B \subseteq \text{Branches}(t) \}.$$

The idea is to combine \mathcal{A} with the automaton defined in the proof of Lemma 17.

5.2.4 Regular cost functions over trees

We introduce the notion of nested automata, which is parameterised by an objective $W \subseteq \Omega^\omega$. Nested automata can be seen as a special form of alternating automata which will be convenient to work with in the technical developments.

► **Definition 22.** A *nested W -automaton* with k slaves over (Σ, S) -trees is given by

- a *master automaton* \mathcal{A} , which is a W -automaton over $(2^k, S)$ -trees, and
- k *slave automata* $(\mathcal{A}_i)_{i \in [k]}$, which are W -automata over (Σ, S) -trees.

The transition relation of the master is $\delta \subseteq Q \times 2^k \times Q^S$. We describe the modus operandi of a nested automaton informally. Let t be a tree and u a node in t , labelled with state q . To take the next transition the master automaton interrogates its slaves: the transition $(q, v, h) \in \delta$ is allowed if for all $i \in v$, the subtree t_u is accepted by \mathcal{A}_i . The formal semantics of nested W -automata can be found in Appendix A.4.

The following theorem shows the equivalence between **distance** and **distance-automata** over trees.

► **Theorem 23** ([15]). *Let f be a cost function over trees. The following statements are effectively equivalent:*

- *there exists a **distance**-automaton recognising f ,*
- *there exists a nested **distance**-automaton recognising f ,*
- *there exists a **distance**-automaton recognising f ,*
- *there exists a nested **distance**-automaton recognising f .*

5.3 Model checking Bound-QCTL*

The *model-checking problem* for BOUND-QCTL* is the following decision problem: given an instance (Φ, \mathcal{S}) where Φ is a sentence of BOUND-QCTL* and \mathcal{S} is a Kripke structure, return ‘Yes’ if $\mathcal{S} \models \Phi$ and ‘No’ otherwise. In this section we prove that this problem is decidable by reducing it to the emptiness problem of parity automata.

We will use the following result about **distance**-automata over infinite words.

► **Theorem 24** ([27, 28]). *For every PROMPT-LTL formula ψ , we can construct a history-deterministic **distance**-automaton \mathcal{A} such that $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}} \approx \llbracket \psi \rrbracket_{\mathbf{inf}}$.*

► **Theorem 25.** *Let Φ be a sentence of BOUND-QCTL*. We construct a non-deterministic parity automaton \mathcal{A}_{Φ} over (Σ, S) -trees such that for every Kripke structure \mathcal{S} over the set of states S , we have $\mathcal{S} \models \Phi$ if, and only if, \mathcal{A}_{Φ} accepts the unfolding $t_{\mathcal{S}}$.*

Proof. Let Φ be a sentence and S a finite set of states.

For each subformula φ of Φ , we construct by induction on φ the following automata:

1. if φ is positive, a **distance**-automaton \mathcal{A}_{φ} such that $\llbracket \mathcal{A}_{\varphi} \rrbracket_{\mathbf{d}} \approx \llbracket \varphi \rrbracket_{\mathbf{inf}}$,
2. if φ is negative, a **distance**-automaton \mathcal{A}_{φ} such that $\llbracket \mathcal{A}_{\varphi} \rrbracket_{\bar{\mathbf{d}}} \approx \llbracket \varphi \rrbracket_{\mathbf{sup}}$.

We give the most interesting inductive cases, the remaining ones can be found in Appendix A.5.

- $\varphi = \mathbf{A}\psi$: The idea is similar to the automata construction for branching-time logic [33]: intuitively, treat ψ as an LTL formula over maximal state subformulas, run a deterministic automaton for ψ on all branches of the tree, and launch automata for the maximal state subformulas of ψ when needed. In our case, we will construct a nested automaton to do this, and in place of a deterministic parity automaton for ψ we will use a history-deterministic **distance**-automaton. Finally, we will convert the nested **distance**-automaton into a **distance**-automaton.

Suppose that φ is positive (the case that φ is negative is treated dually). Then also ψ is positive. We will construct a nested **distance**-automaton \mathcal{B} such that $\llbracket \mathcal{B} \rrbracket_{\mathbf{d}} \approx \llbracket \varphi \rrbracket_{\mathbf{inf}}$.

Let $\varphi_1, \dots, \varphi_k$ be the maximal state subformulas of the path formula ψ . We see these formulas as atomic propositions, so that the formula ψ can be seen as a PROMPT-LTL formula on infinite words over the alphabet 2^k . Apply Theorem 24 to ψ to get a history-deterministic **distance**-automaton \mathcal{A}_{ψ} over infinite words such that $\llbracket \mathcal{A}_{\psi} \rrbracket_{\mathbf{d}} \approx \llbracket \psi \rrbracket_{\mathbf{inf}}$. Then, apply Theorem 20 to \mathcal{A}_{ψ} to get a **distance**-automaton \mathcal{A} such that $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(t) = \sup \{ \llbracket \mathcal{A}_{\psi} \rrbracket_{\mathbf{d}}(\lambda) : \lambda \in \text{Branches}(t) \}$. The master of \mathcal{B} is \mathcal{A} .

Since ψ is positive, the formulas $\varphi_1, \dots, \varphi_k$ are either positive or negative. By the induction hypothesis, for every i , if φ_i is positive we construct a **distance**-automaton \mathcal{A}_i such that $\llbracket \mathcal{A}_i \rrbracket_{\mathbf{d}} \approx \llbracket \varphi_i \rrbracket_{\mathbf{inf}}$; and if φ_i is negative, we construct a **distance**-automaton \mathcal{A}'_i such that $\llbracket \mathcal{A}'_i \rrbracket_{\bar{\mathbf{d}}} \approx \llbracket \varphi_i \rrbracket_{\mathbf{sup}}$. In the latter case, thanks to Theorem 23 we construct a **distance**-automaton \mathcal{A}_i such that $\llbracket \mathcal{A}_i \rrbracket_{\mathbf{d}} \approx \llbracket \varphi_i \rrbracket_{\mathbf{sup}}$. The slaves of \mathcal{B} are $\mathcal{A}_1, \dots, \mathcal{A}_k$.

This completes the construction of \mathcal{B} , see Appendix A.5 for its correctness.

- $\varphi = \mathbf{A}^{\leq N}\psi$: The construction is the same as for $\mathbf{A}\psi$, except for the construction of the master \mathcal{A} , in which we replace Theorem 20 by Theorem 21 to account for the possibility of removing a bounded number of paths.
- $\varphi = \exists N \varphi'$: Note that φ cannot be negative. Since φ is positive, also φ' is positive. By the induction hypothesis, there exists a **distance**-automaton $\mathcal{A}_{\varphi'}$ such that $\llbracket \mathcal{A}_{\varphi'} \rrbracket_{\mathbf{d}} \approx \llbracket \varphi' \rrbracket_{\mathbf{inf}}$. Since φ is a positive sentence, we have $\llbracket \varphi \rrbracket_{\mathbf{inf}}(t) \in \{0, \omega\}$ for every t . Now,

$$\begin{aligned} \llbracket \varphi \rrbracket_{\mathbf{inf}}(t) = 0 &\iff \exists n \in \mathbb{N}, \llbracket \varphi' \rrbracket_{\mathbf{inf}}(t) \leq n \\ &\iff \exists n \in \mathbb{N}, \text{Eve wins } G_{\mathcal{A}_{\varphi'}, t} \text{ for the objective } \mathbf{parity} \cap \mathbf{distance}(n) \\ &\iff \text{Eve wins } G_{\mathcal{A}_{\varphi'}, t} \text{ for the objective } \mathbf{parity} \cap \mathbf{fininc} \end{aligned}$$

The third equivalence follows from Lemma 15. We can now apply Lemma 18 to the **parity** \cap **fininc**-automaton $\mathcal{A}_{\varphi'}$ to get an equivalent parity automaton \mathcal{A}_{φ} . Then the last item is equivalent to Eve winning the parity game $G_{\mathcal{A}_{\varphi}, t}$, which is equivalent to $\llbracket \mathcal{A}_{\varphi} \rrbracket_{\mathbf{d}}(t) = 0$ (since $\llbracket \mathcal{A}_{\varphi} \rrbracket_{\mathbf{d}}(t) \in \{0, \omega\}$ because \mathcal{A}_{φ} has no counter).

This completes the proof of the inductive hypothesis. Finally, since Φ is a sentence, \mathcal{A}_{Φ} is a parity automaton. Indeed, in the inductive steps, the boundedness operators introduces a counter (if there was not one already), the $\exists N$ step removes the counter, and other operators applied to arguments that do not have a counter produce automata with no counters. ◀

5.4 Model checking Prompt-SL and BOSL

The model-checking problem for PROMPT-SL (resp. BOSL) is the following: given a game \mathcal{G} and a sentence Φ of PROMPT-SL (resp. BOSL), decide whether $\mathcal{G} \models \Phi$.

As for ATL* with strategy context [35] and Strategy Logic with imperfect information [7], the model-checking problems for both PROMPT-SL and BOSL (as well as their combination) can be easily reduced to that of BOUND-QCTL* (see Appendix A.6). As a consequence of these reductions and of Theorem 25, we get:

► **Theorem 26.** *The model-checking problem is decidable for PROMPT-SL and BOSL.*

The model-checking procedure is nonelementary, but because PROMPT-SL and BOSL subsume SL we know from [36] that no elementary procedure exists. We leave precise complexity analysis for future work.

6 Conclusion

We introduced two quantitative extensions of Branching-time Strategy Logic (BSL), i.e., PROMPT-SL that extends BSL with $\mathbf{F}^{\leq N}$ that limits the range of the eventuality, and BOSL that extends BSL with $\mathbf{A}^{\leq N}$ that limits the range of the outcome quantifier. We proved that model checking both these logics is decidable. To the best of our knowledge these are the first quantitative extensions of SL with decidable model-checking problem.

In order to prove our results we used notions from the theory of regular cost functions to develop new technical insights necessary to address PROMPT-SL and BOSL. Moreover, as an intermediate formalism between cost automata and logics for strategic reasoning we introduced BOUND-QCTL*, a quantitative extension of QCTL*, and proved its model checking decidable. Using this, it is easy to see that also the extension of BSL with $\exists N$ and both $\mathbf{F}^{\leq N}$ and $\mathbf{A}^{\leq N}$ has a decidable model-checking problem.

References

- 1 Shaull Almagor, Yoram Hirshfeld, and Orna Kupferman. Promptness in ω -regular automata. In *ATVA*, LNCS 6252, pages 22–36. Springer, 2010.
- 2 Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron Peled. Parametric temporal logic for “model measuring”. *ACM Transactions on Computational Logic*, 2(3):388–407, 2001.
- 3 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.
- 4 Benjamin Aminof, Aniello Murano, Sasha Rubin, and Florian Zuleger. Prompt alternating-time epistemic logics. In *KR*, pages 258–267. AAAI Press, 2016. URL: <http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12890>.
- 5 Mordechai Ben-Ari, Zohar Manna, and Amir Pnueli. The temporal logic of branching time. In *POPL*, pages 164–176, 1981. doi:10.1145/567532.567551.
- 6 Raphaël Berthon, Bastien Maubert, and Aniello Murano. Decidability results for ATL* with imperfect information and perfect recall. In *AAMAS*, 2017.
- 7 Raphaël Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin, and Moshe Y. Vardi. Strategy logic with imperfect information. In *LICS*, 2017.
- 8 Alessandro Bianco, Fabio Mogavero, and Aniello Murano. Graded computation tree logic. *ACM Transactions on Computational Logic*, 13(3):25:1–25:53, 2012. doi:10.1145/2287718.2287725.
- 9 Patricia Bouyer, Patrick Gardy, and Nicolas Markey. Weighted strategy logic with boolean goals over one-counter games. In *FSTTCS 2015*, pages 69–83, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.69.
- 10 Arnaud Carayol and Olivier Serre. How good is a strategy in a game with nature? In *LICS*, pages 609–620. IEEE Computer Society, 2015.
- 11 Krishnendu Chatterjee and Nathanaël Fijalkow. Infinite-state games with finitary conditions. In *CSL*, pages 181–196, 2013. doi:10.4230/LIPIcs.CSL.2013.181.
- 12 Krishnendu Chatterjee, Thomas A Henzinger, and Florian Horn. Finitary winning in ω -regular games. *ACM Transactions on Computational Logic*, 11(1):1, 2009.
- 13 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy Logic. *Information and Computation*, 208(6):677–693, 2010. doi:10.1016/j.ic.2009.07.004.
- 14 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *ICALP*, 2009.
- 15 Thomas Colcombet. Fonctions régulières de coût. Habilitation Thesis, 2013.
- 16 Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013.
- 17 Thomas Colcombet and Nathanaël Fijalkow. The bridge between regular cost functions and ω -regular languages. In *ICALP*, pages 126:1–126:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.126.
- 18 Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *LICS*, pages 70–79, 2010. doi:10.1109/LICS.2010.36.
- 19 E. Allen Emerson and Joseph Y. Halpern. “Sometimes” and “Not Never” revisited: On branching versus linear time. In *POPL*, pages 127–140, 1983. doi:10.1145/567067.567081.
- 20 Nathanaël Fijalkow, Florian Horn, Denis Kuperberg, and Michał Skrzypczak. Trading bounds for memory in games with counters. In *ICALP*, pages 197–208, 2015. doi:10.1007/978-3-662-47666-6_16.
- 21 Nathanaël Fijalkow and Martin Zimmermann. Cost-Parity and Cost-Street Games. In *FSTTCS*, volume LIPIcs 18, pages 124–135, 2012.

- 22 Nathanaël Fijalkow and Martin Zimmermann. Parity and Streett games with costs. *Logical Methods in Computer Science*, 10(2), 2014. doi:10.2168/LMCS-10(2:14)2014.
- 23 Tim French. Decidability of quantified propositional branching time logics. In *AJCAI'01*, pages 165–176, 2001. doi:10.1007/3-540-45656-2_15.
- 24 Patrick Gardy. *Semantics of Strategy Logic*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, 2017. URL: <https://tel.archives-ouvertes.fr/tel-01561802>.
- 25 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *CSL*, pages 395–410, 2006.
- 26 Sophia Knight and Bastien Maubert. Dealing with imperfect information in strategy logic. In *SR*, 2015.
- 27 Denis Kuperberg. Linear temporal logic for regular cost functions. *Logical Methods in Computer Science*, 10(1), 2014.
- 28 Denis Kuperberg and Michael Vanden Boom. On the expressive power of cost logics over infinite words. In *ICALP*, pages 287–298, 2012.
- 29 Orna Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. *JLC*, 9(2):135–147, 1999. doi:10.1093/logcom/9.2.135.
- 30 Orna Kupferman, P. Madhusudan, P. S. Thiagarajan, and Moshe Y. Vardi. Open systems in reactive environments: Control and synthesis. In *CONCUR*, LNCS 1877, pages 92–107. Springer, 2000.
- 31 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Annals of Mathematics and Artificial Intelligence*, 78(1):3–20, 2016. doi:10.1007/s10472-016-9508-8.
- 32 Orna Kupferman, Nir Piterman, and Moshe Y Vardi. From liveness to promptness. *Formal Methods in System Design*, 34(2):83–103, 2009.
- 33 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. doi:10.1145/333979.333987.
- 34 François Laroussinie and Nicolas Markey. Quantified CTL: expressiveness and complexity. *LMCS*, 10(4), 2014. doi:10.2168/LMCS-10(4:17)2014.
- 35 François Laroussinie and Nicolas Markey. Augmenting ATL with strategy contexts. *Information and Computation*, 245:98–123, 2015.
- 36 Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. Reasoning about strategies: On the model-checking problem. *ACM Transactions on Computational Logic*, 15(4):34:1–34:47, 2014. doi:10.1145/2631917.
- 37 Fabio Mogavero, Aniello Murano, and Loredana Sorrentino. On promptness in parity games. *Fundamenta Informaticae*, 139(3):277–305, 2015.
- 38 Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977. doi:10.1109/SFCS.1977.32.
- 39 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.
- 40 A Prasad Sistla. *Theoretical Issues in the Design and Certification of Distributed Systems*. PhD thesis, Harvard University, Cambridge, MA, USA, 1983.
- 41 Martin Zimmermann. Optimal bounds in parametric LTL games. *Theoretical Computer Science*, 493:30–45, 2013. doi:10.1016/j.tcs.2012.07.039.

A

 Appendix

A.1 BOSL semantics

► **Definition 4.** The semantics is defined inductively as follows, where φ (resp. ψ) is a cost-SL state (resp. path) formula, \mathcal{G} is a game, χ is an assignment variable-complete for φ (resp. ψ), ρ is a finite play, π an infinite one, $i \in \mathbb{N}$ is a point in time and $n \in \mathbb{N}$ is a bound.

$\mathcal{G}, \chi, \rho, n \models p$	if	$p \in \ell(\text{last}(\rho))$
$\mathcal{G}, \chi, \rho, n \models \neg\varphi$	if	$\mathcal{G}, \chi, \rho, n \not\models \varphi$
$\mathcal{G}, \chi, \rho, n \models \varphi \vee \varphi'$	if	$\mathcal{G}, \chi, \rho, n \models \varphi$ or $\mathcal{G}, \chi, \rho, n \models \varphi'$
$\mathcal{G}, \chi, \rho, n \models \exists s\varphi$	if	there exists $\sigma \in \text{Strat}$ s.t. $\mathcal{G}, \chi[s \mapsto \sigma], \rho, n \models \varphi$
$\mathcal{G}, \chi, \rho, n \models (a, s)\varphi$	if	$\mathcal{G}, \chi[a \mapsto \chi(s)], \rho, n \models \varphi$
$\mathcal{G}, \chi, \rho, n \models (a, ?)\varphi$	if	$\mathcal{G}, \chi[a \mapsto ?], \rho, n \models \varphi$
$\mathcal{G}, \chi, \rho, n \models \mathbf{A}\psi$	if	for all $\pi \in \text{Out}(\chi, \rho)$, $\mathcal{G}, \chi, \pi, \rho - 1, n \models \psi$
$\mathcal{G}, \chi, \rho, n \models \mathbf{A}^{\leq N}\psi$	if	$ \{\pi \in \text{Out}(\rho, \chi) \mid \mathcal{G}, \chi, \pi, \rho - 1, n \not\models \psi\} \leq n$
$\mathcal{G}, \chi, \rho, n \models \exists N\varphi$	if	there exists $n' \in \mathbb{N}$ such that $\mathcal{G}, \chi, \rho, n' \models \varphi$
$\mathcal{G}, \chi, \pi, i, n \models \varphi$	if	$\mathcal{G}, \chi, \pi_{\leq i}, n \models \varphi$
$\mathcal{G}, \chi, \pi, i, n \models \neg\psi$	if	$\mathcal{G}, \chi, \pi, i, n \not\models \psi$
$\mathcal{G}, \chi, \pi, i, n \models \psi \vee \psi'$	if	$\mathcal{G}, \chi, \pi, i, n \models \psi$ or $\mathcal{G}, \chi, \pi, i, n \models \psi'$
$\mathcal{G}, \chi, \pi, i, n \models \mathbf{X}\psi$	if	$\mathcal{G}, \chi, \pi, i + 1, n \models \psi$
$\mathcal{G}, \chi, \pi, i, n \models \psi \mathbf{U}\psi'$	if	$\exists j \geq i$ s.t. $\mathcal{G}, \chi, \pi, j, n \models \psi'$ and $\forall k$ s.t. $i \leq k < j$, $\mathcal{G}, \chi, \pi, k, n \models \psi$

A.2 Bound-QCTL* semantics

Given two trees t, t' and an atomic proposition p , we write $t \equiv_p t'$ if they have the same domain τ and for all p' in AP such that $p' \neq p$, for all u in τ , we have $p' \in \ell(u)$ iff $p' \in \ell'(u)$;

► **Definition 7.** The semantics $t, u, n \models \varphi$ and $t, \lambda, n \models \psi$ are defined inductively, where φ is a BOUND-QCTL* state formula, ψ is a BOUND-QCTL* path formula, $t = (\tau, \ell)$ is a tree, u is a node, λ is a branch in t , and n in \mathbb{N} a bound:

$t, u, n \models p$	if	$p \in \ell(u)$
$t, u, n \models \neg\varphi$	if	$t, u, n \not\models \varphi$
$t, u, n \models \varphi \vee \varphi'$	if	$t, u, n \models \varphi$ or $t, u, n \models \varphi'$
$t, u, n \models \mathbf{A}\psi$	if	$\forall \lambda \in \text{Branches}(t, u)$ we have $t, \lambda, n \models \psi$
$t, u, n \models \mathbf{A}^{\leq N}\psi$	if	$\text{Card}(\{\lambda \in \text{Branches}(t, u) : t, \lambda, n \not\models \psi\}) \leq n$
$t, u, n \models \exists p\varphi$	if	$\exists t' \equiv_p t$ such that $t', u, n \models \varphi$
$t, u, n \models \exists N\varphi$	if	$\exists n' \in \mathbb{N}$ such that $t, u, n' \models \varphi$,
$t, \lambda, n \models \varphi$	if	$t, \lambda_0, n \models \varphi$
$t, \lambda, n \models \neg\psi$	if	$t, \lambda, n \not\models \psi$
$t, \lambda, n \models \psi \vee \psi'$	if	$t, \lambda, n \models \psi$ or $t, \lambda, n \models \psi'$
$t, \lambda, n \models \mathbf{X}\psi$	if	$t, \lambda_{\geq 1}, n \models \psi$
$t, \lambda, n \models \psi \mathbf{U}\psi'$	if	$\exists j \geq 0$ such that $t, \lambda_{\geq j}, n \models \psi'$ and $\forall k$ such that $0 \leq k < j$, $t, \lambda_{\geq k}, n \models \psi$
$t, \lambda, n \models \mathbf{F}^{\leq N}\psi$	if	$\exists j$ such that $0 \leq j \leq n$ and $t, \lambda_{\geq j}, n \models \psi$

A.3 Proof of Theorem 21

► **Theorem 21.** *Let \mathcal{A} be a history-deterministic **distance**-automaton over infinite words. One can construct a **distance**-automaton recognising the function over trees*

$$f : t \mapsto \inf \{ \max(n, \sup \{ \llbracket \mathcal{A} \rrbracket_d(\lambda) : \lambda \notin B \}) : n \in \mathbb{N}, B \subseteq \text{Branches}(t), \text{Card}(B) \leq n \}.$$

To prove Theorem 21 we combine \mathcal{A} with the automaton defined in the proof of Lemma 17.

Proof. We write $\mathcal{A} = (Q, q_0, \delta, c)$ for the history-deterministic **distance**-automaton over infinite words, and let us say that the set of labels is $\{1, \dots, d\} \times \{\epsilon, \mathbf{i}\}$ with d even.

We construct a **distance**-automaton \mathcal{B} for f as follows. The set of states is $Q \times \{p_{0,\epsilon}, p_{0,\mathbf{i}}, p_1\}$, where the semantics of $p_{0,\epsilon}$ and $p_{0,\mathbf{i}}$ is “some path will be skipped” and p_1 means “no path will be skipped”. The initial state is $(q_0, p_{0,\epsilon})$. The first component simulates the automaton \mathcal{A} on all branches, while the second acts as follows, with $p_0 = \{p_{0,\epsilon}, p_{0,\mathbf{i}}\}$.

$$\delta = \begin{cases} (p_{0,\epsilon}, a, h) & \text{if } h \text{ contains at most one } p_0 \\ (p_{0,\mathbf{i}}, a, h) & \text{if } h \text{ contains at least two } p_0 \\ (p_1, a, h) & \text{if } h \text{ contains only } p_1 \end{cases}$$

The labelling function c' is

$$\begin{aligned} c'(q, p_{0,\epsilon}) &= (d, a) && \text{where } c(q) = (o, a) \\ c'(q, p_{0,\mathbf{i}}) &= (d, \mathbf{i}) && \text{where } c(q) = (o, a) \\ c'(q, p_1) &= c(q) \end{aligned}$$

The proof of correctness is the same as for Lemma 17, substantiating the following claims:

- if $f(t) \leq n$, then $\llbracket \mathcal{B} \rrbracket_d(t) \leq n$,
- if $\llbracket \mathcal{B} \rrbracket_d(t) \leq n$, then $f(t) \leq \text{Card}(S)^n$. ◀

A.4 Semantics of nested W -automata

► **Definition 22.** A nested W -automaton with k slaves over (Σ, S) -trees is given by

- a *master automaton* \mathcal{A} , which is a W -automaton over $(2^k, S)$ -trees, and
- k *slave automata* $(\mathcal{A}_i)_{i \in [k]}$, which are W -automata over (Σ, S) -trees.

The transition relation of the master is $\delta \subseteq Q \times 2^k \times Q^S$. We describe the modus operandi of a nested automaton informally. Let t be a tree and u a node in t , labelled with state q . To take the next transition the master automaton interrogates its slaves: the transition $(q, v, h) \in \delta$ is allowed if for all $i \in v$, the subtree t_u is accepted by \mathcal{A}_i .

To define the semantics of nested W -automata, we define the corresponding acceptance games. Given a nested W -automaton $\mathcal{B} = (\mathcal{A}, (\mathcal{A}_i)_{i \in [k]})$ and a tree t , we define the acceptance W -game $G_{\mathcal{B}, t}$ as follows. Let $\mathcal{A} = (Q, q_0, \delta, c)$.

- The set of vertices is $(Q \times t) \cup (Q \times t \times Q^S)$. The vertices of the form (q, u) are controlled by Eve, those of the form (q, u, h) by Adam.
- The initial vertex is (q_0, r) , where r is the root of t .
- The transition relation E is defined as follows

$$\begin{cases} (q, u) E (q, u, h) & \text{if } (q, \ell(u), h) \in \delta \text{ and } \forall i \in v, t_u \text{ is accepted by } \mathcal{A}_i, \\ (q, u, h) E (h(s), u \cdot s). \end{cases}$$

- The labelling function maps (q, u) to $c(q)$, the other vertices are not labelled.

We say that t is accepted by \mathcal{B} if Eve wins the acceptance W -game $G_{\mathcal{B}, t}$.

A.5 Proof of Theorem 25

► **Theorem 25.** *Let Φ be a sentence of BOUND-QCTL*. We construct a non-deterministic parity automaton \mathcal{A}_Φ over (Σ, S) -trees such that for every Kripke structure \mathcal{S} over the set of states S , we have $\mathcal{S} \models \Phi$ if, and only if, \mathcal{A}_Φ accepts the unfolding $t_{\mathcal{S}}$.*

Proof. Let Φ be a sentence and S a finite set of states. Throughout this proof, by trees we mean regular trees, so in particular \approx is understood over such trees.

For each subformula φ of Φ , we construct by induction on φ the following automata:

1. if φ is positive, a **distance**-automaton \mathcal{A}_φ such that $\llbracket \mathcal{A}_\varphi \rrbracket_{\mathbf{d}} \approx \llbracket \varphi \rrbracket_{\mathbf{inf}}$,
2. if φ is negative, a **distance**-automaton \mathcal{A}_φ such that $\llbracket \mathcal{A}_\varphi \rrbracket_{\bar{\mathbf{d}}} \approx \llbracket \varphi \rrbracket_{\mathbf{sup}}$.

Here are the constructions or proofs of correctness not present in the body of the paper.

■ $\varphi = \mathbf{p}$:

The formula φ is both positive and negative. Seeing it a positive formula, we define a **distance**-automaton \mathcal{A}_p with one state q_0 and transition function defined as follows:

$$\delta(q_0, a) = \begin{cases} \top & \text{if } p \in a \\ \perp & \text{otherwise.} \end{cases}$$

Seeing φ as a negative formula, we define a **distance**-automaton \mathcal{A}_p in exactly the same way.

■ $\varphi = \neg\varphi'$:

If φ is negative, then φ' is positive. By definition,

$$\llbracket \varphi \rrbracket_{\mathbf{sup}}(t) = \sup \{n \in \mathbb{N} : t, r, n \models \varphi\} = \inf \{n \in \mathbb{N} : t, r, n \models \varphi'\} - 1 = \llbracket \varphi' \rrbracket_{\mathbf{inf}}(t) - 1.$$

In particular, $\llbracket \varphi \rrbracket_{\mathbf{sup}} \approx \llbracket \varphi' \rrbracket_{\mathbf{inf}}$. By induction hypothesis, there exists a **distance**-automaton $\mathcal{A}_{\varphi'}$ such that $\llbracket \mathcal{A}_{\varphi'} \rrbracket_{\mathbf{d}} \approx \llbracket \varphi' \rrbracket_{\mathbf{inf}}$. Thanks to Theorem 23, there exists a **distance**-automaton \mathcal{A}_φ such that $\llbracket \mathcal{A}_\varphi \rrbracket_{\bar{\mathbf{d}}} \approx \llbracket \mathcal{A}_{\varphi'} \rrbracket_{\mathbf{d}}$. It follows that $\llbracket \mathcal{A}_\varphi \rrbracket_{\bar{\mathbf{d}}} \approx \llbracket \varphi \rrbracket_{\mathbf{sup}}$.

If φ is positive, then φ' is negative, and a similar reasoning applies, using Theorem 23 to turn a **distance**-automaton into an equivalent **distance**-automaton.

■ $\varphi = \varphi_1 \vee \varphi_2$:

If φ is positive, then both φ_1 and φ_2 are positive. By induction hypothesis, there exist two **distance**-automata \mathcal{A}_{φ_1} and \mathcal{A}_{φ_2} such that $\llbracket \mathcal{A}_{\varphi_1} \rrbracket_{\mathbf{d}} \approx \llbracket \varphi_1 \rrbracket_{\mathbf{inf}}$ and $\llbracket \mathcal{A}_{\varphi_2} \rrbracket_{\mathbf{d}} \approx \llbracket \varphi_2 \rrbracket_{\mathbf{inf}}$. We construct \mathcal{A}_φ by taking the disjoint union of \mathcal{A}_{φ_1} and \mathcal{A}_{φ_2} and adding a new initial state that nondeterministically chooses which of \mathcal{A}_{φ_1} or \mathcal{A}_{φ_2} to execute on the input tree, so that $\llbracket \mathcal{A}_\varphi \rrbracket_{\mathbf{d}} = \min \{ \llbracket \mathcal{A}_{\varphi_1} \rrbracket_{\mathbf{d}}, \llbracket \mathcal{A}_{\varphi_2} \rrbracket_{\mathbf{d}} \} \approx \min \{ \llbracket \varphi_1 \rrbracket_{\mathbf{inf}}, \llbracket \varphi_2 \rrbracket_{\mathbf{inf}} \} = \llbracket \varphi \rrbracket_{\mathbf{inf}}$.

If φ is negative, both φ_1 and φ_2 are negative. The same construction yields an automaton \mathcal{A}_φ such that $\llbracket \mathcal{A}_\varphi \rrbracket_{\bar{\mathbf{d}}} = \max \{ \llbracket \mathcal{A}_{\varphi_1} \rrbracket_{\bar{\mathbf{d}}}, \llbracket \mathcal{A}_{\varphi_2} \rrbracket_{\bar{\mathbf{d}}} \} \approx \max \{ \llbracket \varphi_1 \rrbracket_{\mathbf{sup}}, \llbracket \varphi_2 \rrbracket_{\mathbf{sup}} \} = \llbracket \varphi \rrbracket_{\mathbf{sup}}$.

- $\varphi = \mathbf{A}\psi$: The idea is similar to the automata construction for branching-time logic [33]: intuitively, treat ψ as an LTL formula over maximal state subformulas, run a deterministic automaton for ψ on all branches of the tree, and launch automata for the maximal state subformulas of ψ when needed. In our case, we will construct a nested automaton to do this, and in place of a deterministic parity automaton for ψ we will use a history-deterministic **distance**-automaton. Finally, we will convert the nested **distance**-automaton into a **distance**-automaton.

So, suppose that φ is positive (the case that φ is negative is treated dually). Then also ψ is positive. We will construct a nested **distance**-automaton \mathcal{B} such that $\llbracket \mathcal{B} \rrbracket_{\mathbf{d}} \approx \llbracket \varphi \rrbracket_{\mathbf{inf}}$. Let $\varphi_1, \dots, \varphi_k$ be the maximal state subformulas of the path formula ψ . We see these formulas as atomic propositions, so that the formula ψ can be seen as a PROMPT-LTL

formula on infinite words over the alphabet 2^k . Apply Theorem 24 to ψ to get a history-deterministic **distance**-automaton \mathcal{A}_ψ over infinite words such that $\llbracket \mathcal{A}_\psi \rrbracket_{\mathbf{d}} \approx \llbracket \psi \rrbracket_{\mathbf{inf}}$. Then, apply Theorem 20 to \mathcal{A}_ψ to get a **distance**-automaton \mathcal{A} such that $\llbracket \mathcal{A} \rrbracket_{\mathbf{d}}(t) = \sup \{ \llbracket \mathcal{A}_\psi \rrbracket_{\mathbf{d}}(\lambda) : \lambda \in \text{Branches}(t) \}$. The master of \mathcal{B} is \mathcal{A} .

Since ψ is positive, the formulas $\varphi_1, \dots, \varphi_k$ are either positive or negative. By the induction hypothesis, for every i , if φ_i is positive we construct a **distance**-automaton \mathcal{A}_i such that $\llbracket \mathcal{A}_i \rrbracket_{\mathbf{d}} \approx \llbracket \varphi_i \rrbracket_{\mathbf{inf}}$; and if φ_i is negative, we construct a **distance**-automaton \mathcal{A}'_i such that $\llbracket \mathcal{A}'_i \rrbracket_{\mathbf{d}} \approx \llbracket \varphi_i \rrbracket_{\mathbf{sup}}$. In the latter case, thanks to Theorem 23 we construct a **distance**-automaton \mathcal{A}_i such that $\llbracket \mathcal{A}_i \rrbracket_{\mathbf{d}} \approx \llbracket \varphi_i \rrbracket_{\mathbf{sup}}$. The slaves of \mathcal{B} are $\mathcal{A}_1, \dots, \mathcal{A}_k$.

This completes the construction of \mathcal{B} . We now prove that $\llbracket \mathcal{B} \rrbracket_{\mathbf{d}} \approx \llbracket \varphi \rrbracket_{\mathbf{inf}}$. For the sake of simplicity, we assume that $\llbracket \mathcal{A}_\psi \rrbracket_{\mathbf{d}} = \llbracket \psi \rrbracket_{\mathbf{d}}$ and $\llbracket \mathcal{A}_i \rrbracket_{\mathbf{d}} = \llbracket \varphi_i \rrbracket_{\mathbf{d}}$ for every i , i.e. we replace \approx by equality. This simplification does not affect the arguments and makes the proof easier to read.

- We prove that $\llbracket \varphi \rrbracket_{\mathbf{inf}} \leq \llbracket \mathcal{B} \rrbracket_{\mathbf{d}}$. It is sufficient to show that $\llbracket \mathcal{B} \rrbracket_{\mathbf{d}}(t) \leq n$ implies $\llbracket \varphi \rrbracket_{\mathbf{inf}}(t) \leq n$. A run of \mathcal{B} on t witnessing that $\llbracket \mathcal{B} \rrbracket_{\mathbf{d}}(t) \leq n$ yields for each branch λ a run of \mathcal{A}_ψ such that $\llbracket \mathcal{A}_\psi \rrbracket_{\mathbf{d}}(\lambda) \leq n$. The slave automata diligently check that the atomic propositions $\varphi_1, \dots, \varphi_k$ have been correctly used, so indeed $t, \lambda, n \models \psi$, thus $\llbracket \varphi \rrbracket_{\mathbf{inf}}(t) \leq n$.
- We prove that $\llbracket \mathcal{B} \rrbracket_{\mathbf{d}} \leq \llbracket \varphi \rrbracket_{\mathbf{inf}}$. It is sufficient to show that $\llbracket \varphi \rrbracket_{\mathbf{inf}}(t) \leq n$ implies $\llbracket \mathcal{B} \rrbracket_{\mathbf{d}}(t) \leq n$. By the semantics of φ for all branches λ of t we have $t, \lambda, n \models \psi$. This yields a run of \mathcal{B} on t witnessing that $\llbracket \mathcal{B} \rrbracket_{\mathbf{d}}(t) \leq n$.

Finally, applying Theorem 23 to the nested **distance**-automaton \mathcal{B} we get a **distance**-automaton \mathcal{A}_φ such that $\llbracket \mathcal{A}_\varphi \rrbracket_{\mathbf{d}} \approx \llbracket \mathcal{B} \rrbracket_{\mathbf{d}}$.

- $\varphi = \exists \mathbf{p} \varphi'$:

If φ is positive, then φ' is positive. In this case unravelling the definitions we have

$$\llbracket \varphi \rrbracket_{\mathbf{inf}}(t) = \inf \{ \llbracket \varphi' \rrbracket_{\mathbf{inf}}(t') : t' \equiv_p t \}.$$

By the induction hypothesis, there exists a **distance**-automaton $\mathcal{A}_{\varphi'}$ such that $\llbracket \mathcal{A}_{\varphi'} \rrbracket_{\mathbf{d}} \approx \llbracket \varphi' \rrbracket_{\mathbf{inf}}$. We obtain a **distance**-automaton \mathcal{A}_φ by performing the usual projection operation. Everything remains the same, but the transition relation: (q, a, h) is in the new transition relation if there exists a' such that $a' \equiv_p a$ and (q, a', h) is in δ , where $a' \equiv_p a$ if for all p' in AP such that $p' \neq p$, we have $p' \in a'$ if, and only if, $p' \in a$.

If φ is negative, then φ' is negative. The same reasoning and construction applies in this case, with

$$\llbracket \varphi \rrbracket_{\mathbf{sup}}(t) = \sup \{ \llbracket \varphi' \rrbracket_{\mathbf{sup}}(t') : t' \equiv_p t \}.$$

This completes the proof of the inductive hypothesis. Finally, since Φ is a sentence, \mathcal{A}_Φ is a parity automaton. Indeed, in the inductive steps, the boundedness operators introduces a counter (if there was not one already), the $\exists N$ step removes the counter, and every other operator applied to arguments that do not have a counter produces an automaton with no counters. \blacktriangleleft

A.6 Reductions for Prompt-SL and BOSL

Models transformation. We first define for every game \mathcal{G} a Kripke structure $\mathcal{S}_{\mathcal{G}}$ and a bijection $\rho \mapsto u_\rho$ between the set of finite plays starting in the initial vertex and the set of nodes in $t_{\mathcal{S}_{\mathcal{G}}}$. We consider propositions $\text{AP}_v = \{p_v \mid v \in V\}$, that we assume to be disjoint from AP. Define the Kripke structure $\mathcal{S}_{\mathcal{G}} = (S, R, s_0, \ell')$ where

- $S = \{s_v \mid v \in V\}$,
- $R = \{(s_v, s_{v'}) \mid \exists \mathbf{c} \in \text{Act}^{\text{Ag}} \text{ s.t. } \Delta(v, \mathbf{c}) = v'\} \subseteq S^2$,
- $s_0 = s_{v_0}$, and
- $\ell'(s_v) = \ell(v) \cup \{p_v\} \subseteq \text{AP} \cup \text{AP}_v$.

For every finite play $\rho = v_0 \dots v_k$, define the node $u_\rho = s_{v_0} \dots s_{v_k}$ in $t_{\mathcal{S}_G}$ (which exists, by definition of \mathcal{S}_G and of tree unfoldings). Note that the mapping $\rho \mapsto u_\rho$ defines a bijection between the set of paths from v_0 and the set of nodes in $t_{\mathcal{S}_G}$.

Formulas translation. Given a game \mathcal{G} and a formula φ of PROMPT-SL or BOSL, we define a BOUND-QCTL* formula $(\varphi)^f$ such that $\mathcal{G} \models \varphi$ if and only if $\mathcal{S}_G \models (\varphi)^f$. More precisely, this translation is parameterised with a partial function $f : \text{Ag} \rightarrow \text{Var}$ which records bindings of agents to strategy variables. Suppose that $\text{Act} = \{c_1, \dots, c_l\}$. We define the two functions $(\cdot)_s^f$ and $(\cdot)_p^f$ by mutual induction on, respectively, state formulas φ and path formulas ψ .

Here is the definition of $(\cdot)_s^f$ for state formulas:

$$\begin{aligned}
(p)_s^f &= p & (\neg\varphi)_s^f &= \neg(\varphi)_s^f \\
(\varphi_1 \vee \varphi_2)_s^f &= (\varphi_1)_s^f \vee (\varphi_2)_s^f & (\exists N\varphi)_s^f &= \exists N(\varphi)_s^f \\
((a, s)\varphi)_s^f &= (\varphi)_s^{f[a \mapsto s]} & ((a, ?)\varphi)_s^f &= (\varphi)_s^{f[a \mapsto ?]} \\
(\exists s\varphi)_s^f &= \exists p_{c_1}^s \dots \exists p_{c_l}^s. \varphi_{\text{str}}(s) \wedge (\varphi)_s^f, & \text{where } \varphi_{\text{str}}(s) &= \mathbf{AG} \bigvee_{c \in \text{Act}} (p_c^s \wedge \bigwedge_{c' \neq c} \neg p_{c'}^s) \\
(\mathbf{A}\psi)_s^f &= \mathbf{A}(\psi_{\text{out}}(f) \rightarrow (\psi)_p^f) & (\mathbf{A}^{\leq N}\psi)_s^f &= \mathbf{A}^{\leq N}(\psi_{\text{out}}(f) \rightarrow (\psi)_p^f)
\end{aligned}$$

where

$$\psi_{\text{out}}(f) = \mathbf{G} \bigwedge_{v \in V} \left(p_v \rightarrow \bigvee_{\mathbf{c} \in \text{Act}^{\text{Ag}}} \left(\bigwedge_{a \in \text{dom}(f)} p_{c_a}^{f(a)} \wedge \mathbf{X}p_{\Delta(v, \mathbf{c})} \right) \right),$$

and for path formulas:

$$\begin{aligned}
(\varphi)_p^f &= (\varphi)_s^f & (\neg\psi)_p^f &= \neg(\psi)_p^f \\
(\varphi_1 \vee \varphi_2)_p^f &= (\varphi_1)_p^f \vee (\varphi_2)_p^f & (\mathbf{X}\psi)_p^f &= \mathbf{X}(\psi)_p^f \\
(\psi \mathbf{U} \psi')_p^f &= (\psi)_p^f \mathbf{U} (\psi')_p^f & (\mathbf{F}^{\leq N}\psi)_p^f &= \mathbf{F}^{\leq N}(\psi)_p^f
\end{aligned}$$

One can prove the following lemma, where φ is either a PROMPT-SL or a BOSL formula. The translation is essentially the same as in [35] and [7], and the cases for the new operators should be clear from their semantics.

► **Lemma 26.** *Suppose that $\text{dom}(f) = \text{dom}(\chi) \cap \text{Ag}$ and for all $a \in \text{dom}(f)$, $f(a) = x$ implies $\chi(a) = \chi(x)$. Then*

$$\mathcal{G}, \chi, \rho, n \models \varphi \quad \text{if and only if} \quad t_{\mathcal{S}_G}, u_\rho, n \models (\varphi)^f.$$

Applying this to a sentence Φ , any assignment χ , the initial vertex v_0 of \mathcal{G} , any bound n and the empty function \emptyset , we get that

$$\mathcal{G} \models \varphi \quad \text{if and only if} \quad t_{\mathcal{S}_G} \models (\varphi)^\emptyset.$$

A Fully Abstract Game Semantics for Countable Nondeterminism

W. John Gowers¹

Computer Science Department, University of Bath
Claverton Down Road, Bath. BA2 7QY, United Kingdom
W.J.Gowers@bath.ac.uk
 <https://orcid.org/0000-0002-4513-9618>

James D. Laird

Department of Computer Science, University of Bath
Claverton Down Road, Bath. BA2 7QY, United Kingdom
J.D.Laird@bath.ac.uk

Abstract

The concept of fairness for a concurrent program means that the program must be able to exhibit an unbounded amount of nondeterminism without diverging. Game semantics models of nondeterminism show that this is hard to implement; for example, Harmer and McCusker’s model only admits infinite nondeterminism if there is also the possibility of divergence. We solve a long standing problem by giving a fully abstract game semantics for a simple stateful language with a countably infinite nondeterminism primitive. We see that doing so requires us to keep track of infinitary information about strategies, as well as their finite behaviours. The unbounded nondeterminism gives rise to further problems, which can be formalized as a lack of continuity in the language. In order to prove adequacy for our model (which usually requires continuity), we develop a new technique in which we simulate the nondeterminism using a deterministic stateful construction, and then use combinatorial techniques to transfer the result to the nondeterministic language. Lastly, we prove full abstraction for the model; because of the lack of continuity, we cannot deduce this from definability of compact elements in the usual way, and we have to use a stronger universality result instead. We discuss how our techniques yield proofs of adequacy for models of nondeterministic PCF, such as those given by Tsukada and Ong.

2012 ACM Subject Classification Theory of computation → Denotational semantics

Keywords and phrases semantics, nondeterminism, games and logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.24

Acknowledgements This material is based on work supported by the EPSRC under Grant No. EP/K018868/1. I (Gowers) am grateful to Martin Hyland for our conversation that helped me to develop some of this material. We are also grateful for the comments made by the anonymous referees for helping to clarify certain points, elaborate on technical difficulties and point to similarities with other parts of the literature.

1 Introduction

Picture two concurrent processes P and Q with shared access to a variable v that holds natural numbers and is initialized to 0. The execution of P consists in an infinite loop that increments the value of v at each iteration. Meanwhile, Q performs some computation A ,

¹ funded by EPSRC grant EP/K018868/1



and then prints out the current value of v and terminates the whole program. Since we cannot predict in advance how many cycles of the loop in P will have elapsed by the time the computation A has completed, the value that ends up printed to the screen may be arbitrarily large. Furthermore, under the basic assumption that the task scheduler is *fair*; i.e., any pending task must eventually be executed, our program must always terminate by printing out some value to the screen.

We have therefore built an *unbounded nondeterminism* machine, that can print out arbitrarily large natural numbers but which never diverges. This is strictly more powerful than finitary choice nondeterminism². What we have just shown is that if we want to solve the problem of building a fair task scheduler, then we must in particular be able to solve the problem of building an unbounded nondeterminism machine.

This is an important observation to make about concurrent programming, because the task of modelling unbounded nondeterminism is difficult – indeed, considerably more so than that of modelling bounded nondeterminism. Dijkstra argues in [7, Ch. 9] that it is impossible to implement unbounded nondeterminism, showing that the natural constructs from which we construct imperative programs satisfy a *continuity* property that unbounded nondeterminism lacks. Park [17] shows that these problems can be surmounted if we use a weaker version of continuity (e.g., ω_1 - rather than ω -continuity), but the failure of composition to be continuous is a problem in itself for semanticists, for whom continuity is often a key ingredient in proofs of computational adequacy and full abstraction.

We shall explore some of these problems and how they may be solved, using game semantics to give a fully abstract model of a simple stateful language – Idealized Algol – enhanced with a countable nondeterminism primitive. We begin with a pair of examples that will illustrate the lack of continuity, from a syntactic point of view. Let \mathbf{nat} be our natural number type and consider a sequence of functions $\langle n : \mathbf{nat} \rightarrow \mathbf{nat} \rangle$, where $\langle n \ k$ evaluates to 0 if $k < n$ and diverges otherwise. In that case, the least upper bound of the $\langle n$ is the function that combines all their convergent behaviours; i.e., the function $\lambda k.k;0$ that evaluates its input and then returns 0. If $? : \mathbf{nat}$ is an unbounded nondeterminism machine, then function application to $?$ is not continuous; indeed, $\langle m \ ?$ may diverge – since $?$ may evaluate to $m + 1$, say. But $(\lambda k.k;0) \ ?$ always converges to 0.

Lack of continuity is a problem in denotational semantics because fixed-point combinators are typically built using least upper bounds, and proving adequacy of the model typically requires that these least upper bounds be preserved. In a non-continuous situation, we will need to come up with new techniques in order to prove adequacy without using continuity.

A closely connected problem with unbounded nondeterminism is that it leads to terms that may be distinguished only by their *infinitary* behaviour. A program that flashes a light an unboundedly nondeterministic number of times cannot reliably be distinguished in finite time from a program that flashes that light forever: however long we watch the light flash, there is always a chance that it will stop at some point in the future. From a game semantics point of view, this corresponds to the observation that it is not sufficient to consider sets of finite plays in order to define strategies: we must consider infinite sequences of moves as well.

² Using recursion, we can build a program out of finite nondeterminism that can produce arbitrarily large natural numbers; however, this program also admits the possibility of divergence, unless we are able to insist on fairness.

1.1 Related Work

Our game semantics model bears closest resemblance to that of Harmer and McCusker [8], which is a fully abstract model of Idealized Algol with *finite* nondeterminism. Indeed, our work can be viewed as an extension of the Harmer-McCusker model with the extra information on infinite plays that we need to model countable nondeterminism.

The idea of adding infinite traces into strategies in order to model unbounded nondeterminism goes back to Roscoe's work on CSP [20], and is very similar to work by Levy [13] on game semantics for a higher order language. In particular, we will need something similar to Levy's notion of a *lively* strategy – one that is a union of deterministic strategies – a property that does not automatically hold when we start tracking infinite plays.

An alternative approach to the game semantics of nondeterminism can be found in Tsukada and Ong's sheaf model of nondeterministic PCF [21] and in the more general work on concurrency by Winskel et al. (e.g., see [22] and [5]), in which there is a very natural interpretation of nondeterminism. Although we are able to give a model of Idealized Algol with countable nondeterminism in the more traditional Harmer-McCusker style, it seems necessary to introduce this extra machinery in order to model stateless languages such as PCF (and certainly to model concurrency). In the last section of this paper, we will show how our methods can be applied under very general circumstances, and in particular to some of these models of nondeterministic stateless languages.

Related work by Laird [11, 12] discusses a semantics for PCF with unbounded nondeterminism based on sequential algorithms and explores the role played by continuity; however, this semantics is not fully abstract. Laird's work is interesting because it shows that we can obtain a traditional adequacy proof for a semantics with one-sided continuity: composition is continuous with respect to functions, but not with respect to arguments.

The idea of using some constrained version of continuity to prove adequacy for countable nondeterminism goes back to Plotkin's work on power-domains [4]. A crucial observation in both [4] and [11] is that this sort of proof requires a Hoare logic in which we can reason about all the countable ordinals. We cannot use these techniques here, however, because our composition is not continuous on either side.

1.2 Contributions

The main concepts of game semantics and the steps we take to establish full abstraction are well-established, with a few exceptions. The idea of including infinitary information in strategies is not new, but this particular presentation, though closely related to that of [13], is the first example of using the technique to establish a compositional full abstraction result for may and must testing.

Levy's work in [13] is part of a tradition of techniques used to handle unbounded nondeterminism operationally, normally using Labelled Transition Systems (see, for example, [19]). The contribution of this work is to apply the basic idea of including infinitary information to a compositional setting, where the semantics is built using the algebraic structure of higher-order programs.

There are two points in the traditional Full Abstraction proof that depend on composition being continuous, and we have had to come up with ways of getting round them. Firstly, in the absence of continuity, it no longer suffices to show that we can define every *compact* strategy; instead, we need a *universality* result allowing us to define certain infinite strategies – specifically, the *recursive* ones.

For the proof of adequacy, we have had to come up with a new technique, which can be thought of as a kind of synthesis between the two usual methods of proving adequacy – one involving logical relations and the other using more hands-on operational techniques. We do this by separating out the deterministic, continuous part of the strategy from the nondeterministic, discontinuous part. Using the stateful language, we can simulate individual evaluation paths of a nondeterministic program using a deterministic device that corresponds to the idea of ‘mocking’ a random number generator for testing purposes. This allows us to appeal to the adequacy result for deterministic Idealized Algol. We then rely on more combinatorial techniques in order to factor the nondeterminism back in.

This new technique is actually very generally applicable. We shall show that it may be used to prove adequacy for models of nondeterministic PCF under very mild assumptions. The Tsukada-Ong model, for example, satisfies these assumptions, allowing us to obtain an adequacy result for PCF with countable nondeterminism.

2 Idealized Algol with Countable Nondeterminism

We describe a type theory and operational semantics for Idealized Algol with countable nondeterminism. The types of our language are defined inductively as follows:

$$T ::= \mathbf{nat} \mid \mathbf{com} \mid \mathbf{Var} \mid T \rightarrow T.$$

Meanwhile, the terms are those given in [3], together with the nondeterministic choice:

$$\begin{aligned} M ::= & x \mid \lambda x.M \mid M M \mid \mathbf{Y}_T \mid \\ & \mathbf{n} \mid \mathbf{skip} \mid \mathbf{succ} \mid \mathbf{pred} \mid \\ & \mathbf{If0} \mid _ ; _ \mid _ := _ \mid \\ & @^3 \mid \mathbf{new}_T \mid \mathbf{mkvar} \mid ?. \end{aligned}$$

The typing rule for $?$ is $\Gamma \vdash ? : \mathbf{nat}$. We shall use v to range over variables of type \mathbf{Var} .

We define a small-step operational semantics for the language; this presentation is equivalent to the big-step semantics given in [8], except with a different rule for the countable rather than finite nondeterminism.

First, we define a Felleisen-style notion of *evaluation context* E inductively as follows.

$$\begin{aligned} E ::= & - \mid EM \mid \mathbf{succ} E \mid \mathbf{pred} E \mid \mathbf{If0} E \mid \\ & E; _ \mid E := _ \mid @E \mid \mathbf{mkvar} E \mid \mathbf{new}_T E \end{aligned}$$

We then give the appropriate small-step rules in Figure 1. In each rule, $\langle s, M \rangle$ is a *configuration* of the language, where M is a term, and s is a *store*; i.e., a function from the set of variables free in M to the set of natural numbers. If s is a store and v a variable, we write $\langle s \mid v \mapsto n \rangle$ for the state formed by updating the value of the variable v to n .

If $\langle \emptyset, M \rangle$ is a configuration with empty store, we call M a *closed term*. Given a closed term M of ground type \mathbf{com} or \mathbf{nat} , we write that $M \Downarrow x$ (where $x = \mathbf{skip}$ in the \mathbf{com} case and is a natural number in the \mathbf{nat} case) if there is a finite sequence $M \longrightarrow M_1 \longrightarrow \cdots \longrightarrow M_n = x$. If there is no infinite sequence $M \longrightarrow M_1 \longrightarrow M_2 \longrightarrow \cdots$, then we say that M *must converge*, and write $M \Downarrow^{\mathbf{must}}$. In general, we refer to a (finite or infinite) sequence $M \longrightarrow M_1 \longrightarrow \cdots$ that either terminates at an observable value or continues forever as an *evaluation* π of M .

³ That is, variable $@\text{access}$.

$$\begin{array}{c}
\frac{}{\langle s, (\lambda x.M) N \rangle \longrightarrow \langle s, M[N/x] \rangle} \qquad \frac{}{\langle s, \mathbf{Y}_T M \rangle \longrightarrow \langle s, M(\mathbf{Y}_T M) \rangle} \\
\frac{}{\langle s, \mathbf{succ} \ n \rangle \longrightarrow \langle s, n + 1 \rangle} \qquad \frac{}{\langle s, \mathbf{pred} \ n \rangle \longrightarrow \langle s, 0 \sqcup (n - 1) \rangle} \qquad \frac{}{\langle s, \mathbf{If0} \ 0MN \rangle \longrightarrow \langle s, M \rangle} \\
\frac{}{\langle s, \mathbf{If0} \ (n + 1)MN \rangle \longrightarrow \langle s, N \rangle} \qquad \frac{}{\langle s, \mathbf{@}(\mathbf{mkvar} \ MN) \rangle \longrightarrow \langle s, M \rangle} \\
\frac{}{\langle s, (\mathbf{mkvar} \ MN) := L \rangle \longrightarrow \langle s, N L \rangle} \qquad \frac{}{\langle s, v := n \rangle \longrightarrow \langle \langle s \mid v \mapsto n \rangle, \mathbf{skip} \rangle} \\
\frac{s(v) = n}{\langle s, \mathbf{@}v \rangle \longrightarrow \langle s, n \rangle} \qquad \frac{}{\langle s, \mathbf{skip}; M \rangle \longrightarrow \langle s, M \rangle} \\
\frac{}{\langle s, \mathbf{new}_T \lambda v.M \rangle \longrightarrow \langle \langle s \mid v \mapsto 0 \rangle, M \rangle} \qquad \frac{\langle s, M \rangle \longrightarrow \langle s, M' \rangle}{\langle s, E[M] \rangle \longrightarrow \langle s, E[M'] \rangle} \\
\frac{}{\langle s, ? \rangle \longrightarrow \langle s, n \rangle} \quad n \in \mathbb{N}
\end{array}$$

■ **Figure 1** Small-step operational semantics for Idealized Algol with countable nondeterminism.

Since the only case where we have any choice in which rule to use is the application of the rule for $?$, π may be completely specified by a finite or infinite sequence of natural numbers.

Let T be an Idealized Algol type, and let $M, N : T$ be closed terms. Then we write $M \sqsubseteq_{m\&m} N$ if for all contexts $C[-]$ of ground type with a hole of type T , we have

$$\begin{array}{l}
C[M] \Downarrow V \Rightarrow C[N] \Downarrow V \\
C[M] \Downarrow^{\text{must}} \Rightarrow C[N] \Downarrow^{\text{must}}
\end{array}$$

We write $M \equiv_{m\&m} N$ if $M \sqsubseteq_{m\&m} N$ and $N \sqsubseteq_{m\&m} M$.

3 Game Semantics

3.1 Arenas

An *arena* is given by a triple $A = (M_A, \lambda_A, \vdash_A)$, where

- M_A is a countable set of moves,
- $\lambda_A : M_A \rightarrow \{O, P\} \times \{Q, A\}$ designates each move as either an *O-move* or a *P-move*, and as either a *question* or an *answer*. We define $\lambda_A^{OP} = \text{pr}_1 \circ \lambda_A$ and $\lambda_A^{QA} = \text{pr}_2 \circ \lambda_A$. We also define $\neg : \{O, P\} \times \{Q, A\} \rightarrow \{O, P\} \times \{Q, A\}$ to be the function that reverses the values of O and P while leaving $\{Q, A\}$ unchanged.
- \vdash_A is an *enabling relation* between $M_A + \{*\}$ and M_A satisfying the following rules:
 - If $a \vdash_A b$, then $\lambda_A^{OP}(a) \neq \lambda_A^{OP}(b)$.
 - If $* \vdash_A a$, then $\lambda_A(a) = OQ$ and $b \not\vdash_A a$ for all $b \in M_A$.
 - If $a \vdash_A b$ and b is an answer, then a is a question.

We say that a move $a \in M_A$ is *initial* in A if $* \vdash_A a$.

Our base arenas will be the *flat arenas* for the types **nat** and **com**. Given a set X , the flat arena on X is the arena with a single *O*-question q and a *P*-answer x for each $x \in X$, where

$* \vdash q$ and $q \vdash x$ for each x . The denotations of the types \mathbf{nat} and \mathbf{com} are the flat arenas \mathbb{N} and \mathbb{C} on, respectively, the set of natural numbers and the singleton $\{a\}$.

We assume that our arenas are *enumerated*; i.e., that the set M_A is equipped with a partial surjection $\mathbb{N} \rightarrow M_A$. The denotation of any IA type has a natural enumeration.

Given an arena A , a *justified string* in A is a sequence s of moves in A , together with *justification pointers* that go from move to move in the sequence. The justification pointers must be set up in such a way that every non-initial move m in s has exactly one justification pointer going back to an earlier move n in s such that $n \vdash_A m$; we say that n *justifies* m . In particular, every justified string begins with an initial move, and hence with an O -question.

A *legal play* s is a justified string in A that strictly alternates between O -moves and P -moves and is such that the corresponding QA -sequence formed by applying λ_A^{QA} to moves is well-bracketed. We write L_A for the set of legal plays in A .

If s is a justified string, we will write sa for an arbitrary justified string extending s by a single move a , itself justified by some move in s .

3.2 Games and strategies

We use the approach taken by Abramsky and McCusker [3] – a middle road between the *arenas* of Hyland and Ong and the *games* of [2] that makes the linear structure more apparent.

Let s be a legal play in some arena A . If m and n are moves in s such that there is a chain of justification pointers leading from m back to n , we say that n *hereditarily justifies* m . Given some set S of initial moves in s , we write $s|_S$ for the subsequence of s made up of all those moves hereditarily justified by some move in S .

A *game* is a tuple $A = (M_A, \lambda_A, \vdash_A, P_A)$, where $(M_A, \lambda_A, \vdash_A)$ is an arena and P_A is a non-empty prefix-closed set of legal plays in that arena such that if $s \in P_A$ and I is a non-empty set of initial moves in s , then $s|_I \in P_A$.

Our base games will be the games \mathbb{N} and \mathbb{C} on the arenas of the same names, where $P_{\mathbb{N}} = \{\epsilon, q\} \cup \{qn : n \in \mathbb{N}\}$ and $P_{\mathbb{C}} = \{\epsilon, q, qa\}$.

3.2.1 Connectives

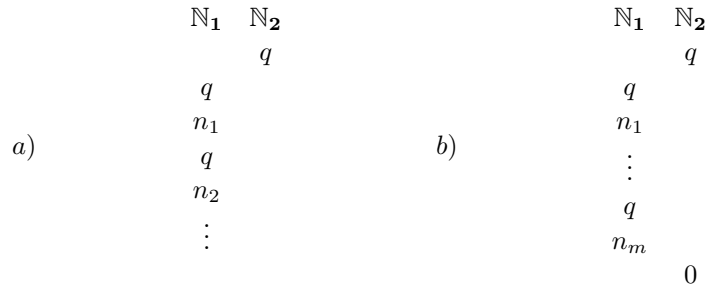
Let A, B be games. Then we may define games $A \times B$, $A \otimes B$, $A \multimap B$ and $!A$ as in [3]. As an example, we give the definition of $A \multimap B$:

$$\begin{aligned}
 M_{A \multimap B} &= M_A + M_B. \\
 \lambda_{A \multimap B} &= [\neg \circ \lambda_A, \lambda_B]. \\
 * \vdash_{A \multimap B} n &\Leftrightarrow * \vdash_B n. \\
 m \vdash_{A \multimap B} n &\Leftrightarrow \begin{array}{l} m \vdash_A n \text{ or } m \vdash_B n \\ \text{or (for } m \neq *) * \vdash_B m \text{ and} \\ * \vdash_A n. \end{array} \\
 P_{A \multimap B} &= \{s \in L_{A \multimap B} : s|_A \in P_A \text{ and } s|_B \in P_B\}.
 \end{aligned}$$

3.2.2 Modelling countable nondeterminism

As in [8], we model nondeterministic computations by relaxing the determinism constraint on strategies – so player P may have multiple replies to any given O -move.

In addition, we have to keep track of any possible divergence in the computation so we can distinguish terms such as $\text{If } 0 ? \Omega \ 0$, which may diverge, and 0 , which must converge.



■ **Figure 2** Finite plays alone are not sufficient to distinguish between terms of a language with countable nondeterminism.

To fix this problem, we follow [8] by modelling a strategy as a pair (T_σ, D_σ) , where T_σ is a nondeterministic strategy in the usual sense and D_σ is the set of those O -positions where there is a possibility of divergence.

We need to take some care when we compose strategies using ‘parallel composition plus hiding’. Specifically, we need to be able to add new divergences into strategies when they arise through ‘infinite chattering’ or *livelock*. For example, the denotation of the term

$$M = \mathbf{Y}_{\text{nat} \rightarrow \text{nat}}(\lambda f. \lambda n. n; (fn))$$

is given by a total strategy, without divergences: namely the strategy μ with plays of the form shown in Figure 2(a). However, when we compose this strategy with any total strategy for \mathbb{N} on the left, we expect the resulting strategy to contain divergences, since the term $M\mathbf{n}$ diverges for any \mathbf{n} . Semantically, this corresponds to the fact that we have a legal interaction $q q n q n \dots$ with an infinite tail in \mathbb{N}_1 ; when we perform ‘hiding’ by restricting the interaction to \mathbb{N} , we have no reply to the initial move q .

The approach adopted in [8] is to check specifically for infinite chattering between strategies $\sigma: A \multimap B$ and $\tau: B \multimap C$ by checking whether there is an infinite increasing sequence of interactions between σ and τ with an infinite tail in B . If there is such a sequence, then it restricts to some O -position in $\sigma; \tau$ and we add in a divergence at that position.

Harmer and McCusker’s approach works very satisfactorily for finite nondeterminism, but not at all for countable nondeterminism. To see why, consider the term

$$N = \mathbf{Y}_{\text{nat} \rightarrow \text{nat} \rightarrow \text{nat}}(\lambda g. \lambda mn. \text{If0 } m \ 0 \ (n; (g (\text{pred } m) n)))?.$$

This term first chooses a natural number m , and then reads from its input n for a total of m times before eventually returning 0. Thus, its denotation is the strategy ν with maximal plays of arbitrary length of the form shown in Figure 2(b). Note that this strategy strictly contains the strategy μ that we considered before, and therefore that the denotation of $\text{If0 } ?MN$ has the same denotation as N , even though for any n , $M\mathbf{n} \not\Downarrow^{\text{must}}$, while $N\mathbf{n} \Downarrow^{\text{must}}$. Moreover, if we try to compose $\llbracket N \rrbracket$ with the strategy on \mathbb{N} that always returns 1, then we end up with an infinite increasing sequence of positions, which triggers the introduction of a divergent play into the composite strategy – even though N must converge.

Aside from showing that the naive extension of the Harmer-McCusker model cannot be sound, this example actually leads to composition not being associative (e.g., see [9, 4.4.1]).

What this illustrates is the point made by Park in [17] and [18]: namely, that we can no longer deduce the infinitary behaviour of a strategy by looking at the limits of its finite plays; instead, we need to keep track of infinite sequences of moves explicitly, in the style

of [20] and [13]. When we use this technique, the denotation of M will contain an infinite sequence, while the denotation of N will contain arbitrarily long finite sequences, but no infinite sequences.

3.2.3 Strategies

We define an *infinite justified string* in an arena A in the obvious way. We say such a string is *recursive* if it corresponds, via the enumeration on M_A , to a pair of recursive functions $\mathbb{N} \rightarrow \mathbb{N}$ – one giving the sequence of moves and the other giving the justification relation.

We define $\overline{P_A}$ to be P_A together with the set of all those recursive infinite justified sequences that have all finite prefixes in P_A . Note that we deliberately ignore any non-recursive infinitary behaviours, since these cannot be detected by computable contexts.

We shall represent a strategy using two sets: a set T_σ of *traces*, which takes the role of the plays that may occur in the strategy (as in the usual definition of a deterministic strategy), and a second set D_σ of *divergences*; i.e., O -positions at which the strategy may elect to diverge. In order to model observational equivalence more closely, we shall require D_σ to be postfix-closed, since observable contexts cannot detect divergences that occur after the program might already have diverged: consider, for example, the terms $\Omega_{\text{nat} \rightarrow \text{nat}}$ or $(\lambda n.n)$ and $\Omega_{\text{nat} \rightarrow \text{nat}}$ or $(\lambda n.(n \text{ or } \Omega_{\text{nat}}))$ (where we have defined M or N to be $\text{If0} \ ? \ M \ N$).

For technical reasons we keep track of infinite plays in both T_σ and D_σ , with the rule that any infinite play in T_σ must be contained in D_σ (since it clearly corresponds to a divergent evaluation). We will require that every divergence arise from a trace; i.e., if every play in D_σ must have some prefix that is contained in both T_σ and D_σ . A consequence of this is that if $d \in D_\sigma$ is infinite and has no finite prefixes in D_σ , then it must also be contained in T_σ . Not too much importance should be given, however, to the presence or absence of infinite plays in T_σ : it is quick to show that once we pass to the intrinsic quotient, any such distinction vanishes.

Let A be a game. A *strategy* σ for A is a pair (T_σ, D_σ) , where:

- T_σ is a non-empty prefix-closed subset of $\overline{P_A}$ such that if $s \in T_\sigma$ is a P -position and $sa \in P_A$ then $sa \in T_\sigma$.
- $D_\sigma \subset \overline{P_A}$ is a postfix-closed set of plays in $\overline{P_A}$ that either end with an O -move or are infinite. We require D_σ to obey the following rules:
 - Divergences come from traces** If $d \in D_\sigma$ then there exists $s \sqsubseteq d$ such that $s \in T_\sigma \cap D_\sigma$.
 - Diverge-or-reply** If $s \in T_\sigma$ is an O -position, then either $s \in D_\sigma$ or $sa \in T_\sigma$ for some sa .
 - Infinite positions are divergent** If $s \in T_\sigma$ is infinite, then $s \in D_\sigma$.

3.2.4 Composition of strategies

Given games A, B, C , we define a justified string over A, B, C to be a sequence \mathfrak{s} of moves with justification pointers from all moves except the initial moves in C . Given such a string, we may form the restrictions $\mathfrak{s}|_{A,B}$ and $\mathfrak{s}|_{B,C}$ by removing all moves in either C or A , together with all justification pointers pointing into these games. We define $\mathfrak{s}|_{A,C}$ to be the sequence formed by removing all moves from B from \mathfrak{s} and all pointers to moves in B , *unless* we have a sequence of pointers $a \rightarrow b \rightarrow c$, in which case we replace them with a pointer $a \rightarrow c$.

We call \mathfrak{s} a *legal interaction* if $\mathfrak{s}|_{A,B} \in P_{A \multimap B}$, $\mathfrak{s}|_{B,C} \in P_{B \multimap C}$ and $\mathfrak{s}|_{A,C} \in P_{A \multimap C}$. We write $\text{int}_\infty(A, B, C)$ for the set of (possibly infinite) legal interactions between A , B and C .

Now, given strategies $\sigma: A \multimap B$ and $\tau: B \multimap C$, we define

$$T_\sigma \| T_\tau = \{ \mathfrak{s} \in \text{int}_\infty(A, B, C) : \mathfrak{s}|_{A,B} \in T_\sigma, \mathfrak{s}|_{B,C} \in T_\tau \},$$

and then set $T_{\sigma;\tau} = \{ \mathfrak{s}|_{A,C} : \mathfrak{s} \in T_\sigma \| T_\tau \}$.

As for divergences in $\sigma; \tau$, our approach is actually simpler than that in [8]; we set

$$D_{\sigma \dot{\downarrow} \tau} = \left\{ \mathfrak{s} \in \text{int}_{\infty}(A, B, C) \left| \begin{array}{l} \text{either } \mathfrak{s}|_{A,B} \in D_{\sigma} \text{ and } \mathfrak{s}|_{B,C} \in \\ T_{\tau} \\ \text{or } \mathfrak{s}|_{A,B} \in T_{\sigma} \text{ and } \mathfrak{s}|_{B,C} \in D_{\tau} \end{array} \right. \right\}.$$

We then set $D_{\sigma; \tau} = \text{pocl}_{A \rightarrow C} \{ \mathfrak{s}|_{A,C} : \mathfrak{s} \in D_{\sigma \dot{\downarrow} \tau} \}$, where $\text{pocl } X$ denotes the *postfix closure* of X ; i.e., the set of all O -plays in $P_{A \rightarrow C}$ that have some prefix in X .

Note that there is no need to consider separately, as Harmer and McCusker do, divergences that arise through ‘infinite chattering’: in our model, we will see that a case of infinite chattering between strategies σ and τ is itself a legal interaction between the two strategies, which is necessarily divergent (because it is infinite) and therefore gives rise to some divergence in $\sigma; \tau$.

We need to impose one more condition on strategies:

► **Definition 1.** Let σ be a strategy for a game A . We say that σ is *complete* if $T_{\sigma} = \overline{T_{\sigma}}$; i.e., T_{σ} contains an recursive infinite play s if it contains every finite prefix of s .

Any finite-nondeterminism strategy in the sense of [8] may be interpreted as a complete strategy by enlarging it with all its infinite recursive limiting plays. However, when we introduce countable nondeterminism, we also introduce strategies that are not complete. For example, the strategy ν that we mentioned above has an infinite increasing sequence of plays $q0 \sqsubseteq q0q0 \sqsubseteq \dots$, but has no infinite play corresponding to its limit. Nonetheless, we do not want to allow arbitrary strategies: for example, the strategy μ above should include the infinite play $qq0q0\dots$; the strategy μ° formed by removing this infinite play has no meaning in our language. Indeed, if we compose μ° with the strategy 0 for \mathbb{N} on the left, then the resulting strategy does not satisfy *diverge-or-reply*. The difference with ν is that every play $qq0\dots q0 \in T_{\nu}$ may be completed in ν by playing the move 0 on the right. In other words, ν is the union of complete strategies, while μ° is not.

► **Definition 2.** Let σ be a strategy for a game A . We say that σ is *locally complete* if it may be written as the union of complete strategies; i.e., there exist σ_i such that $T_{\sigma} = \bigcup T_{\sigma_i}$ and $D_{\sigma} = \bigcup D_{\sigma_i}$. Note that since T_{σ} and D_{σ} are countable sets (because there are countably many recursive plays), this union may be taken to be countable.

It will be slightly more convenient to use an equivalent definition, based on unions of *deterministic* strategies, which are a special case of complete strategies.

► **Definition 3.** We say that a strategy σ for a game A is *deterministic* if

- it is complete;
- if sa, sb are P -plays in T_{σ} then $a = b$ and the justifier of a is the justifier of b ;
- if $s \in D_{\sigma}$ then either s is infinite or there is no a such that $sa \in T_{\sigma}$.

We say that a strategy σ is *lively* or *locally deterministic* if there exists a collection of deterministic strategies σ_i such that $T_{\sigma} = \bigcup T_{\sigma_i}$ and $D_{\sigma} = \bigcup D_{\sigma_i}$. It is clear that a strategy is lively if and only if it is locally complete, and that the collection of σ_i may again be taken to be countable.

From now on, we will use *strategy* to mean *lively* (or *locally complete*) *strategy*. This means that we will need to show that the composition of lively strategies is again lively.

► **Lemma 4.** Let A, B, C be games and let $\sigma : A \multimap B$, $\tau : B \multimap C$ be deterministic strategies. Then $\sigma; \tau$ is complete.

Proof. The proof relies on a lemma from [10] that states (in our language) that if σ and τ are deterministic strategies and $s \in T_{\sigma;\tau}$ then there is a unique minimal $\mathfrak{s} \in T_{\sigma} \parallel T_{\tau}$ such that $\mathfrak{s}|_{A,C} = s$. That means that if $s_1 \sqsubseteq s_2 \sqsubseteq \dots$ is an infinite increasing sequence of plays in $T_{\sigma;\tau}$, with limit s , then there is a corresponding infinite increasing sequence of legal interactions $\mathfrak{s}_1 \sqsubseteq \mathfrak{s}_2 \sqsubseteq \dots$. Then the limit of the \mathfrak{s}_i is an infinite legal interaction \mathfrak{s} and we must have $\mathfrak{s}|_{A,B} \in \sigma$, $\mathfrak{s}|_{B,C} \in \tau$ by completeness of σ and τ . Therefore, $s = \mathfrak{s}|_{A,C} \in T_{\sigma;\tau}$. ◀

It is, of course, true that the composition of deterministic strategies is deterministic, but we do not really need this fact.

► **Corollary 5.** *The composition of strategies $\sigma: A \multimap B$ and $\tau: B \multimap C$ is a well-formed strategy for $A \multimap C$.*

Proof. The only tricky point is establishing that diverge-or-reply holds for $\sigma;\tau$. Again, it is sufficient to prove this in the case that σ and τ are deterministic and complete. Then it essentially follows from the argument used in [1] that shows that a partiality at an O -position $s \in T_{\sigma;\tau}$ must arise either from a partiality in T_{σ} or T_{τ} or from ‘infinite chattering’ between σ and τ . In the first case, the diverge-or-reply rule for σ and τ gives us a divergence at s in $\sigma;\tau$. In the second case, an infinite chattering between σ and τ corresponds to an infinite interaction $\mathfrak{s} \in \text{int}_{\infty}(A, B, C)$ (with a tail in B) such that $\mathfrak{s}|_{A,C} = s$. Completeness for σ and τ tells us that $\mathfrak{s}|_{A,B} \in D_{\sigma}$ and $\mathfrak{s}|_{B,C} \in D_{\tau}$ and therefore that $\mathfrak{s}|_{A,C} \in D_{\sigma;\tau}$. ◀

3.2.5 Associativity of composition

The proof of associativity of composition is the same in our model as it is in any other model of game semantics if we treat infinite plays the same as finite ones. However, it is worth saying a few words about associativity, since the model obtained by naively extending the Harmer-McCusker model to unbounded nondeterminism does not have an associative composition. The point is that there is not really a problem with associativity itself, but rather that this naive model gives the wrong result for the composition of strategies with infinite nondeterminism. For example, if ν is the strategy we defined above, and 0 is the ‘constant 0’ strategy on \mathbb{N} , then $0;\nu$ has a divergence in the naive model, because the strategies 0 and ν appear to be engaged in infinite chattering. In our model, on the other hand, the strategy ν contains no infinite plays, and so no divergences arise in the composition.

3.3 A symmetric monoidal closed category

Given a game A , we define a strategy id_A on $A \multimap A$, where T_{id_A} is given by

$$\{s \in P_{A_1 \multimap A_2} : \text{for all even-length } t \sqsubseteq s, t|_{A_1} = t|_{A_2}\},$$

where we distinguish between the two copies of A by calling them A_1 and A_2 , and where D_{σ} is the set of all infinite plays in T_{σ} . This is an identity for the composition we have defined, and so we get a category \mathcal{G}_{ND} of games and nondeterministic strategies. Moreover, the connectives \otimes and \multimap exhibit \mathcal{G}_{ND} as a symmetric monoidal closed category.

\mathcal{G}_{ND} has an important subcategory \mathcal{G}_D of deterministic complete strategies; this category is isomorphic to the category considered in [3].

3.4 A Cartesian closed category

We follow the construction given in [3], using the connectives $!$ and \times to build a Cartesian closed category $\mathcal{G}_{ND}^!$ from \mathcal{G}_{ND} whose objects are the well-opened games in \mathcal{G}_{ND} and where a morphism from A to B in $\mathcal{G}_{ND}^!$ is a morphism from $!A$ to B in \mathcal{G}_{ND} .

This is similar to the construction of a co-Kleisli category for a linear exponential comonad, but technical issues relating to well-openedness prevent us from presenting it in this way.

3.5 Constraining strategies

Given a non-empty justified string s , we define the P -view $\ulcorner s \urcorner$ of s inductively as follows.

$$\begin{aligned} \ulcorner sm \urcorner &= m, & \text{if } m \text{ is initial;} \\ \ulcorner sntm \urcorner &= \ulcorner s \urcorner nm, & \text{if } m \text{ is an } O\text{-move and} \\ & & n \text{ justifies } m; \\ \ulcorner sm \urcorner &= \ulcorner s \urcorner m, & \text{if } m \text{ is a } P\text{-move.} \end{aligned}$$

We say that a play sm ending in a P -move is P -visible if the justifier of m is contained in $\ulcorner m \urcorner$. We say that a strategy σ for a game A is *visible* if every P -position $s \in T_\sigma$ is P -visible. It can be shown that the composition of visible strategies is visible, and that we can build a Cartesian closed category using our exponential.

After passing to the intrinsic quotient, the resulting category $\mathcal{G}_{D,vis}^!$ of games and deterministic visible strategies is a fully abstract model of Idealized Algol [3].

3.6 Recursive strategies

Most full abstraction results go via a definability result that says that all *compact* strategies are definable [6]. However, deducing full abstraction from compact definability makes essential use of continuity properties that are absent when we deal with countable nondeterminism. We will therefore need to appeal to a stronger result – that of *universality*, which states that *every* strategy is definable. Clearly, universality does not hold for any of our categories of games – for example, there are many non-computable functions $\mathbb{N} \rightarrow \mathbb{N}$. However, Hyland and Ong proved in [10] that every *recursively presentable* innocent strategy is PCF-definable.

If σ is a complete strategy for a game A , we say σ is *recursive* if $T_\sigma \cap P_A$ and $D_\sigma \cap P_A$ are recursively enumerable subsets of ω^ω (under the enumeration of M_A). Here, we throw away the infinite plays in T_σ and D_σ , but we do not lose any information because σ is complete.

If σ is lively, we say that σ is *recursive*, and if σ is the union of complete recursive strategies $\sigma_1, \sigma_2, \dots$, where the map $i \mapsto \sigma_i$ is a recursive function $\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow 2$.

Note that there are plenty of strategies that we want that are not the union of a recursive sequence of *deterministic* strategies – for example, the strategy on $(\mathbb{N} \rightarrow \mathbb{C}) \rightarrow \mathbb{C}$ that calls its natural-number argument infinitely many times is complete and has no O -branching, but its infinite traces include every recursive sequence of natural numbers.

Using these definitions, it seems to be hard to prove that the composition of recursive strategies is itself a recursive strategy: the tricky point is to show that the decomposition into complete strategies may still be taken to be given by a recursive strategy. The example in the previous paragraph shows that we cannot use the same proof as we did in the non-recursive case, which used deterministic strategies. Fortunately, we do not need to be able to show that the composition of recursive strategies is recursive in order to prove our full abstraction result, so we leave this problem for future work.

In the case that σ is recursive and *deterministic*, we can prove the following result.

► **Proposition 6** (Recursive Universality for Idealized Algol). *Let S be an Idealized Algol type and let $\sigma: \llbracket S \rrbracket$ be a recursive deterministic strategy. Then there exists a term $M: S$ of Idealized Algol such that $\sigma = \llbracket M \rrbracket$.*

Proof. We use the ‘innocent factorization’ result of [3] to reduce to the innocent case and then proceed in a manner similar to the argument used in [16]. ◀

Note that Proposition 6 is sharper than the result in [10], which only proves that every recursive strategy may be defined *up to observational equivalence*. Idealized Algol allows us to store variables and then use them multiple times without having to read them again, which allows us to define all recursive visible strategies exactly. Compare with [16], which proves a similar result for *call-by-value* PCF.

3.7 Deterministic Factorization

Our definability results will hinge on a *factorization theorem*, showing that every non-deterministic strategy may be written as the composition of a deterministic strategy with the nondeterministic ‘oracle’ $\top_{\mathbb{N}}$. We can then deduce universality from universality in the model of deterministic Idealized Algol.

Note that our result is a bit simpler than in [8] because of the unbounded nondeterminism.

► **Proposition 7.** *Let $\sigma: I \rightarrow A$ be a strategy for a game A in \mathcal{G}_{ND} . Then we may write σ as $\top_{\mathbb{N}}; \text{Det}(\sigma)$, where $\text{Det}(\sigma): !\mathbb{N} \rightarrow A$ is a deterministic strategy and $\top_{\mathbb{N}}: \mathbb{N}$ is the strategy that contains every play in $!\mathbb{N}$ and has no finite divergences.*

Proof. We begin by fixing an injection code_A from the set of P -moves in A into the natural numbers. In the enumerated case, this is given to us already.

We first assume that the strategy σ is complete. Then the strategy $\text{Det}(\sigma)$ is very easy to describe. For each O -position $s \in T_\sigma$, we have some set B of possible replies to s , which we order as b_1, b_2, \dots , where $\text{code}_A(b_1) < \text{code}_A(b_2) < \dots$. We insert a request to the oracle for a natural number; then, depending on her answer j , we play the next move as follows:

- If $0 < j \leq \text{code}_A(b_1)$, then play b_1 .
- If $\text{code}_A(b_n) < j \leq \text{code}_A(b_{n+1})$ then play b_{n+1} .
- If $j = 0$ and $s \in D_\sigma$, then play nothing, and put the resulting play inside $D_{\text{Det}(\sigma)}$. Otherwise, play b_1 .

We close under limits to make the strategy $\text{Det}(\sigma)$ complete. $\text{Det}(\sigma)$ is clearly deterministic. Checking that $\top_{\mathbb{N}}; \text{Det}(\sigma) = \sigma$ is easy for finite plays; for infinite plays, it follows by completeness of σ .

Lastly, if σ is the union of complete strategies $\sigma_1, \sigma_2, \dots$, we insert an additional request to the oracle immediately after the very first move by player O ; after receiving a reply k , we play according to σ_k . ◀

Note that $\text{Det}(\sigma)$ is recursive if σ is and is visible if σ is.

4 Full abstraction

4.1 Denotational Semantics

The category in which we shall model our language is the category $\mathcal{G}_{ND,vis}^!$ – the Cartesian closed category of (enumerated) games with nondeterministic visible strategies. We have a natural embedding $\mathcal{G}_{D,vis}^! \hookrightarrow \mathcal{G}_{ND,vis}^!$, and we know that $\mathcal{G}_{D,vis}^!$ is a universal and fully abstract model of Idealized Algol.

We model the language compositionally, using denotations as in [8] for the nondeterministic constants and modelling ? using the strategy $\top_{\mathbb{N}}: \mathbb{N}$.

Any term $M : T$ of Idealized Algol with countable nondeterminism may be written as $M = C[?]$, where C is a multi-holed context not involving the constant $?$. Then the term $\lambda n.C[n]$ is a term of Idealized Algol, and therefore has a denotation $!N \rightarrow \llbracket T \rrbracket$ as in [3].

► **Lemma 8.** *The term $C[?]$ has the same denotation as the term $(\lambda n.C[n])?$.*

Proof. This is a straightforward argument by structural induction on C , and the constant \top_N does not really play a role. We prove inductively on T that if $\Gamma \vdash C[?]: T$ is a term-in-context, then its denotation may be given by the following composite.

$$\llbracket \Gamma \rrbracket \xrightarrow{\text{lunit}; (\top_N \times \text{id})} N \times \llbracket \Gamma \rrbracket \xrightarrow{\llbracket \Gamma, n: \text{nat} \vdash C[n]: T \rrbracket} \llbracket T \rrbracket \quad \blacktriangleleft$$

4.2 Computational Adequacy

The *computational adequacy* result for our model can be stated as follows.

► **Proposition 9** (Computational Adequacy). *Let $M : \text{com}$ be a closed term of nondeterministic Idealized Algol. $M \Downarrow \text{skip}$ if and only if $qa \in T_{\llbracket M \rrbracket}$. $M \Downarrow^{\text{must}}$ if and only if $D_{\llbracket M \rrbracket} = \emptyset$.*

Traditional proofs of computational adequacy using logical relations make essential use of the continuity of composition with respect to a natural ordering on strategies (see, for example, [8] and [9] for the finite nondeterminism case). In our case, since composition is not continuous in the language itself, we cannot use this technique. In order to prove adequacy, we use a new technique that involves using a deterministic stateful construction to model the nondeterminism inside a deterministic world in which continuity holds. To do this, we shall return to the concept of an *evaluation* π of a term as a sequence of natural numbers encoding the nondeterministic choices that we have made.

► **Lemma 10.** *Let $M = C[?]$ be a term of type com , where $C[-]$ is a multi-holed context of (deterministic) Idealized Algol. Write σ_M for the denotation of the term $\lambda n.C[n]$.*

- *If $M \Downarrow \text{skip}$ then there exists some total deterministic strategy $\sigma : !N$ such that $qa \in T_{\sigma; \sigma_M}$.*
- *If $M \Downarrow^{\text{must}}$ then there exists some total deterministic strategy $\sigma : !N$ such that $D_{\sigma; \sigma_M} \neq \emptyset$.*

Proof. Let n_1, \dots, n_k, d be a finite sequence of natural numbers. We define an Idealized Algol term $N_{n_1, \dots, n_k, d}: (\text{nat} \rightarrow \text{com}) \rightarrow \text{com}$ to be the following.

$$\lambda f.\text{new}_{\text{nat}}(\lambda v.f(v := (\text{suc } @v); \text{case}_{k+1} @v \Omega n_1 \dots n_k d)).$$

Here, $\text{case}_{k+1} a n_0 \dots n_k d$ is a new shorthand that evaluates to n_i if a evaluates to i , and evaluates to d if a evaluates to $j > k$. This term calls the function f , passing in n_1 the first time, n_2 the second and so on, passing in d at every call beyond $k + 1$.

Now let π be a finite evaluation of $\langle s, C[?] \rangle$ that converges to skip . Encode π as a sequence n_1, \dots, n_k . Let d be some arbitrary number. Then we can show that the following term also converges to skip in the same way:

$$N_{n_1, \dots, n_k, d}(\lambda n.C[n]).$$

The idea here is similar to one used in testing; we want to test the behaviour of a non-deterministic program, and to do so we *mock* the random number generator in order to simulate a particular evaluation path using purely deterministic programs.

If instead π is a finite evaluation of $\langle s, C[?] \rangle$ that diverges (but nevertheless only involves finitely many calls to the nondeterministic oracle), then the term $N_{n_1, \dots, n_k, d}(\lambda n.C[n])$ will diverge according to the same execution path.

24:14 Game Semantics for Countable Nondeterminism

Digging into the construction of **new** within Idealized Algol, as given in [3], we see that for any term F of type $\mathbf{nat} \rightarrow \mathbf{com}$ the denotation of $N_{n_1, \dots, n_k, d} F$ is given by the composite

$$I \xrightarrow{\text{cell}_0} !\mathbf{Var} \xrightarrow{![\lambda v.v := (\text{succ } \text{?}v); \text{case}_{k+1} \text{?}v \Omega n_1 \dots n_k d]} !\mathbb{N} \xrightarrow{[F]} \mathbb{C}.$$

We set σ_π to be the composite of the left two arrows. Observe that σ_π is the strategy with unique maximal infinite play as follows.

$$q n_1 \dots q n_k q d q d \dots$$

Setting $F = \lambda n.C[n]$, we see that $[F] = \sigma_M$. So, by adequacy for the Idealized Algol model, $qa \in T_{\sigma_\pi; \sigma_M}$ if and only if we have $N_{n_1, \dots, n_k, d}(\lambda n.C[n]) \Downarrow \mathbf{skip}$, which is the case if and only if $M \Downarrow \mathbf{skip}$ along the evaluation π . Similarly, $D_{\sigma_\pi; \sigma_M} \neq \emptyset$ if and only if $N_{n_1, \dots, n_k, d}(\lambda n.C[n])$ diverges, which is equivalent to saying that M diverges along the evaluation π .

Lastly, we need to deal with the case that there is an infinite evaluation $\pi = n_1, n_2, \dots$ of M that consults the nondeterministic oracle infinitely often. In this case, M must certainly diverge along the evaluation π . For each j , we define $\pi_n^{(j)}$ to be the strategy for $!\mathbb{N}$ corresponding to the term $N_{n_1, \dots, n_j, \Omega}$. So $\pi_n^{(j)}$ has a unique finite maximal play

$$q n_1 q n_2 \dots q n_j q,$$

at which point the strategy has a partiality.

Evaluation of the term $N_{n_1, \dots, n_j, \Omega}(\lambda n.C[n])$ must diverge, since it will proceed according to the evaluation π and eventually reach the divergence (since π consults the oracle infinitely often). This implies that $D_{\pi_n^{(j)}; \sigma_M} \neq \emptyset$ for all j .

We define σ_π to be the least upper bound of the $\sigma_\pi^{(j)}$ (e.g., in the sense of [8]). Since composition is continuous for deterministic (!) strategies, we deduce that $D_{\sigma_\pi; \sigma_M} \neq \emptyset$.

σ_π has plays of the form $q n_1 q n_2 \dots$, and so it is total. \blacktriangleleft

From the proof of this result, we can establish the converse, which we will also need.

► **Lemma 11.** *Let $M = C[?]$ be as before. Let $\sigma : !\mathbb{N}$ be a total deterministic strategy.*

■ *If $qa \in T_{\sigma; \sigma_M}$ then $M \Downarrow \mathbf{skip}$.*

■ *If $D_{\sigma; \sigma_M} \neq \emptyset$ then $M \Downarrow^{must}$.*

Proof. Since σ is total and deterministic, it must have a maximal infinite play s_σ of the form $q m_1 q m_2 \dots$, where m_1, m_2, \dots is some infinite sequence of natural numbers. If the strategy σ_M contains some play \mathfrak{s} such that $\mathfrak{s}|_{!\mathbb{N}} = s$, then $\sigma = \sigma_\pi$ for some infinite evaluation π of M . Otherwise, let t be the maximal sub-play of s such that $\mathfrak{s}|_{!\mathbb{N}} = t$ for some $\mathfrak{s} \in \sigma_M$. Then, if we replace σ with the strategy σ' that plays according to t and subsequently plays $q d q d \dots$ for our fixed value d , we will have $\sigma'; \sigma_M = \sigma; \sigma_M$. In either case, $\sigma' = \sigma_\pi$ for some evaluation π of the term M .

Now suppose that there exists $\sigma : !\mathbb{N}$ such that $qa \in T_{\sigma; \sigma_M}$. We may assume that $\sigma = \sigma_\pi$ for some evaluation π of M . Therefore, $qa \in T_{\sigma_\pi; \sigma_M}$, which means that $M \Downarrow \mathbf{skip}$ along π . The corresponding statement for must convergence follows in the same way. \blacktriangleleft

Note that these last two lemmas may be cast entirely in the model of *deterministic* Idealized Algol given in [3], since they only refer to the denotations of deterministic terms. We can therefore prove a more general version of Proposition 9.

► **Definition 12.** Let $\sigma : A \rightarrow B$ be a (deterministic) strategy. We say that σ is *winning* if every play in σ may be extended to a play that ends with a P -move in B ; i.e., σ is total and contains no sequences having an infinite tail in A .

This definition is motivated by Lemmas 10 and 11 in the following sense: if $\sigma_M : \mathbb{N} \rightarrow \mathbb{C}$ is a strategy, then there exists some σ_M such that $D_{\sigma; \sigma_M} \neq \emptyset$ if and only if σ is not winning.

The following is now an easy corollary of Lemmas 10 and 11.

► **Corollary 13.** *Let \mathcal{C} be a Cartesian closed category that admits a faithful Cartesian functor $J : \mathcal{G}_{vis}^1 \hookrightarrow \mathcal{C}$. Let $\top_{\mathbb{N}} : 1 \rightarrow J\mathbb{N}$ be a morphism in \mathcal{C} and use it to extend the semantics of Idealized Algol of \mathcal{G}_{vis}^1 to a semantics of nondeterministic Idealized Algol, as in Section 4.1.*

Suppose we have two predicates $\Downarrow \text{skip}$ and \Downarrow^{must} defined on strategies $1 \rightarrow J\mathbb{C}$ in \mathcal{C} satisfying the following rules for all strategies $\sigma : \mathbb{N} \rightarrow \mathbb{C}$ in \mathcal{G}_{vis}^1 .

- $(\top_{\mathbb{N}}; J\sigma) \Downarrow \text{skip}$ if and only if there is some $s \in \sigma$ such that $s|_{\mathbb{C}} = qa$.
- $(\top_{\mathbb{N}}; J\sigma) \Downarrow^{must}$ if and only if σ is winning.

Then the semantics of nondeterministic Idealized Algol inside \mathcal{C} is adequate in the following sense. For all terms M of nondeterministic Idealized Algol of type com :

- $M \Downarrow \text{skip}$ if and only if $\llbracket M \rrbracket \Downarrow \text{skip}$.
- $M \Downarrow^{must}$ if and only if $\llbracket M \rrbracket \Downarrow^{must}$.

We can then deduce Proposition 9 by verifying that the following predicates on strategies $\sigma : 1 \rightarrow \mathbb{C}$ in the category $\mathcal{G}_{ND, vis}^1$ satisfy the conditions of Corollary 13.

- $\sigma \Downarrow \text{skip} \Leftrightarrow qa \in T_{\sigma}$.
- $\sigma \Downarrow^{must} \Leftrightarrow D_{\sigma} = \emptyset$.

Corollary 13 is very general, and is intended to be applied in multiple situations. In particular, it may be applied to a game semantics in which we define a ‘nondeterministic visible strategy’ on a game A to be a deterministic visible strategy for $\mathbb{N} \rightarrow A$, up to a suitable equivalence relation. This model is an example of a much more general construction that is the subject of ongoing research by the authors. In this sense, our main model based on nondeterministic strategies is not necessary in order to obtain our full abstraction result. Nevertheless, we felt it important to give a model based on nondeterministic strategies, since these are the ‘natural’ game semantic interpretation of nondeterminism.

4.3 Intrinsic Equivalence and Soundness

We define *intrinsic equivalence of strategies* as follows. If σ, τ are two strategies for a game A , we say that $\sigma \sim \tau$ if for all test morphisms $\alpha : A \rightarrow \mathbb{C}$ we have $\sigma; \alpha = \tau; \alpha$. Having defined this equivalence, we may prove *soundness* in the usual way.

► **Theorem 14 (Soundness).** *Let M, N be two closed terms of type T . If $\llbracket M \rrbracket \sim \llbracket N \rrbracket$ then $M \equiv_{m\&m} N$.*

For full abstraction, we need to take the intrinsic quotient in order to identify, for example, the terms $\lambda n. \Omega$ and $\lambda n. \text{If } 0 \ n \ \Omega : \text{nat} \rightarrow \text{nat}$. These terms are observationally equivalent, but their denotations are not equal; for example, $q \in D_{\llbracket \lambda n. \Omega \rrbracket}$, but $q \notin D_{\llbracket \lambda n. \text{If } 0 \ n \ \Omega \rrbracket}$.

The point here is that even though q is not explicitly a divergence in the second case, it is nonetheless impossible to prevent the strategy from eventually reaching a divergence.

Given a nondeterministic strategy σ for a game A , we may treat σ as a game in its own right (a sub-game of A). Moreover, for any $s \in T_{\sigma}$, we have a particular branch of that game in which play starts at s . We say that s is *unreliable* if player P has a strategy for the game starting at s that ensures that the (possibly infinite) limiting play is in D_{σ} .

We then say that a strategy σ is *divergence-complete* if every unreliable point of σ is contained in D_{σ} . Every strategy σ can clearly be extended to a minimal divergence-complete strategy $\text{dc}(\sigma)$; Murawski’s explicit characterization of the intrinsic collapse [15], which may

be applied to our model, essentially says that $\sigma \sim \tau$ if and only if σ and τ have the same *complete* plays and $\text{dc}(\sigma) = \text{dc}(\tau)$.

An important fact about intrinsic equivalence is the following Lemma, whose proof makes use of the fact that the infinite plays in our strategies are given by recursive functions.

► **Lemma 15.** *Let σ, τ be strategies for a game A . Suppose that $\sigma; \alpha = \tau; \alpha$ for all recursive strategies $\alpha: A \rightarrow \mathbb{C}$. Then $\sigma \sim \tau$.*

4.4 Universality

Let S, T be Idealized Algol types and let $\sigma: S \rightarrow T$ be a recursive morphism in $\mathcal{G}_{ND,vis}^1$. We want to prove that σ is the denotation of some term.

By our nondeterministic factorization result, we know that $\sigma = \top_{\mathbb{N}}; \text{Det}(\sigma)$, where $\text{Det}(\sigma)$ is a deterministic recursive strategy. By universality for $\mathcal{G}_{D,vis}^1$, we know that $\text{Det}(\sigma) = \llbracket M \rrbracket$ for some closed term $M: S \rightarrow T$. Then $\sigma = \top_{\mathbb{N}}; \text{Det}(\sigma) = \llbracket ? \rrbracket; \llbracket M \rrbracket = \llbracket M ? \rrbracket$.

4.5 Full abstraction

► **Theorem 16** (Full abstraction). *Let M, N be two closed terms of type T . If $M \equiv_{m\&m} N$ then $\llbracket M \rrbracket \sim \llbracket N \rrbracket$.*

Proof. Let $A = \llbracket T \rrbracket$. Suppose that $\llbracket M \rrbracket \not\sim \llbracket N \rrbracket$; so there is some strategy $\alpha: A \rightarrow \mathbb{C}$ such that $\llbracket M \rrbracket; \alpha \neq \llbracket N \rrbracket; \alpha$. By Lemma 15, we can choose α to be recursively presentable; by universality, we have $\alpha = \llbracket P \rrbracket$ for some closed term P of type $T \rightarrow \text{com}$. Then we have $\llbracket M \rrbracket; \llbracket P \rrbracket \neq \llbracket N \rrbracket; \llbracket P \rrbracket$; by computational adequacy, it follows that $M \not\equiv_{m\&m} N$. ◀

5 Conclusion

We conclude by making a few remarks about the situation when our base deterministic language is PCF rather than Idealized Algol.

The principal difficulties in modelling nondeterministic stateless languages were overcome by Tsukada and Ong in [21], where they outlined how to define an innocent nondeterministic strategy by retaining ‘branching time information’ in strategies. An additional benefit of the retention of branching time information is that we no longer need to keep track of infinite plays in order to model unbounded nondeterminism. Tsukada and Ong’s primary model was based on sheaves over a site of plays, but they also give a more direct way of characterizing nondeterministic innocence, based on ideas by Levy [14].

The model given in [21] is not sound for must-equivalence, but the authors make the claim that their model may be easily modified to yield a model that *is* sound for this type of equivalence, using the same techniques from [8] that we have used.

We could use our methods to help establish this claim in the case of unbounded nondeterminism; specifically, our proof of adequacy will extend to such a model. Indeed, Corollary 13 can easily be modified to apply to PCF, even though we have used Idealized Algol terms in the proof. Corollary 13 then reduces the proof of adequacy to a combinatorial check on morphisms from $\mathbb{N} \rightarrow \mathbb{C}$ on strategies in the well-known category \mathcal{G}_{vis}^1 , together with an examination of what happens to those strategies when we compose them with $\top_{\mathbb{N}}$.

References

- 1 Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *The Journal of Symbolic Logic*, 59(2):543–574, 1994. URL: <http://arxiv.org/abs/1311.6057>.
- 2 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, 2000. doi:10.1006/inco.2000.2930.
- 3 Samson Abramsky and Guy McCusker. Linearity, sharing and state: a fully abstract game semantics for idealized algol with active expressions: Extended abstract. *Electronic Notes in Theoretical Computer Science*, 3:2–14, 1996. Linear Logic 96 Tokyo Meeting. doi:10.1016/S1571-0661(05)80398-6.
- 4 K. R. Apt and G. D. Plotkin. A cook’s tour of countable nondeterminism. In Shimon Even and Oded Kariv, editors, *Automata, Languages and Programming*, pages 479–494, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg.
- 5 Simon Castellan, Pierre Clairambault, and Glynn Winskel. Concurrent Hyland-Ong games. working paper or preprint, 2016. URL: <https://hal.archives-ouvertes.fr/hal-01068769>.
- 6 Pierre-Louis Curien. Definability and full abstraction. *Electronic Notes in Theoretical Computer Science*, 172:301–310, 2007. Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin. doi:10.1016/j.entcs.2007.02.011.
- 7 Edsger Wybe Dijkstra. *A Discipline of Programming*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1997.
- 8 R. Harmer and G. McCusker. A fully abstract game semantics for finite nondeterminism. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 422–430, 1999. doi:10.1109/LICS.1999.782637.
- 9 Russell S. Harmer. Games and full abstraction for nondeterministic languages. Technical report, University of London/Imperial College, 1999.
- 10 J.M.E. Hyland and C.-H.L. Ong. On full abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, 2000. doi:10.1006/inco.2000.2917.
- 11 J. Laird. Sequential algorithms for unbounded nondeterminism. *Electronic Notes in Theoretical Computer Science*, 319:271–287, 2015. doi:10.1016/j.entcs.2015.12.017.
- 12 James Laird. Higher-order programs as coroutines. to appear, 2016.
- 13 Paul Blain Levy. Infinite trace equivalence. *Annals of Pure and Applied Logic*, 151(2):170–198, 2008. doi:10.1016/j.apal.2007.10.007.
- 14 Paul Blain Levy. Morphisms between plays. Lecture slides, GaLoP, 2014.
- 15 A. S. Murawski. Reachability games and game semantics: Comparing nondeterministic programs. In *23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008)(LICS)*, volume 00, pages 353–363, 06 2008. doi:10.1109/LICS.2008.24.
- 16 Andrzej S. Murawski and Nikos Tzevelekos. Block structure vs scope extrusion: between innocence and omniscience. *Logical Methods in Computer Science*, 12(3), 2016. doi:10.2168/LMCS-12(3:3)2016.
- 17 David Park. On the semantics of fair parallelism. In *Abstract Software Specifications*, pages 504–526. Springer, 1980.
- 18 David Park. Concurrency and automata on infinite sequences. In *Theoretical computer science*, pages 167–183. Springer, 1981.
- 19 Corin Pitcher. *Functional programming and erratic non-determinism*. PhD thesis, University of Oxford/Trinity College, 2001.
- 20 A. W. Roscoe. Unbounded non-determinism in CSP. *Journal of Logic and Computation*, 3(2):131, 1993. doi:10.1093/logcom/3.2.131.

24:18 Game Semantics for Countable Nondeterminism

- 21 Takeshi Tsukada and C. H. Luke Ong. Nondeterminism in game semantics via sheaves. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, LICS '15, pages 220–231, Washington, DC, USA, 2015. IEEE Computer Society. doi:10.1109/LICS.2015.30.
- 22 Glynn Winskel. Strategies as profunctors. In Frank Pfenning, editor, *Foundations of Software Science and Computation Structures*, pages 418–433, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

Dependency Concepts up to Equivalence

Erich Grädel¹

Mathematical Foundations of Computer Science, RWTH Aachen University, Aachen, Germany
graedel@logic.rwth-aachen.de

Matthias Hoelzel²

Mathematical Foundations of Computer Science, RWTH Aachen University, Aachen, Germany
hoelzel@logic.rwth-aachen.de

Abstract

Modern logics of dependence and independence are based on different variants of atomic dependency statements (such as dependence, exclusion, inclusion, or independence) and on team semantics: A formula is evaluated not with a single assignment of values to the free variables, but with a set of such assignments, called a team.

In this paper we explore logics of dependence and independence where the atomic dependency statements cannot distinguish elements up to equality, but only up to a given equivalence relation (which may model observational indistinguishabilities, for instance between states of a computational process or between values obtained in an experiment).

Our main goal is to analyse the power of such logics, by identifying equally expressive fragments of existential second-order logic or greatest fixed-point logic, with relations that are closed under the given equivalence. Using an adaptation of the Ehrenfeucht-Fraïssé method we further study conditions on the given equivalences under which these logics collapse to first-order logic, are equivalent to full existential second-order logic, or are strictly between first-order and existential second-order logic.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Logics of dependence and independence, Team semantics, Existential second-order logic, Observational equivalence, Expressive power

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.25

1 Introduction

Logics of dependence and independence (sometimes called logics of imperfect information) originally go back to the work of Henkin [9], Enderton [2], Walkoe [16], Blass and Gurevich [1], and others on Henkin quantifiers, whose semantics can be naturally described in terms of games of imperfect information. A next step in this direction have been the independence-friendly (IF) logics by Hintikka and Sandu [10] that incorporate explicit dependencies of quantifiers on each other and where again, the semantics is usually given in game-theoretic terms. For a detailed account on independence-friendly logics we refer to [13].

An important achievement towards the modern framework for logics of dependence and independence has been the model-theoretic semantics for IF-logics, due to Hodges [11], in terms of what he called *trumps*. This semantics is today called *team semantics*, where a *team* is understood as a set of assignments $s : \mathcal{V} \rightarrow A$, mapping a common finite domain of variables into the universe of a structure. The next step towards modern logics of

¹ This work has been initiated in a discussion between the first author and Jouko Väänänen during the Logical Structures in Computation Programme at the Simons Institute for Computing at UC Berkeley.

² Supported by DFG.



dependence and independence was the proposal by Väänänen [15] to consider dependencies as *atomic properties of teams* rather than stating them via annotations of quantifiers. He first introduced *dependence logic*, which is first-order logic on teams together with dependence atoms $\text{dep}(x_1, \dots, x_m, y)$, saying that, in the given team, the variable y is functionally dependent on (i.e. completely determined by) the variables x_1, \dots, x_m . But there are many other atomic dependence properties that give rise to interesting logics based on team semantics. In [8] we have discussed the notion of independence (which is a much more delicate but also more powerful notion than dependence) and introduced *independence logics*, and Galliani [5] and Engström [3] have studied several logics with team properties based on notions originating in database dependency theory. The most important ones are *inclusion logic* $\text{FO}(\subseteq)$, which extends first-order logic by atomic inclusion dependencies $(\bar{x} \subseteq \bar{y})$, which are true in a team X if every value for \bar{x} in X also occurs as a value for \bar{y} in X , and *exclusion logic*, based on exclusion statements $(\bar{x} \mid \bar{y})$, saying that \bar{x} and \bar{y} have disjoint sets of values in the team X . Exclusion logic has turned out to be equivalent to dependence logic [5].

Altogether this modern framework has led to a genuinely new area in logic, with an interdisciplinary motivation of providing logical systems for reasoning about the fundamental notions of dependence and independence that permeate many scientific disciplines. Methods from several areas of computer science, including finite model theory, database theory, and the algorithmic analysis of games have turned out as highly relevant for this area. For more information, we refer to the volume [4] and the references there.

In this paper we explore logics that are based on weaker variants of dependencies. We consider atomic dependence statements that do not distinguish elements up to equality, but only up to coarser equivalencies. This is motivated by the quite familiar situation in many applications that elements, such as for instance states in a computation or values obtained in experiments, are subject to observational indistinguishabilities, which we model here via an equivalence relation \approx on the set of possible values. For *dependence atoms* $\text{dep}_{\approx}(\bar{x}, y)$ this means that we can say that whenever the values of \bar{x} are indistinguishable for certain assignments in a team, then so are the values of y . Similarly an exclusion statement between x and y , up to an equivalence relation \approx , says that no value for x in the team is equivalent to a value of y , and an *inclusion statement* $x \subseteq_{\approx} y$ means that every value for x is equivalent to some value for y . Finally, the most powerful of such notions, independence of x and y up to equivalence, means that additional information about the equivalence class of the value of one variable does not help to learn anything new about the value of the other, or to put it differently, whenever a value a for x and a value b for y occur in the team, then there is an assignment in the team whose value for x is equivalent to a and whose value for y is equivalent to b .

Formal definitions of these dependencies, extended to tuples of variables, will be given in the next section.

The main goal of this paper is to *understand the expressive power of the logics with dependencies up to equivalence*. In general, logical operations on teams have a second-order nature, and indeed, dependencies and team semantics may take the power of first-order logic FO up to existential second-order logic Σ_1^1 . To make this precise we recall the *standard translation*, due to [15, 12], from formulae with team semantics into sentences of existential second-order logic.

First of all, we have to keep in mind the different nature of team semantics and classical Tarski semantics. For a formula with team semantics, we write $\mathfrak{A} \models_X \varphi$ to denote that φ is true in the structure \mathfrak{A} for the team X , and for classical Tarski semantics we write $\mathfrak{A} \models_s \varphi$ to denote that φ is true in \mathfrak{A} for the assignment s .

For formulae with free variables the translation from a logic with team semantics into one with Tarski semantics requires that we represent the team in some way. The standard way to do this is by identifying a team X of assignments $s : \{x_1, \dots, x_k\} \rightarrow A$ with the relation $\{s(x_1, \dots, x_k) \in A^k : s \in X\} \subseteq A^k$ which, by slight abuse of notation, we also denote by X . One then translates formulae $\varphi(x_1, \dots, x_k)$ of vocabulary τ of a logic L with team semantics into *sentences* φ^* in Σ_1^1 of the expanded vocabulary $\tau \cup \{X\}$ such that for every structure \mathfrak{A} and every team X we have that

$$\mathfrak{A} \models_X \varphi(x_1, \dots, x_k) \iff (\mathfrak{A}, X) \models \varphi^*.$$

To illustrate this second-order nature we recall the meaning of disjunctions and existential quantifications in team semantics, and their standard translation into Σ_1^1 . Disjunctions split the team, i.e.,

$$\mathfrak{A} \models_X \psi \vee \varphi \iff X = Y \cup Z \text{ such that } \mathfrak{A} \models_Y \psi \text{ and } \mathfrak{A} \models_Z \varphi$$

which leads to the translation $(\psi \vee \varphi)^*(X) := \exists Y \exists Z (X = Y \cup Z \wedge \psi^*(Y) \wedge \varphi^*(Z))$. Existential quantification requires the extension of the given team by providing for each of its assignments a *non-empty set of witnesses* for quantified variables, i.e.,

$$\mathfrak{A} \models_X \exists y \psi \iff \text{there exists a function } F : X \rightarrow \mathcal{P}^+(A) \text{ such that } \mathfrak{A} \models_{X[y \mapsto F]} \psi$$

where $X[y \mapsto F]$ is the set of all assignments $s[y \mapsto a]$ that update an assignment $s \in X$ by mapping y to some value $a \in F(s)$. This leads to the translation $(\exists y \psi)^*(X) := \exists Y \forall \bar{x} ((X\bar{x} \leftrightarrow \exists y Y \bar{x}y) \wedge \psi^*(Y))$.

Some remarks are in order: One may wonder why it is appropriate to provide a non-empty set of witnesses for an existentially quantified variable rather than just a single witness as in standard Tarski semantics for first-order logic. Indeed there are many cases where a single witness, i.e. a function $F : X \rightarrow A$ rather than $F : X \rightarrow \mathcal{P}^+(A)$ suffices, in fact in all cases where the logic is *downwards closed*, i.e. when $\mathfrak{A} \models_X \psi$ implies that also $\mathfrak{A} \models_Y \psi$ for all subteams $Y \subseteq X$. Examples of downwards closed logics are dependence logic and exclusion logic. However, for logics that are not downwards closed, such as inclusion logic and independence logic, the so-called strict semantics requiring single witnesses of existentially quantified variables leads to pathologies such as non-locality: the meaning of a formula might depend on the values of variables that do not even occur in it. A second relevant remark is that all the logics considered here have the *empty team property*: For all sentences φ and all structures \mathfrak{A} , we have that $\mathfrak{A} \models_{\emptyset} \varphi$. To evaluate sentences (formulae without free variables) we therefore have to consider not the empty team, but the team $\{\emptyset\}$ consisting just of the empty assignment. For a sentence ψ we write $\mathfrak{A} \models \psi$ if $\mathfrak{A} \models_{\{\emptyset\}} \psi$.

On the basis of the standard translation in Σ_1^1 we can say that we understand the expressive power of a first-order logic with dependencies, when we have identified the fragment \mathcal{F} of existential second-order logic which is equivalent in the sense just described. The following is known in this context:

- (1) Dependence logic and exclusion logic are equivalent to the fragment of all Σ_1^1 -sentences $\psi(X)$ in which the predicate X describing the team appears only negatively [12].
- (2) Independence logic and inclusion-exclusion logic are equivalent with full Σ_1^1 (and thus can describe all NP-properties of teams) [5].
- (3) The extension of FO by inclusion and exclusion atoms of single variables only (not tuples of variables) is equivalent to monadic Σ_1^1 [14].

- (4) First-order logic without any dependence atoms has the so-called *flatness property*: $\mathfrak{A} \models_X \varphi \iff \mathfrak{A} \models_s \varphi$ for all $s \in X$. It thus corresponds to a very small fragment of Σ_1^1 , namely FO-sentences of form $\forall \bar{x}(X\bar{x} \rightarrow \varphi(\bar{x}))$ where $\varphi(\bar{x})$ does not contain X .
- (5) Inclusion logic $\text{FO}(\subseteq)$ corresponds to GFP^+ , the fragment of fixed-point logic that uses only (non-negated) greatest fixed-points. Since a greatest fixed-point formula $[\mathbf{gfp} R\bar{x}. \psi(R, \bar{x})](\bar{y})$ readily translates into $(\exists R)((\forall \bar{x}(R\bar{x} \rightarrow \psi(R, \bar{x})) \wedge R\bar{y}))$, GFP^+ can be viewed as a fragment of Σ_1^1 . Galliani and Hella [6] established that inclusion logic is equivalent to the set of sentences of form $\forall \bar{x}(X\bar{x} \rightarrow \psi(X, \bar{x}))$, where $\psi(X, \bar{x})$ is a formula in GFP^+ in which X occurs only positively. A different proof for this result, based on safety games and game interpretations, has been presented in [7].

Hence the question arises how these fragments change when the standard dependency notions are replaced by dependencies up to equivalence. There is a natural conjecture: *One has to restrict existential second-order quantification to relations that are closed under the given equivalence relation, i.e. to relations that can be written as unions of equivalence classes* (where equivalence is extended to tuples component-wise). We denote the resulting variant of existential second-order logic by $\Sigma_1^1(\approx)$.

Notice however, that to decide this conjecture is far from being trivial and, in fact, *the restriction of the standard translation to quantification over \approx -closed relations fails*. Even for simple disjunctions, the existential second-order expression given above describing the split of the team will not work anymore once we restrict quantification to \approx -closed relations because we cannot assume that the relevant subteams are \approx -closed. Here is a simple example, not even involving any dependencies: Consider the formula $x = y \vee x \neq y$ which is trivially true in any team X , by the split $X = Y \cup Z$ where Y contains the assignments s which $s(x) = s(y)$ and $Z = X \setminus Y$ (and this is the only split that works). However if there are elements $a \neq b$ with $a \approx b$ then in general neither Y nor Z are \approx -closed, even if X is.

Nevertheless we shall prove that the conjecture is true, and that we can characterize the expressive power of dependence logics up to equivalence by appropriate fragments of $\Sigma_1^1(\approx)$. This is based on a much more sophisticated translation from logics with team semantics into existential second-order logic that adapts ideas from [14]. We shall also present a fragment of GFP^+ that has the same expressive power as inclusion logic up to equivalence.

Our next question is then how the expressive power of $\Sigma_1^1(\approx)$, and hence logics of dependence up to equivalence, compare to first-order logic and to full Σ_1^1 . Of course this depends on the properties of the underlying equivalence relation, notably on the number and sizes of its equivalence classes.

- (1) On any class of structures on which \approx has only a bounded number of equivalence classes, $\Sigma_1^1(\approx)$, and hence all logics with dependencies up to equivalence as well, collapse to FO.
- (2) On any class of structures in which all equivalence classes have bounded size, and only a bounded number of classes have more than one element, $\Sigma_1^1(\approx) \equiv \Sigma_1^1$.
- (3) In general, and in particular on the classes of structures where all equivalence classes have size at most k (for $k > 1$), or that have only a bounded number of equivalence classes of size >1 , the expressive power of $\Sigma_1^1(\approx)$, and all the considered logics of dependence up to equivalence, are strictly between FO and Σ_1^1 .

To prove this we shall use appropriate variants of Ehrenfeucht-Fraïssé games for these logics.

2 The Logics $\text{FO}(\Omega_{\approx})$ and $\Sigma_1^1(\approx)$

Let τ be a signature containing a binary relation symbol \approx and let (τ, \approx) denote the class of τ -structures \mathfrak{A} in which \approx is interpreted by an equivalence relation on the universe A of \mathfrak{A} . For every $\mathfrak{A} \in (\tau, \approx)$ and every $\bar{a}, \bar{b} \in A^n$ we write $\bar{a} \approx \bar{b}$, if $a_i \approx b_i$ for every $i \in \{1, \dots, n\}$. Given two relations $R, S \subseteq A^k$ of the same arity we write $R \subseteq_{\approx} S$ if for every $\bar{a} \in R$, there exists some $\bar{b} \in S$ with $\bar{a} \approx \bar{b}$. We further write $R \approx S$ if $R \subseteq_{\approx} S$ and $S \subseteq_{\approx} R$. Furthermore, we define the \approx -closure of R as $R_{\approx} := \{\bar{a} : \bar{a} \approx \bar{b} \text{ for some } \bar{b} \in R\}$ and say that R is \approx -closed if, and only if, $R = R_{\approx}$.

A team over \mathfrak{A} is a set X of assignments $s : \text{dom}(X) \rightarrow A$ mapping a common finite domain of variables into the universe A of \mathfrak{A} . Given a tuple \bar{y} of variables from $\text{dom}(X)$, we denote by $X(\bar{y}) := \{s(\bar{y}) : s \in X\}$ the set of values that \bar{y} takes in X . The semantics of (in)dependence, inclusion and exclusion atoms up to \approx is given as follows:

► **Definition 1.** Let X be a team over \mathfrak{A} . Then we define

$$\begin{aligned} \mathfrak{A} \models_X \text{dep}_{\approx}(\bar{x}, y) & \quad \iff \text{ for all } s, s' \in X, \text{ if } s(\bar{x}) \approx s'(\bar{x}) \text{ then also } s(y) \approx s'(y), \\ \mathfrak{A} \models_X \bar{x} \perp_{\approx} \bar{y} & \quad \iff \text{ for all } s, s' \in X \text{ there exists some } s'' \in X \text{ such that} \\ & \quad \quad \quad s''(\bar{x}) \approx s(\bar{x}) \text{ and } s''(\bar{y}) \approx s'(\bar{y}), \\ \mathfrak{A} \models_X \bar{x} \subseteq_{\approx} \bar{y} & \quad \iff X(\bar{x}) := \{s(\bar{x}) : s \in X\} \subseteq_{\approx} X(\bar{y}), \\ \mathfrak{A} \models_X \bar{x} \mid_{\approx} \bar{y} & \quad \iff s(\bar{x}) \not\approx s'(\bar{y}) \text{ for all } s, s' \in X. \end{aligned}$$

For $\Omega_{\approx} \subseteq \{\text{dep}_{\approx}, \perp_{\approx}, \subseteq_{\approx}, \mid_{\approx}\}$ we denote by $\text{FO}(\Omega_{\approx})$ the set of all first-order formulas in negation normal form where we additionally allow positive occurrences of Ω_{\approx} -atoms. The semantics of first-order literals and of the logical operators are the usual ones in (lax) team semantics:

► **Definition 2.** Let $\varphi_1, \varphi_2, \psi \in \text{FO}(\Omega_{\approx})$, ϑ be some first-order literal and X a team over \mathfrak{A} .

$$\begin{aligned} \mathfrak{A} \models_X \vartheta & \quad \iff \mathfrak{A} \models_s \vartheta \text{ for every } s \in X, \\ \mathfrak{A} \models_X \varphi_1 \wedge \varphi_2 & \quad \iff \mathfrak{A} \models_X \varphi_1 \text{ and } \mathfrak{A} \models_X \varphi_2 \\ \mathfrak{A} \models_X \varphi_1 \vee \varphi_2 & \quad \iff X \text{ can be represented as } X = X_1 \cup X_2 \text{ such that} \\ & \quad \quad \quad \mathfrak{A} \models_{X_1} \varphi_1 \text{ and } \mathfrak{A} \models_{X_2} \varphi_2 \\ \mathfrak{A} \models_X \forall x \psi & \quad \iff \mathfrak{A} \models_{X[x \mapsto A]} \psi \\ \mathfrak{A} \models_X \exists x \psi & \quad \iff \mathfrak{A} \models_{X[x \mapsto F]} \psi \text{ for some } F : X \rightarrow \mathcal{P}(A) \setminus \{\emptyset\}. \end{aligned}$$

Here we have $X[x \mapsto A] := \{s[x \mapsto a] : s \in X, a \in A\}$ and $X[x \mapsto F] := \{s[x \mapsto a] : s \in X, a \in F(s)\}$. Sometimes we shall call a team Y an $\{x\}$ -extension of X , if $Y = X[x \mapsto F]$ for some function $F : X \rightarrow \mathcal{P}(A) \setminus \{\emptyset\}$.

Many standard results concerning the closure properties and relationships between different logics of dependence and independence (see e.g. [5]) carry over to this new setting with equivalences, by easy and straightforward adaptations of proofs (which are therefore omitted here). In particular, this includes the following observations:

- For all formulae in these logics the *locality principle* holds: $\mathfrak{A} \models_X \varphi$ if, and only if, $\mathfrak{A} \models_{X \upharpoonright \text{free}(\varphi)} \varphi$ (where $X \upharpoonright \text{free}(\varphi) := \{s \upharpoonright \text{free}(\varphi) : s \in X\}$ is the restriction of X to the free variables of φ).
- The logics $\text{FO}(\text{dep}_{\approx})$ and $\text{FO}(\mid_{\approx})$ are equivalent and downwards closed.
- The logic $\text{FO}(\subseteq_{\approx})$ is closed under unions of teams, and incomparable with $\text{FO}(\text{dep}_{\approx})$ and $\text{FO}(\mid_{\approx})$.
- Independence logic with equivalences, $\text{FO}(\perp_{\approx})$, is equivalent to inclusion-exclusion logic with equivalences, $\text{FO}(\subseteq_{\approx}, \mid_{\approx})$.

A much more difficult problem is to understand the expressive power of these logics in connection with existential second-order logic Σ_1^1 . As mentioned above, formulae of independence logic or, equivalently, inclusion-exclusion logic (without equivalences) have the same expressive power as existential second-order sentences, and weaker logics such as dependence logic, exclusion logic, or inclusion logic correspond to fragments of Σ_1^1 . To describe the expressive power of dependence logics with equivalences we introduce the \approx -closed fragment $\Sigma_1^1(\approx)$ of Σ_1^1 and show that it captures the expressiveness of $\text{FO}(\subseteq_{\approx}, |\approx)$.

► **Definition 3.** The logic $\Sigma_1^1(\approx)$ consists of sentences of the form

$$\psi := \exists_{\approx} R_1 \dots \exists_{\approx} R_k \varphi(R_1, \dots, R_k)$$

where $\varphi \in \text{FO}(\tau \cup \{R_1, \dots, R_k\})$. The semantics of ψ is given in terms of \approx -closed relations:

$$\mathfrak{A} \models \psi \iff \text{there are } \approx\text{-closed relations } R_1, \dots, R_k \text{ such that } (\mathfrak{A}, R_1, \dots, R_k) \models \varphi.$$

3 The Expressive Power of $\text{FO}(\subseteq_{\approx}, |\approx)$

In this section we establish that $\text{FO}(\subseteq_{\approx}, |\approx)$ has exactly the expressive power of $\Sigma_1^1(\approx)$. This means that every formula $\varphi(\bar{x}) \in \text{FO}(\subseteq_{\approx}, |\approx)$ can be translated into an equivalent sentence $\varphi' \in \Sigma_1^1(\approx)$ using an additional predicate for the team such that

$$\mathfrak{A} \models_X \varphi(\bar{x}) \iff (\mathfrak{A}, X) \models \varphi'(X).$$

Conversely, we are also going to show how a given sentence $\psi \in \Sigma_1^1(\approx)$ can be translated into an equivalent sentence $\psi^+ \in \text{FO}(\subseteq_{\approx}, |\approx)$.

3.1 From $\Sigma_1^1(\approx)$ to $\text{FO}(\subseteq_{\approx}, |\approx)$

To capture the semantics of a sentence $\exists_{\approx} R_1 \dots \exists_{\approx} R_k \varphi \in \Sigma_1^1(\approx)$ in $\text{FO}(\subseteq_{\approx}, |\approx)$ we adapt ideas by Rönnholm [14] and use tuples of variables $\bar{v}_1, \dots, \bar{v}_k$ of length $|\bar{v}_i| = \text{ar}(R_i)$ in order to simulate the (\approx -closed) relations R_1, \dots, R_k . The reason why this is possible lies in the fact that we are using team semantics: In a given team X with $\{\bar{v}_1, \dots, \bar{v}_k\} \subseteq \text{dom}(X)$ we naturally have that $X(\bar{v}_i)$ corresponds to a (not necessarily \approx -closed) relation. The most important step is to find a formula $\varphi^*(\bar{v}_1, \dots, \bar{v}_k) \in \text{FO}(\subseteq_{\approx}, |\approx)$ such that

$$(\mathfrak{A}, \bar{R}) \models \varphi \iff \mathfrak{A} \models_X \varphi^*(\bar{v}_1, \dots, \bar{v}_k)$$

where $X = \{s : s(\bar{v}_i) \in R_i \text{ for every } i \in \{1, \dots, k\}\}$. Towards this end, φ^* is constructed (inductively) while using inclusion/exclusion atoms to express (non)membership in R_1, \dots, R_k . For example, $\bar{x} \subseteq_{\approx} \bar{v}_i$ means that $s(\bar{x}) \in X(\bar{v}_i)_{\approx} = R_i$ for every $s \in X$, while $\bar{x} |\approx \bar{v}_i$ expresses that $s(\bar{x}) \notin X(\bar{v}_i)_{\approx} = R_i$ for every $s \in X$. Therefore, the semantics of $R_i \bar{x}$ resp. $\neg R_i \bar{x}$ is captured by $\bar{x} \subseteq_{\approx} \bar{v}_i$ resp. $\bar{x} |\approx \bar{v}_i$. But of course, it could be the case that φ is a much more complicated formula made up of quantifiers, conjunction or disjunctions. It turns out that quantifiers and conjunction can be handled with ease by simply setting

$$\begin{aligned} (Qu\vartheta)^* &:= Qu(\vartheta^*) \text{ for both quantifiers } Q \in \{\exists, \forall\}, \text{ and} \\ (\vartheta_1 \wedge \vartheta_2)^* &:= \vartheta_1^* \wedge \vartheta_2^*, \end{aligned}$$

because when evaluating conjunctions in team semantics, the team is not modified and in the process of evaluating quantifiers there are just more columns added to the team (w.l.o.g. we assume that every variable in the formula occurs either freely or is quantified

exactly once). However, for disjunctions the situation is much more delicate because it is *not* possible to define $(\vartheta_1 \vee \vartheta_2)^*$ as $\vartheta_1^* \vee \vartheta_2^*$. The reason for this is that after splitting the team X into X_1, X_2 with $X = X_1 \cup X_2$ and $\mathfrak{A} \models_{X_j} \vartheta_j^*$ it cannot be guaranteed that $X_j(\bar{v}_i)$ still describes the original R_i (up to equivalence). To make sure that we do not lose information about R_1, \dots, R_k , we use instead an adaptation of the *value preserving disjunction* that was introduced by Rönholm [14].

► **Lemma 4.** *Let $\psi_1, \psi_2 \in \text{FO}(\subseteq_{\approx}, |\approx)$ and $\bar{v}_1, \dots, \bar{v}_k$ be some tuples of variables. Then there exists a formula $\psi_1 \underset{\bar{v}_1, \dots, \bar{v}_k}{\vee} \psi_2 \in \text{FO}(\subseteq_{\approx}, |\approx)$ such that the following are equivalent:*

- (i) $\mathfrak{A} \models_X \psi_1 \underset{\bar{v}_1, \dots, \bar{v}_k}{\vee} \psi_2$
- (ii) $X = X_1 \cup X_2$ for some teams X_1, X_2 such that for both $j = 1$ and $j = 2$:
 - $\mathfrak{A} \models_{X_j} \psi_j$, and
 - if $X_j \neq \emptyset$, then $X_j(\bar{v}_i) \approx X(\bar{v}_i)$ for all $i \in \{1, \dots, k\}$.

Proof. The construction of $\psi_1 \underset{\bar{v}_1, \dots, \bar{v}_k}{\vee} \psi_2$ relies on the *intuitionistic disjunction* $\psi_1 \sqcup \psi_2$ with

$$\mathfrak{A} \models_X \psi_1 \sqcup \psi_2 \iff \mathfrak{A} \models_X \psi_1 \text{ or } \mathfrak{A} \models_X \psi_2.$$

On structures $\mathfrak{A} \in (\tau, \approx)$ with $\approx^{\mathfrak{A}} \neq A^2$ this is definable in $\text{FO}(\subseteq_{\approx}, |\approx)$ since

$$\psi_1 \sqcup \psi_2 \equiv \exists c_\ell \exists c_r (\text{dep}_{\approx}(c_\ell) \wedge \text{dep}_{\approx}(c_r) \wedge [(c_\ell \approx c_r \wedge \psi_1) \vee (\neg c_\ell \approx c_r \wedge \psi_2)])$$

where c_ℓ and c_r are some variables not occurring in ψ_1 or ψ_2 . Note that $\text{dep}_{\approx}(c)$ expresses that c only assumes values from a single equivalence class. The proof of this equivalence is a simple exercise. Now consider the following formula, which is a modification of a construction by Rönholm [14].

$$\begin{aligned} \psi_1 \underset{\bar{v}_1, \dots, \bar{v}_k}{\vee'} \psi_2 := & (\psi_1 \sqcup \psi_2) \sqcup \exists c_\ell \exists c_r \left(\text{dep}_{\approx}(c_\ell) \wedge \text{dep}_{\approx}(c_r) \wedge c_\ell \not\approx c_r \wedge \right. \\ & \left. \exists y \left([(y \approx c_\ell \wedge \psi_1) \vee (y \approx c_r \wedge \psi_2)] \wedge \right. \right. \\ & \left. \left. \bigwedge_{i=1}^k \Theta_i \wedge \Theta'_i \right) \right). \end{aligned}$$

Θ_i and Θ'_i are given by

$$\begin{aligned} \Theta_i := & \exists \bar{z}_1 \exists \bar{z}_2 \left([(y \approx c_\ell \wedge \bar{z}_1 = \bar{v}_i \wedge \bar{z}_2 = \bar{c}_\ell) \vee (y \approx c_r \wedge \bar{z}_1 = \bar{c}_r \wedge \bar{z}_2 = \bar{v}_i)] \right. \\ & \left. \wedge \bar{v}_i \subseteq_{\approx} \bar{z}_1 \wedge \bar{v}_i \subseteq_{\approx} \bar{z}_2 \right) \\ \Theta'_i := & \exists \bar{z}_1 \exists \bar{z}_2 \left([(y \approx c_\ell \wedge \bar{z}_1 = \bar{v}_i \wedge \bar{z}_2 = \bar{c}_r) \vee (y \approx c_r \wedge \bar{z}_1 = \bar{c}_r \wedge \bar{z}_2 = \bar{v}_i)] \right. \\ & \left. \wedge \bar{v}_i \subseteq_{\approx} \bar{z}_1 \wedge \bar{v}_i \subseteq_{\approx} \bar{z}_2 \right). \end{aligned}$$

where $\bar{c}_\ell = (c_\ell, c_\ell, \dots, c_\ell)$ and $\bar{c}_r = (c_r, c_r, \dots, c_r)$ are always tuples of the correct length.

It is not difficult to prove that this formula is almost what we want: it satisfies the properties required by Lemma 4 under the additional condition that \approx has at least two different equivalence classes. To get rid of this condition, we put:

$$\psi_1 \underset{\bar{v}_1, \dots, \bar{v}_k}{\vee} \psi_2 := [\forall x \forall y (x \approx y) \wedge (\psi_1 \vee \psi_2)] \vee \left[\exists x \exists y (x \not\approx y) \wedge \left(\psi_1 \underset{\bar{v}_1, \dots, \bar{v}_k}{\vee'} \psi_2 \right) \right]. \quad \blacktriangleleft$$

We can now complete the inductive definition of φ^* by:

$$(\vartheta_1 \vee \vartheta_2)^* := \vartheta_1^* \underset{\bar{v}_1, \dots, \bar{v}_k}{\vee} \vartheta_2^*$$

By rather straightforward inductions one can establish the following two lemmata.

► **Lemma 5.** For every $\mathfrak{A} \in (\tau, \approx)$ and every team X with $\text{dom}(X) = \{\bar{v}_1, \dots, \bar{v}_k\}$,

$$\mathfrak{A} \models_X \varphi^* \implies (\mathfrak{A}, R_1^X, \dots, R_k^X) \models_X \varphi$$

where $R_i^X := (X(\bar{v}_i))_{\approx}$ for $i = 1, \dots, k$, i.e. R_i^X is defined as the \approx -closure of $X(\bar{v}_i)$.

► **Lemma 6.** Let $\bar{R} = (R_1, \dots, R_k)$ be a tuple of non-empty \approx -closed relations with $(\mathfrak{A}, \bar{R}) \models \varphi$. Then $\mathfrak{A} \models_Y \varphi^*$ where $Y := \{(\bar{v}_1, \dots, \bar{v}_k) \mapsto (\bar{a}_1, \dots, \bar{a}_k) : \bar{a}_1 \in R_1, \dots, \bar{a}_k \in R_k\}$.

The non-emptiness requirement of R_1, \dots, R_k does not create a serious problem, because by rewriting the formula φ it can be assumed w.l.o.g. that $\exists_{\approx} R_1 \dots \exists_{\approx} R_k \varphi$ is satisfied in a structure \mathfrak{A} if, and only if, there are *non-empty* \approx -closed relations R_1, \dots, R_k such that $(\mathfrak{A}, \bar{R}) \models \varphi$.

► **Theorem 7.** $\psi := \exists \bar{v}_1 \dots \exists \bar{v}_k \varphi^*$ is equivalent to $\exists_{\approx} R_1 \dots \exists_{\approx} R_k \varphi$.

Proof. Assume that $\mathfrak{A} \models \psi$. It follows that there exists some team X with $\text{dom}(X) = \{\bar{v}_1, \dots, \bar{v}_k\}$ and $\mathfrak{A} \models_X \varphi^*$. By Lemma 5 and $\text{free}(\varphi) = \emptyset$, we obtain that $(\mathfrak{A}, R_1^X, \dots, R_k^X) \models \varphi$. By definition, the relations R_i^X are \approx -closed and, hence, $\mathfrak{A} \models \exists_{\approx} R_1 \dots \exists_{\approx} R_k \varphi$.

For the converse direction, let $\mathfrak{A} \models \exists_{\approx} R_1 \dots \exists_{\approx} R_k \varphi$. So there exists some *non-empty* \approx -closed relations R_1, \dots, R_k such that $(\mathfrak{A}, R_1, \dots, R_k) \models \varphi$. So by Lemma 6, it follows that $\mathfrak{A} \models_Y \varphi^*$ where Y is the team given in Lemma 6. This leads to $\mathfrak{A} \models \exists \bar{v}_1 \dots \exists \bar{v}_k \varphi^*$. ◀

3.2 From $\text{FO}(\subseteq_{\approx}, |\approx)$ to $\Sigma_1^1(\approx)$

Up to this point we only know that $\Sigma_1^1(\approx) \leq \text{FO}(\subseteq_{\approx}, |\approx)$. In this section we prove that these two logics have in fact the same expressive power. Towards this end, we demonstrate how a given formula $\varphi \in \text{FO}(\subseteq_{\approx}, |\approx)$ can be translated into $\Sigma_1^1(\approx)$. There are two obstacles that we need to overcome:

1. When viewed as relations, teams usually are not \approx -closed, so we cannot use the quantifier \exists_{\approx} to fetch the subteams we would need to satisfy the subformulae of e.g. a disjunction.
2. Unlike in Σ_1^1 , where a formula of the form $\forall x \exists Y(\dots)$ is equivalent to formula like $\exists Y' \forall x(\dots)$ where $\text{ar}(Y') = \text{ar}(Y) + 1$, there seems to be no obvious way to perform a similar syntactic manipulation in $\Sigma_1^1(\approx)$. Thus we have to be content with the limited quantification that $\Sigma_1^1(\approx)$ allows us.

The main idea of the construction, which is inspired by [14], is to replace every inclusion and exclusion atom ϑ by a separate new relation symbol R_{ϑ} that contains certain values enabling us to express the semantics of φ in $\Sigma_1^1(\approx)$.

First we describe how this approach deals with exclusion atoms. Let $\vartheta_1, \dots, \vartheta_k$ be an enumeration of all occurrences of exclusion atoms $\vartheta_i = \bar{u}_i \mid_{\approx} \bar{w}_i$ in φ . We assume w.l.o.g. that the tuples $\bar{u}_1, \dots, \bar{u}_k, \bar{w}_1, \dots, \bar{w}_k$ are pairwise different. We use new relation symbols $R_{\vartheta_1}, \dots, R_{\vartheta_k}$ that are intended to separate the sets of possible values for \bar{v}_i and \bar{w}_i (up to equivalence). The desired translation φ^* of φ is now obtained by replacing the exclusion atoms $\vartheta_i = \bar{u}_i \mid_{\approx} \bar{w}_i$ by $R_{\vartheta_i} \bar{u}_i \wedge \neg R_{\vartheta_i} \bar{w}_i$. This construction leads to the following result. The proof is by induction over φ and is given in the appendix.

► **Theorem 8.** For every formula $\varphi(\bar{x}) \in \text{FO}(|_{\approx}, \subseteq_{\approx})$ with signature τ there exists a formula $\varphi^*(\bar{x}) \in \text{FO}(\subseteq_{\approx})$ with signature $\tau \cup \{\bar{R}\}$, where \bar{R} is a tuple of new relation symbols, such that for every $\mathfrak{A} \in (\tau, \approx)$ and every team X the following are equivalent:

- (i) $\mathfrak{A} \models_X \varphi$
- (ii) There are \approx -closed relations \bar{R} over \mathfrak{A} such that $(\mathfrak{A}, \bar{R}) \models_X \varphi^*$.

After this elimination of the exclusion atoms we still need to cope with \subseteq_{\approx} -atoms. Towards this end, let $\varphi \in \text{FO}(\subseteq_{\approx})$ and $\vartheta_1, \dots, \vartheta_k$ be an enumeration of all occurrences of inclusion atoms in φ , with $\vartheta_i := \bar{x}_i \subseteq_{\approx} \bar{y}_i$ for every $i \in \{1, \dots, k\}$. We shall use new relation symbols $R_{\vartheta_1}, \dots, R_{\vartheta_k}$ with the intended semantics that $R_{\vartheta_i} \subseteq X(\bar{y}_i)_{\approx}$ where X is the team that “arrives” at ϑ_i . This will allow us to replace the subformulae ϑ_i by the formula $R_{\vartheta_i} \bar{x}_i$. However, this formula alone does not verify that $R_{\vartheta_i} \subseteq X(\bar{y}_i)_{\approx}$ really holds. Additional formulae $\varphi^{(1)}(\bar{z}_1), \dots, \varphi^{(k)}(\bar{z}_k)$ are required for the verification that values from R_{ϑ_i} could occur (up to equivalence) as a value for \bar{y}_i in the team X that arrives at the corresponding inclusion atom. More precisely, $\varphi^{(i)}$ is constructed such that

$$(\mathfrak{A}, R_{\vartheta_1}, \dots, R_{\vartheta_k}) \models_{s[\bar{z}_i \mapsto \bar{a}]} \varphi^{(i)}(\bar{z}_i)$$

implies that the assignment s also satisfies φ and, more importantly, leads to an assignment s' that satisfies $s'(\bar{z}_i) \approx \bar{a}$ and that could be part of the team that satisfies the inclusion atom. Formally, we are going to prove that

$$\begin{aligned} \mathfrak{A} \models_X \varphi &\iff \text{there are } \approx\text{-closed relations } R_{\vartheta_1}, \dots, R_{\vartheta_k} \text{ such that } (\mathfrak{A}, \bar{R}) \models_X \varphi^* \text{ and} \\ &\text{for every } \bar{a} \in R_{\vartheta_i} \text{ there is an } s \in X \text{ with } (\mathfrak{A}, \bar{R}) \models_{s[\bar{z}_i \mapsto \bar{a}]} \varphi^{(i)}(\bar{z}_i). \end{aligned}$$

As already pointed out, φ^* results from φ by replacing every inclusion atom $\vartheta_i = \bar{x}_i \subseteq_{\approx} \bar{y}_i$ by $R_{\vartheta_i} \bar{x}_i$, while $\varphi^{(i)}$ is defined by induction (for every $i \in \{1, \dots, k\}$). Let ϑ be a subformula of φ . First-order literals are unchanged, i.e. $\vartheta^* := \vartheta :=: \vartheta^{(i)}$ if ϑ is such a literal. The inclusion atoms are translated as follows:

$$(\bar{x}_j \subseteq \bar{y}_j)^{(i)} := \begin{cases} R_{\vartheta_i} \bar{x}_i \wedge \bar{y}_i \approx \bar{z}_i, & \text{if } i = j \\ R_{\vartheta_j} \bar{x}_j, & \text{if } i \neq j \end{cases}$$

Conjunctions and existential quantifiers are handled by defining

$$\begin{aligned} (\exists x \tilde{\vartheta})^{(i)} &:= \exists x \tilde{\vartheta}^{(i)} \text{ and} \\ (\tilde{\vartheta}_1 \wedge \tilde{\vartheta}_2)^{(i)} &:= \tilde{\vartheta}_1^{(i)} \wedge \tilde{\vartheta}_2^{(i)}. \end{aligned}$$

However, the translation of universal quantifiers or disjunctions is more complex:

$$\begin{aligned} (\tilde{\vartheta}_1 \vee \tilde{\vartheta}_2)^{(i)} &:= \begin{cases} \tilde{\vartheta}_j^{(i)}, & \text{if } \bar{x}_i \subseteq_{\approx} \bar{y}_i \text{ occurs in } \tilde{\vartheta}_j \\ (\tilde{\vartheta}_1 \vee \tilde{\vartheta}_2)^*, & \text{otherwise} \end{cases} \\ (\forall x \tilde{\vartheta})^{(i)} &:= \exists x \tilde{\vartheta}^{(i)} \wedge (\forall x \tilde{\vartheta})^*. \end{aligned}$$

By construction we have that $(\forall x \vartheta)^*$ is implied by $(\forall x \vartheta)^{(i)}$, because it is a subformula, while $\exists x \vartheta^{(i)}$ fetches the correct extension of the current assignment such that we end up with an assignment satisfying $\bar{y}_i \approx \bar{z}_i$ when arriving at the translation of $\bar{x}_i \subseteq_{\approx} \bar{y}_i$. The next lemma states that this construction actually captures the intuition that we have described above. The proof is given in the appendix.

► **Lemma 9.** *Let $\mathfrak{A} \in (\tau, \approx)$ and X be a team over \mathfrak{A} with $\text{free}(\varphi) = \text{dom}(X)$. Then the following are equivalent:*

- (i) $\mathfrak{A} \models_X \varphi$
- (ii) *There are \approx -closed relations $\bar{R} = (R_{\vartheta_1}, \dots, R_{\vartheta_k})$ over \mathfrak{A} such that $(\mathfrak{A}, \bar{R}) \models_X \varphi^*$ and for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_{\vartheta_i}$ there exists some $s \in X$ such that $(\mathfrak{A}, \bar{R}) \models_{s[\bar{z}_i \mapsto \bar{a}]} \varphi^{(i)}$.*

We are now ready to show how inclusion atoms are translated into $\Sigma_1^1(\approx)$.

25:10 Dependency Concepts up to Equivalence

► **Theorem 10.** *For every formula $\varphi(\bar{x}) \in \text{FO}(\subseteq_{\approx})$ there exists a sentence $\varphi'(X) \in \Sigma_1^1(\approx)$ such that $\mathfrak{A} \models_X \varphi(\bar{x}) \iff (\mathfrak{A}, X) \models \varphi'(X)$ for every structure \mathfrak{A} and every team X .*

Proof. Let

$$\varphi' := \exists_{\approx} R_{\vartheta_1} \dots \exists_{\approx} R_{\vartheta_k} \left(\forall \bar{x} (X\bar{x} \rightarrow \varphi^*(\bar{x})) \wedge \bigwedge_{i=1}^k \forall \bar{z}_i (R_{\vartheta_i} \bar{z}_i \rightarrow \exists \bar{x} (X\bar{x} \wedge \varphi^{(i)}(\bar{x}, \bar{z}_i))) \right).$$

By construction, $(\mathfrak{A}, X) \models \varphi'$ if, and only if, there exist \approx -closed relations \bar{R} over \mathfrak{A} such that $(\mathfrak{A}, \bar{R}) \models_s \varphi^*$ for every $s \in X$, and for every $\bar{a} \in R_i$ there exists some $s \in X$ with $(\mathfrak{A}, \bar{R}) \models_{s[\bar{z}_i \mapsto \bar{a}]} \varphi^{(i)}$. Since φ^* is a first-order formula, $\mathfrak{A} \models_s \varphi^*$ for every $s \in X$ if, and only if, $(\mathfrak{A}, \bar{R}) \models_X \varphi^*$. Hence, by Lemma 9, we can conclude that $(\mathfrak{A}, X) \models \varphi' \iff \mathfrak{A} \models_X \varphi$. ◀

In particular, every sentence $\varphi \in \text{FO}(\subseteq_{\approx})$ can be translated into an equivalent sentence $\varphi' \in \Sigma_1^1(\approx)$.

4 FO(\subseteq_{\approx}) vs. GFP

An important result on logics with team semantics is the tight connection between inclusion logic and GFP^+ , established by Galliani and Hella [6]. In this section we prove a similar result for $\text{FO}(\subseteq_{\approx})$ by defining a fragment of GFP^+ which has the same expressive power as $\text{FO}(\subseteq_{\approx})$.

► **Definition 11 (para-GFP $_{\approx}^+$).** The logic para-GFP $_{\approx}^+$ is defined as an extension of FO in negation normal form by the following formula formation rule. Let $k \geq 1$ and $\bar{R} = (R_1, \dots, R_k)$ be a tuple of unused relation symbols of arity n_1, \dots, n_k respectively and let $(\varphi_i(\bar{R}, \bar{x}_i))_{i=1, \dots, k}$ be a tuple of $\text{FO}(\tau \cup \{R_1, \dots, R_k\})$ -formulae in negation normal form where $|\bar{x}_i| = n_i$ and every R_i occurs only positively in $\varphi_1, \dots, \varphi_k$. Furthermore, let $j \in \{1, \dots, k\}$ and \bar{v} be a n_j -tuple of variables. Then

$$\varphi(\bar{v}) := [\text{para-GFP}_{\approx} (R_i, \bar{x}_i)_{i=1, \dots, k} \cdot (\varphi_i(\bar{R}, \bar{x}_i))_{i=1, \dots, k}]_j(\bar{v})$$

is a para-GFP $_{\approx}^+$ -formula.

On every structure $\mathfrak{A} \in (\tau, \approx)$, the system $(\varphi_i(\bar{R}, \bar{x}_i))_{i=1, \dots, k}$ defines a parallel update operator $\Gamma^{\mathfrak{A}} : \mathcal{P}(A^{n_1}) \times \dots \times \mathcal{P}(A^{n_k}) \rightarrow \mathcal{P}(A^{n_1}) \times \dots \times \mathcal{P}(A^{n_k})$, by

$$\begin{aligned} \Gamma(\bar{R}) &:= (\Gamma_1(\bar{R}), \dots, \Gamma_k(\bar{R})) \text{ where} \\ \Gamma_i(\bar{R}) &:= \llbracket \varphi_i(\bar{R}) \rrbracket_{\approx}^{\mathfrak{A}} = \{\bar{a} \in A^{n_i} : (\mathfrak{A}, \bar{R}) \models \varphi_i(\bar{R}, \bar{a})\}_{\approx} \end{aligned}$$

A tuple (\mathfrak{A}, s) where $\mathfrak{A} \in (\tau, \approx)$ and $s : \{\bar{v}\} \rightarrow A$ is called a model of φ (and we write $\mathfrak{A} \models_s \varphi$ in this case) if, and only if, for the greatest fixed-point $\bar{S} = (S_1, \dots, S_k)$ of $\Gamma^{\mathfrak{A}}$ we have that $s(\bar{v}) \in S_j$.

The non-parallel variant GFP_{\approx}^+ , where it is only allowed to use the operator $\text{para-GFP}_{\approx}$ in a non-parallel way, i.e. only in the following shape

$$[\text{GFP}_{\approx} R\bar{x} \cdot \varphi(R, \bar{x})](\bar{y}) := [\text{para-GFP}_{\approx}^+ R\bar{x} \cdot \varphi(R, \bar{x})]_1(\bar{y}),$$

has exactly the same expressive power as para-GFP $_{\approx}^+$.

The following lemma gives a characterization of the fixed-points of Γ :

► **Lemma 12** (Knaster-Tarski-Theorem for para-GFP $_{\approx}^+$). *Let*

$$\varphi(\bar{v}) = [\text{para-GFP}_{\approx} (R_i, \bar{x}_i)_{i=1, \dots, k} \cdot (\varphi_i(\bar{R}, \bar{x}_i))_{i=1, \dots, k}]_j(\bar{v})$$

be a para-GFP $_{\approx}^+$ -formula, $\mathfrak{A} \in (\tau, \approx)$ and $\Gamma (= \Gamma^{\mathfrak{A}})$ be the corresponding parallel update operator w.r.t. $\varphi_1, \dots, \varphi_k$. For two given k -tuples \bar{R}, \bar{S} of relations, we write $\bar{R} \subseteq \bar{S}$ if, and only if $R_i \subseteq S_i$ for every $i \in \{1, \dots, k\}$.

Let $X := \{\bar{S} : \bar{S} \subseteq \Gamma(\bar{S})\}$. Then $\bigcup X := (Y_1, \dots, Y_k)$ where for every $j \in \{1, \dots, k\}$, $Y_j := \bigcup_{\bar{S} \in X} S_j$ is the greatest fixed-point of Γ . Furthermore, these Y_j are \approx -closed.

4.1 From FO(\subseteq_{\approx}) to GFP $_{\approx}^+$

► **Theorem 13.** *For every formula $\varphi(\bar{x}) \in \text{FO}(\subseteq_{\approx})$ there exists a sentence $\varphi^+ \in \text{GFP}_{\approx}^+$ such that $\mathfrak{A} \models_X \varphi \iff (\mathfrak{A}, X) \models \varphi^+$ for every structure $\mathfrak{A} \in (\tau, \approx)$ and every team X over \mathfrak{A} .*

Proof. In the last section we have presented the FO-formulae $\varphi^*(\bar{R})$ and $\varphi^{(i)}(\bar{R})$ (for $i \in \{1, \dots, k\}$) using new relation symbols $\bar{R} = (R_1, \dots, R_k)$ such that for every $\mathfrak{A} \in (\tau, \approx)$ and every team X over \mathfrak{A} with $\text{dom}(X) \supseteq \text{free}(\varphi)$ the following are equivalent:

- (1) $\mathfrak{A} \models_X \varphi$
- (2) There are \approx -closed relations \bar{R} over \mathfrak{A} such that $(\mathfrak{A}, \bar{R}) \models_X \varphi^*$ and for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_i$ there exists some $s_{i, \bar{a}} \in X$ such that $(\mathfrak{A}, \bar{R}) \models_{s_{i, \bar{a}}[\bar{z}_i \mapsto \bar{a}]} \varphi^{(i)}$.

Furthermore, the relation symbols R_1, \dots, R_k occur only positively in φ^* and $\varphi^{(i)}$ and the tuple \bar{z}_i occurs exactly once in a subformula of the form $\bar{x}_i \approx \bar{z}_i$ in $\varphi^{(i)}$. Let $\tilde{\varphi}^*$ and the $\tilde{\varphi}^{(i)}$ be the formulae that result from $\varphi^*, \varphi^{(i)}$ by replacing every occurrence of the form $R_i \bar{v}$ by its guarded version $(R_i)_{\approx} \bar{v} := \exists \bar{w}(\bar{v} \approx \bar{w} \wedge R_i \bar{w})$. This allows us to drop the requirement that the relations \bar{R} are \approx -closed.

► **Claim 14.** For every \mathfrak{A} and every team X over \mathfrak{A} , (1) and (2) are equivalent to:

- (3) There are relations \bar{R} over \mathfrak{A} such that $(\mathfrak{A}, \bar{R}) \models_X \tilde{\varphi}^*$ and for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_i$ there exists some $s_{i, \bar{a}} \in X$ such that $(\mathfrak{A}, \bar{R}) \models_{s_{i, \bar{a}}[\bar{z}_i \mapsto \bar{a}]} \tilde{\varphi}^{(i)}$.

To prove this, one has to exploit the fact that every R_j ($j \in \{1, \dots, k\}$) occurs only \approx -guarded in $\tilde{\varphi}^*, \tilde{\varphi}^{(1)}, \dots, \tilde{\varphi}^{(k)}$ and the variables \bar{z}_i occur (exactly once) in a subformula of the form $\bar{w} \approx \bar{z}_i$ in $\varphi^{(i)}$. By expressing (3) in existential second-order logic, we obtain the following equivalent statement:

- (4) $(\mathfrak{A}, X) \models \exists \bar{R} (\forall \bar{x} (X\bar{x} \rightarrow \tilde{\varphi}^*(\bar{R}, \bar{x})) \wedge \psi)$ where $\psi := \bigwedge_{i=1}^k \forall \bar{z}_i (R_i \bar{z}_i \rightarrow \eta_i(\bar{R}, \bar{z}_i))$ and $\eta_i(\bar{R}, \bar{z}_i) := \exists \bar{x} (X\bar{x} \wedge \tilde{\varphi}^{(i)}(\bar{R}, \bar{z}_i, \bar{x}))$.

Let $\Gamma(\bar{R}) := (\Gamma_1(\bar{R}), \dots, \Gamma_k(\bar{R}))$ where

$$\Gamma_i(\bar{R}) := \llbracket \eta_i(\bar{R}, \bar{z}_i) \rrbracket^{(\mathfrak{A}, X)} = \{\bar{a} \in A^{\text{ar}(R_i)} : (\mathfrak{A}, X(\bar{x}), \bar{R}) \models \eta_i(\bar{a})\}.$$

Note that $\llbracket \eta_i(\bar{R}, \bar{z}_i) \rrbracket^{(\mathfrak{A}, X(\bar{x}))} = \llbracket \eta_i(\bar{R}, \bar{z}_i) \rrbracket_{\approx}^{(\mathfrak{A}, X(\bar{x}))}$, because the free variables \bar{z}_i occur exactly once in a subformula of the form $\bar{w} \approx \bar{z}_i$. This is the reason why Γ is the para-GFP $_{\approx}^+$ -update operator w.r.t. η_1, \dots, η_k .

Furthermore, $(\mathfrak{A}, X, \bar{R}) \models \forall \bar{z}_i (R_i \bar{z}_i \rightarrow \eta_i(\bar{R}, \bar{z}_i))$ if, and only if, $R_i \subseteq \Gamma_i(\bar{R})$. Consequently, we have $(\mathfrak{A}, X, \bar{R}) \models \psi$ if, and only if, $\bar{R} \subseteq \Gamma(\bar{R})$.

► **Claim 15.** For $j \leq k$, let $\vartheta_j(\bar{z}_j) := [\text{para-GFP}_{\approx} (R_i, \bar{z}_i)_{i=1, \dots, k} \cdot (\eta_i(\bar{R}, \bar{z}_i))_{i=1, \dots, k}]_j(\bar{z}_j)$, and let γ result from $\tilde{\varphi}^*$ by replacing every occurrence of $R_j(\bar{w})$ by $\vartheta_j(\bar{w})$. Then, for every $\mathfrak{A} \in (\tau, \approx)$ and every team X , (4) is equivalent to

- (5) $(\mathfrak{A}, X) \models \forall \bar{x} (X\bar{x} \rightarrow \gamma)$.

Proof. (4) \implies (5) : Let $(\mathfrak{A}, X) \models \exists \bar{R} (\forall \bar{x} (X\bar{x} \rightarrow \tilde{\varphi}^*(\bar{R}, \bar{x})) \wedge \psi)$. Then there are relations \bar{R} such that $(\mathfrak{A}, X, \bar{R}) \models \forall \bar{x} (X\bar{x} \rightarrow \tilde{\varphi}^*(\bar{R}, \bar{x}))$ and $(\mathfrak{A}, X, \bar{R}) \models \psi$. As observed above, it follows that $\bar{R} \subseteq \Gamma(\bar{R})$. So, by Lemma 12, $\bar{R} \subseteq \bar{S}$ where \bar{S} is the greatest fixed-point of Γ . Since we have $(\mathfrak{A}, X, \bar{R}) \models \forall \bar{x} (X\bar{x} \rightarrow \tilde{\varphi}^*(\bar{R}, \bar{x}))$ and the relations symbols R_1, \dots, R_k occur only positively in $\tilde{\varphi}^*$, we can conclude that $(\mathfrak{A}, X, \bar{S}) \models \forall \bar{x} (X\bar{x} \rightarrow \tilde{\varphi}^*(\bar{S}, \bar{x}))$. Because \bar{S} is the greatest fixed-point of Γ , it follows that $S_i = \llbracket \vartheta_i(\bar{z}_i) \rrbracket^{(\mathfrak{A}, X)}$ and, by construction of γ , we obtain that $(\mathfrak{A}, X) \models \forall \bar{x} (X\bar{x} \rightarrow \gamma)$.

(5) \implies (4) : Let $(\mathfrak{A}, X) \models \forall \bar{x} (X\bar{x} \rightarrow \gamma)$ and let \bar{S} be the greatest fixed-point of Γ . Then $(\mathfrak{A}, X, \bar{S}) \models \forall \bar{x} (X\bar{x} \rightarrow \tilde{\varphi}^*)$ and $\bar{S} = \Gamma(\bar{S})$. Therefore, we have $(\mathfrak{A}, X, \bar{S}) \models \forall \bar{z}_i (S_i \bar{z}_i \rightarrow \eta_i(\bar{z}_i))$ for every $i \in \{1, \dots, k\}$ and, hence, $(\mathfrak{A}, X, \bar{S}) \models \psi(\bar{S})$. \blacktriangleleft

By construction $\forall \bar{x} (X\bar{x} \rightarrow \gamma) \in \text{para-GFP}_{\approx}^+$. Since $\text{para-GFP}_{\approx}^+$ has the same expressive power as GFP_{\approx}^+ , there also exists a sentence $\varphi^+ \in \text{GFP}_{\approx}^+$ that is equivalent to $\varphi(\bar{x})$. \blacktriangleleft

4.2 From GFP_{\approx}^+ to $\text{FO}(\subseteq_{\approx})$

In order to translate a given sentence $\varphi \in \text{GFP}_{\approx}^+$ into a $\text{FO}(\subseteq_{\approx})$ -formula, we assume that φ is in a normal form which is given by the following lemma. By using adaptations of ideas from [6] we then show that such a sentence can be expressed in $\text{FO}(\subseteq_{\approx})$.

► **Lemma 16.** *For every sentence $\varphi \in \text{para-GFP}_{\approx}^+$ there exists a formula $\psi(R, \bar{x}) \in \text{FO}$, in which R occurs only positively and only \approx -guarded, such that φ is equivalent to*

$$\exists \bar{v} [\text{GFP}_{\approx} R\bar{x} . \psi(R, \bar{x})](\bar{v}).$$

Our next lemma shows that we can eliminate the relation symbol R in ψ by introducing \subseteq_{\approx} -atoms and encoding R in a tuple \bar{x} of variables.

► **Lemma 17.** *Let R be a relation symbol of arity n , let \bar{x}, \bar{y} be tuples of variables where $|\bar{x}| = n$ (whereas \bar{y} is of arbitrary length and can also be empty). Furthermore, let $\psi(R, \bar{x}, \bar{y}) \in \text{FO}(\tau \cup \{R\})$ be a first-order formula in which R occurs only positively and \approx -guarded, and with $\text{free}(\psi) \subseteq \{\bar{x}, \bar{y}\}$ such that the variables in \bar{x} are never quantified in ψ . Then there exists a formula $\psi^*(\bar{x}, \bar{y}) \in \text{FO}(\subseteq_{\approx})$ of signature τ such that for every $\mathfrak{A} \in (\tau, \approx)$ and every team X we have that*

$$\mathfrak{A} \models_X \psi^*(\bar{x}, \bar{y}) \iff (\mathfrak{A}, X(\bar{x})) \models_s \psi(R, \bar{x}, \bar{y}) \text{ for every } s \in X.$$

This lemma can be shown by induction over the structure of ψ . Now we are able to express $[\text{GFP}_{\approx} R\bar{x} . \varphi(R, \bar{x})]$ in $\text{FO}(\subseteq_{\approx})$.

► **Theorem 18.** *Let $\psi(R, \bar{x}) \in \text{FO}$ where $\text{ar}(R) = |\bar{x}|$, R occurs only positively and \approx -guarded in ψ , and the variables in \bar{x} are never quantified in ψ . Then there exists a formula $\psi^+(\bar{x}) \in \text{FO}(\subseteq_{\approx})$ such that for every $\mathfrak{A} \in (\tau, \approx)$ and every team X we have that*

$$\mathfrak{A} \models_X \psi^+(\bar{x}) \iff \mathfrak{A} \models_s [\text{GFP}_{\approx} R\bar{x} . \psi(R, \bar{x})](\bar{x}) \text{ for every } s \in X.$$

Proof. Let $\psi^+(\bar{x}) := \exists \bar{y} (\bar{x} \subseteq_{\approx} \bar{y} \wedge \exists \bar{z} (\bar{y} \approx \bar{z} \wedge \psi^*(\bar{z})))$.

“ \implies ”: First we assume that $\mathfrak{A} \models_X \psi^+(\bar{x})$. Then there exists a function $F : X \rightarrow \mathcal{P}(A^n) \setminus \{\emptyset\}$ such that $\mathfrak{A} \models_Y \bar{x} \subseteq_{\approx} \bar{y} \wedge \exists \bar{z} (\bar{y} \approx \bar{z} \wedge \psi^*(\bar{z}))$ where $Y := X[\bar{y} \mapsto F]$. So there exists a function $G : Y \rightarrow \mathcal{P}(A^n) \setminus \{\emptyset\}$ satisfying $\mathfrak{A} \models_Z \bar{y} \approx \bar{z} \wedge \psi^*(\bar{z})$ where $Z := Y[\bar{z} \mapsto G]$. By Lemma 17, it follows that

$$(\mathfrak{A}, Z(\bar{z})) \models_s \psi(R, \bar{z}) \text{ for every } s \in Z.$$

So we have $Z(\bar{z}) \subseteq \llbracket \psi(R, \bar{z}) \rrbracket^{(\mathfrak{A}, Z(\bar{z}))} \subseteq \llbracket \psi(R, \bar{z}) \rrbracket_{\approx}^{(\mathfrak{A}, Z(\bar{z}))} = \Gamma_\psi(Z(\bar{z}))$ where $\Gamma_\psi := \Gamma_\psi^{\mathfrak{A}}$ is the GFP_{\approx}^+ -update operator w.r.t. ψ . It follows that $Z(\bar{z}) \subseteq \text{gfp}(\Gamma_\psi)$ (by Lemma 12). Since $\text{gfp}(\Gamma_\psi)$ is \approx -closed and $X(\bar{x}) \subseteq_{\approx} Y(\bar{y}) \approx Z(\bar{z})$, we have that $X(\bar{x}) \subseteq \text{gfp}(\Gamma_\psi)$. Hence, we obtain that $\mathfrak{A} \models_s [\text{GFP}_{\approx} R\bar{x} . \psi(R, \bar{x})](\bar{x})$ for every $s \in X$.

“ \Leftarrow ”: Now we assume that $\mathfrak{A} \models_s [\text{GFP}_{\approx} R\bar{x} . \psi(R, \bar{x})](\bar{x})$ for every $s \in X$. If $X = \emptyset$, then $\mathfrak{A} \models_X \psi^+(\bar{x})$ follows from the empty team property. Henceforth, let $X \neq \emptyset$. Let $\Gamma_\psi = \Gamma_\psi^{\mathfrak{A}}$ be the GFP_{\approx}^+ -update operator defined w.r.t. $\psi(R)$. From our assumption follows that $X(\bar{x}) \subseteq \text{gfp}(\Gamma_\psi)$. Since $X \neq \emptyset$, it follows that $\text{gfp}(\Gamma_\psi) \neq \emptyset$. Our goal is to prove that $\mathfrak{A} \models_X \psi^+(\bar{x})$. Towards this end, we define $F : X \rightarrow \mathcal{P}(A^n) \setminus \{\emptyset\}$, $F(s) := \text{gfp}(\Gamma_\psi)$ and $Y := X[\bar{y} \mapsto F]$ and claim that $\mathfrak{A} \models_Y \bar{x} \subseteq_{\approx} \bar{y} \wedge \exists \bar{z}(\bar{y} \approx \bar{z} \wedge \psi^*(\bar{z}))$. Since $Y(\bar{x}) = X(\bar{x}) \subseteq \text{gfp}(\Gamma_\psi) = Y(\bar{y})$ it is clear that $\mathfrak{A} \models_Y \bar{x} \subseteq_{\approx} \bar{y}$.

We still need to prove that $\mathfrak{A} \models_Y \exists \bar{z}(\bar{y} \approx \bar{z} \wedge \psi^*(\bar{z}))$. By definition of Y , we know that $Y(\bar{y}) = \text{gfp}(\Gamma_\psi) = \Gamma_\psi(\text{gfp}(\Gamma_\psi)) = \llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{x}) \rrbracket_{\approx}^{\mathfrak{A}}$. This implies that for every $s \in Y$ there exists some $\bar{a} \in \llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{x}) \rrbracket_{\approx}^{\mathfrak{A}}$ such that $\bar{a} \approx s(\bar{y})$.

Let $G : Y \rightarrow \mathcal{P}(A^n) \setminus \{\emptyset\}$ be given by

$$G(s) := \{\bar{a} \in \llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{x}) \rrbracket_{\approx}^{\mathfrak{A}} : s(\bar{y}) \approx \bar{a}\}$$

and $Z := Y[\bar{z} \mapsto G]$. Clearly it holds that $Z(\bar{z}) \subseteq \llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{x}) \rrbracket_{\approx}^{\mathfrak{A}}$. We claim that even $Z(\bar{z}) = \llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{x}) \rrbracket_{\approx}^{\mathfrak{A}}$ is true. To see this, let $\bar{a} \in \llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{x}) \rrbracket_{\approx}^{\mathfrak{A}} \subseteq \llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{x}) \rrbracket_{\approx}^{\mathfrak{A}} = Y(\bar{y})$. So there exists an $s \in Y$ with $s(\bar{y}) \approx \bar{a}$. Hence, we have that $\bar{a} \in G(s)$ and, consequently, $\bar{a} \in Z(\bar{z})$.

It is the case that $\mathfrak{A} \models_Z \bar{y} \approx \bar{z}$, because this follows from the definition of G . Now we prove that $\mathfrak{A} \models_Z \psi^*(\bar{z})$. By Lemma 17, we need to verify that $(\mathfrak{A}, Z(\bar{z})) \models_s \psi(R, \bar{z})$ for every $s \in Z$. In other words, we need to verify that $Z(\bar{z}) \subseteq \llbracket \psi(Z(\bar{z}), \bar{z}) \rrbracket_{\approx}^{\mathfrak{A}}$. Since $Z(\bar{z}) = \llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{x}) \rrbracket_{\approx}^{\mathfrak{A}}$, we can conclude that

$$\llbracket \psi(Z(\bar{z}), \bar{z}) \rrbracket_{\approx}^{\mathfrak{A}} = \llbracket \psi(\llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{x}) \rrbracket_{\approx}^{\mathfrak{A}}, \bar{z}) \rrbracket_{\approx}^{\mathfrak{A}}$$

Due to the fact that R occurs only \approx -guarded in ψ , we can observe that

$$\begin{aligned} \llbracket \psi(\llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{x}) \rrbracket_{\approx}^{\mathfrak{A}}, \bar{z}) \rrbracket_{\approx}^{\mathfrak{A}} &= \llbracket \psi(\llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{x}) \rrbracket_{\approx}^{\mathfrak{A}}, \bar{z}) \rrbracket_{\approx}^{\mathfrak{A}} \\ &= \llbracket \psi(\Gamma_\psi(\text{gfp}(\Gamma_\psi)), \bar{z}) \rrbracket_{\approx}^{\mathfrak{A}} \\ &= \llbracket \psi(\text{gfp}(\Gamma_\psi), \bar{z}) \rrbracket_{\approx}^{\mathfrak{A}} = Z(\bar{z}) \end{aligned}$$

Therefore, we have $Z(\bar{z}) = \llbracket \psi(Z(\bar{z}), \bar{z}) \rrbracket_{\approx}^{\mathfrak{A}}$ which implies that $Z(\bar{z}) \subseteq \llbracket \psi(Z(\bar{z}), \bar{z}) \rrbracket_{\approx}^{\mathfrak{A}}$. So we have $(\mathfrak{A}, Z(\bar{z})) \models_s \psi(R, \bar{z})$ for every $s \in Z$, which concludes the proof of $\mathfrak{A} \models_Z \psi^*$ and of $\mathfrak{A} \models_X \psi^+$. \blacktriangleleft

► **Corollary 19.** *For every GFP_{\approx}^+ -sentence φ there is an equivalent sentence $\vartheta \in \text{FO}(\subseteq_{\approx})$.*

Proof. Let $\varphi \in \text{GFP}_{\approx}^+$. By Lemma 16, there exists a first-order formula $\psi(R, \bar{x})$ where the n -ary relation symbol R occurs only positively and only \approx -guarded in ψ such that

$$\varphi \equiv \exists \bar{v}[\text{GFP}_{\approx} R\bar{x} . \psi(R, \bar{x})](\bar{v}).$$

W.l.o.g. we can assume that the variables in \bar{x} are never quantified in ψ . So, by Theorem 18, it follows that there exists some $\psi^+(\bar{x}) \in \text{FO}(\subseteq_{\approx})$ such that for every $\mathfrak{A} \in (\tau, \approx)$ and every team X over \mathfrak{A} with $\text{dom}(X) \supseteq \{\bar{x}\}$ holds

$$\mathfrak{A} \models_X \psi^+(\bar{x}) \iff \mathfrak{A} \models_s [\text{GFP}_{\approx} R\bar{x} . \psi(R, \bar{x})](\bar{x}) \text{ for every } s \in X$$

Let $\vartheta := \exists \bar{v} \psi^+(\bar{v})$ and $\mathfrak{A} \in (\tau, \approx)$. Our goal is to prove that $\mathfrak{A} \models \varphi \iff \mathfrak{A} \models \vartheta$.

“ \Leftarrow ”:

Let $\mathfrak{A} \models \vartheta$. Then there exists a function $F : \{\emptyset\} \rightarrow \mathcal{P}(A^{|\bar{v}|}) \setminus \{\emptyset\}$ such that $\mathfrak{A} \models_Y \psi^+(\bar{v})$ where $Y = \{\emptyset\}[\bar{v} \mapsto F]$. Then we have $\mathfrak{A} \models_s [\text{GFP}_{\approx} R\bar{x}. \psi(R, \bar{x})](\bar{v})$ for every $s \in Y$ and, since Y is non-empty, it follows that $\mathfrak{A} \models \exists \bar{v} [\text{GFP}_{\approx} R\bar{x}. \psi(R, \bar{x})](\bar{v})$.

“ \Rightarrow ”:

Now let $\mathfrak{A} \models \varphi \equiv \exists \bar{v} [\text{GFP}_{\approx} R\bar{x}. \psi(R, \bar{x})](\bar{v})$. Then there exists some $\bar{a} \in A$ such that $\mathfrak{A} \models [\text{GFP}_{\approx} R\bar{x}. \psi(R, \bar{x})](\bar{a})$. Let $Y = \{s\}$ be the singleton team consisting only of s with $s(\bar{v}) = \bar{a}$. Then it follows that $\mathfrak{A} \models_s [\text{GFP}_{\approx} R\bar{x}. \psi(R, \bar{x})](\bar{v})$ for every $s \in Y$ and, consequently, $\mathfrak{A} \models_Y \psi^+(\bar{v})$, proving that $\mathfrak{A} \models_{\{\emptyset\}} \exists \bar{v} \psi^+(\bar{v}) = \vartheta$. \blacktriangleleft

5 $\Sigma_1^1(\approx)$ on restricted classes of structures

In this section we compare $\Sigma_1^1(\approx)$ with FO and Σ_1^1 and study how restrictions imposed on the given equivalence influence the expressive power of $\Sigma_1^1(\approx)$. Our first result is that the expressive power of $\Sigma_1^1(\approx) \equiv \text{FO}(\subseteq_{\approx}, |\approx)$ lies strictly between FO and Σ_1^1 . Furthermore, we also have $\text{FO} < \text{FO}(\subseteq_{\approx}, |\approx) < \Sigma_1^1$ on the class of structures with only a bounded number of non-trivial equivalence classes and on the class of structures where each equivalence class is of size $\leq k$ (for some fixed $k > 1$). However, when restricting both the size of the equivalence classes and the number of non-trivial equivalence classes, then $\text{FO}(\subseteq_{\approx}, |\approx)$ has the same expressive power as Σ_1^1 . To prove these results, we use an adaption of the Ehrenfeucht-Fraïssé method for $\text{FO}(\subseteq_{\approx}, |\approx)$, which relies on the games presented in [15].

► Definition 20. Let $\mathfrak{A}, \mathfrak{B} \in (\tau, \approx)$, $n \in \mathbb{N}$ and $\Omega_{\approx} \subseteq \{\text{dep}_{\approx}, \perp_{\approx}, \subseteq_{\approx}, |\approx\}$. The game $\mathcal{G}_{\Omega_{\approx}, n}(\mathfrak{A}, \mathfrak{B})$ is played by two players which are called Duplicator and Spoiler. The positions of the game are tuples (X, Y) of teams over $\mathfrak{A}, \mathfrak{B}$ with $\text{dom}(X) = \text{dom}(Y)$. Unless stated otherwise the game starts at position $(\{\emptyset\}, \{\emptyset\})$ and then n moves are played. In each move Spoiler always chooses between one of the following 3 moves to continue the game:

1. Move \vee :
 - Spoiler represents X as a union $X = X_0 \cup X_1$.
 - Duplicator replies with a representation of Y as $Y = Y_0 \cup Y_1$.
 - Spoiler chooses $i \in \{0, 1\}$ and the game continues at position (X_i, Y_i) .
2. Move \exists :
 - Spoiler chooses a function $F : X \rightarrow \mathcal{P}(A) \setminus \{\emptyset\}$.
 - Duplicator replies with a function $G : Y \rightarrow \mathcal{P}(B) \setminus \{\emptyset\}$.
 - The game continues at position $(X[v \mapsto F], Y[v \mapsto G])$ where v is a new variable.
3. Move \forall :
 - The game continues at position $(X[v \mapsto A], Y[v \mapsto B])$ where v is a new variable.

Positions (X, Y) with $\mathfrak{A} \models_X \vartheta$ but $\mathfrak{B} \not\models_Y \vartheta$ for some literal $\vartheta \in \text{FO}(\Omega_{\approx})$ are Spoiler’s winning position. Duplicator wins, if such positions are avoided for n moves.

The game $\mathcal{G}_{\Omega_{\approx}}(\mathfrak{A}, \mathfrak{B})$ is played similarly: first Spoiler chooses a number $n \in \mathbb{N}$ and then $\mathcal{G}_{\Omega_{\approx}, n}(\mathfrak{A}, \mathfrak{B})$ is played.

These games characterize semi-equivalences of \mathfrak{A} and \mathfrak{B} (up to a certain depth). The depth of $\varphi \in \text{FO}(\Omega_{\approx})$, denoted as $\text{depth}(\varphi)$, is defined inductively:

$$\begin{aligned} \text{depth}(\vartheta) &:= 0 \text{ for every literal } \vartheta \in \text{FO}(\Omega_{\approx}) \\ \text{depth}(\exists v \varphi') &:= \text{depth}(\varphi') + 1 =: \text{depth}(\forall v \varphi') \\ \text{depth}(\varphi_1 \vee \varphi_2) &:= \max(\text{depth}(\varphi_1), \text{depth}(\varphi_2)) + 1 \\ \text{depth}(\varphi_1 \wedge \varphi_2) &:= \max(\text{depth}(\varphi_1), \text{depth}(\varphi_2)) \end{aligned}$$

► **Definition 21** (Semi-equivalence, [15]). Let $\mathfrak{A}, \mathfrak{B} \in (\tau, \approx)$ and X, Y be teams over $\mathfrak{A}, \mathfrak{B}$ with $\text{dom}(X) = \text{dom}(Y)$. We write $\mathfrak{A}, X \Rightarrow_{\Omega_{\approx}, n} \mathfrak{B}, Y$ (and say that \mathfrak{A}, X is semi-equivalent to \mathfrak{B}, Y up to depth n), if $\mathfrak{A} \models_X \varphi$ implies $\mathfrak{B} \models_Y \varphi$ for every $\varphi \in \text{FO}(\Omega_{\approx})$ with $\text{depth}(\varphi) \leq n$. Furthermore, we write $\mathfrak{A}, X \Rightarrow_{\Omega_{\approx}} \mathfrak{B}, Y$, if $\mathfrak{A}, X \Rightarrow_{\Omega_{\approx}, n} \mathfrak{B}, Y$ for every $n \in \mathbb{N}$. When Ω_{\approx} is clear from the context, we sometimes omit it as a subscript.

In first-order logic, the concept of semi-equivalence coincides with the usual equivalence concept between structures, but this is not the case in logics with team semantics. For example $\mathfrak{A}, X \Rightarrow \mathfrak{B}, \emptyset$ follows from the empty team property, but $\mathfrak{B}, \emptyset \Rightarrow \mathfrak{A}, X$ is not true in general. We write $\mathfrak{A}, X \equiv_n \mathfrak{B}, Y$, if $\mathfrak{A}, X \Rightarrow_n \mathfrak{B}, Y$ and $\mathfrak{B}, Y \Rightarrow_n \mathfrak{A}, X$. $\mathfrak{A}, X \equiv \mathfrak{B}, Y$ is defined analogously.

► **Theorem 22.** *Let τ be a finite signature and $\mathfrak{A}, \mathfrak{B} \in (\tau, \approx)$. Duplicator has a winning strategy for $\mathcal{G}_{\Omega_{\approx}, n}(\mathfrak{A}, \mathfrak{B})$ from position (X, Y) if, and only if $\mathfrak{A}, X \Rightarrow_{\Omega_{\approx}, n} \mathfrak{B}, Y$.*

Having these games at our disposal, we can prove that $\text{FO}(\subseteq_{\approx}, |\approx|)$ is strictly less powerful than Σ_1^1 . Consider the following problem:

$$\mathcal{C}_{\text{even}} := \{\mathfrak{A} \in (\tau, \approx) : \text{there is some } a \in A \text{ such that } |[a]_{\approx}| \text{ is even}\}.$$

► **Theorem 23.** *$\mathcal{C}_{\text{even}}$ is not expressible in $\text{FO}(\subseteq_{\approx}, |\approx|)$.*

We just give a short sketch of the proof: Consider $\mathfrak{A}_m := (A_m, \approx^{\mathfrak{A}_m})$ and $\mathfrak{B}_m := (B_m, \approx^{\mathfrak{B}_m})$ where $|A_m| = 2m$, $|B_m| = 2m + 1$, $\approx^{\mathfrak{A}_m} := A_m \times A_m$ and $\approx^{\mathfrak{B}_m} := B_m \times B_m$. Then $\mathfrak{A}_m \in \mathcal{C}_{\text{even}}$ while $\mathfrak{B}_m \notin \mathcal{C}_{\text{even}}$. It is not difficult to prove that Duplicator wins the games $\mathcal{G}_m(\mathfrak{A}_m, \mathfrak{B}_m)$ and $\mathcal{G}_m(\mathfrak{B}_m, \mathfrak{A}_m)$ by maintaining as an invariant that the equality types induced by the assignments in the two teams are always equal. On the other hand, it is easy to see that $\text{FO}(\text{dep}_{\approx}) \leq \text{FO}(\subseteq_{\approx}, |\approx|)$ can express that the number of equivalence classes is even, but this is not definable in first-order logic.

► **Corollary 24.** $\text{FO} < \text{FO}(\subseteq_{\approx}, |\approx|) < \Sigma_1^1$.

Next we study whether restrictions imposed on the given equivalence influence the expressive power of Σ_1^1 . Consider the class $\mathcal{K}_{\leq p}$ of structures $\mathfrak{A} \in (\tau, \approx)$ where every equivalence class of \mathfrak{A} is of size $\leq p$. On $\mathcal{K}_{\leq 1}$, $\Sigma_1^1(\approx)$ has the same expressive power as Σ_1^1 , because every relation over $\mathfrak{A} \in \mathcal{K}_{\leq 1}$ is $\approx^{\mathfrak{A}}$ -closed. However, this is not the case for $p \geq 2$ as the next theorem shows.

► **Theorem 25.** *Let $p \geq 2$. $\text{FO} < \text{FO}(\subseteq_{\approx}, |\approx|) < \Sigma_1^1$ holds on the class $\mathcal{K}_{\leq p}$ of structures $\mathfrak{A} \in (\tau, \approx)$ with $|[a]_{\approx}| \leq p$ for every $a \in A$.*

To prove this (see appendix), we are using an Ehrenfeucht-Fraïssé argument and prove that $\text{FO}(\subseteq_{\approx}, |\approx|)$ is unable to express non-connectivity of graphs when the equivalence classes are allowed to contain up to 2 elements.

Restricting the number of equivalence classes is not really interesting, because it leads to a situation where $\Sigma_1^1(\approx)$ has the same expressive power as FO, because there are only $2^{(k^r)}$ many \approx -closed relations of arity r when k is the number of \approx -classes, which can be simulated in first-order logic.

Another possible restriction is to admit only a bounded number of non-trivial equivalence classes (which consist of more than one element). Let $\mathcal{K}_{\text{NT} \leq p}$ be the class of all $\mathfrak{A} \in (\tau, \approx)$ with at most p many non-trivial equivalence classes (for some $p \geq 1$).

But then again, $\mathcal{C}_{\text{even}} \cap \mathcal{K}_{\text{NT} \leq p}$ is not definable in $\text{FO}(\subseteq_{\approx}, |\approx|)$ on $\mathcal{K}_{\text{NT} \leq p}$. Hence, we also have $\text{FO} < \Sigma_1^1(\approx) < \Sigma_1^1$ on $\mathcal{K}_{\text{NT} \leq p}$.

However, combining the conditions imposed on the number of non-trivial equivalence and their size, leads to an interesting situation: $\Sigma_1^1(\approx) \equiv \Sigma_1^1$ on $\mathcal{K}_{\text{NT} \leq p_1, \leq p_2} := \mathcal{K}_{\text{NT} \leq p_1} \cap \mathcal{K}_{\leq p_2}$. The reason for this is that at most $p_1 \cdot p_2$ many elements are located inside non-trivial equivalence classes, while all the other elements are only equivalent to themselves. Since $\Sigma_1^1(\approx)$ allows us to obtain a linear order on the equivalence classes, it is possible to encode arbitrary relations and, hence, to simulate Σ_1^1 .

References

- 1 A. Blass and Y. Gurevich. Henkin quantifiers and complete problems. *Annals of Pure and Applied Logic*, 32:1–16, 1986.
- 2 H. Enderton. Finite partially ordered quantifiers. *Z. Math. Logik*, 16:393–397, 1970.
- 3 F. Engström. Generalized quantifiers in dependence logic. *Journal of Logic, Language, and Information*, 2012.
- 4 S. Abramsky et al., editor. *Dependence Logic. Theory and Applications*. Birkhäuser, 2016.
- 5 P. Galliani. Inclusion and exclusion in team semantics — on some logics of imperfect information. *Annals of Pure and Applied Logic*, 163:68–84, 2012.
- 6 P. Galliani and L. Hella. Inclusion Logic and Fixed Point Logic. In *Computer Science Logic 2013*, pages 281–295, 2013.
- 7 E. Grädel. Games for inclusion logic and fixed-point logic. In S. Abramsky et al., editor, *Dependence Logic. Theory and Applications*. Birkhäuser, 2016.
- 8 E. Grädel and J. Väänänen. Dependence and independence. *Studia Logica*, 101(2):399–410, 2013.
- 9 L. Henkin. Some remarks on infinitely long formulas. *Journal of Symbolic Logic*, pages 167–183, 1961.
- 10 J. Hintikka and G. Sandu. Informational independence as a semantical phenomenon. In *Studies in Logic and Foundations of Mathematics*, volume 126, pages 571–589. North-Holland, 1989.
- 11 W. Hodges. Compositional semantics for a logic of imperfect information. *Logic Journal of IGPL*, 5:539–563, 1997.
- 12 J. Kontinen and J. Väänänen. On definability in dependence logic. *Journal of Logic, Language, and Information*, 18:317–241, 2009.
- 13 A. Mann, G. Sandu, and M. Sevenster. *Independence-Friendly Logic. A Game-Theoretic Approach*, volume 386 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, 2012.
- 14 R. Rönholm. Capturing k -ary existential second-order logic with k -ary inclusion-exclusion logic. *Ann. Pure Appl. Logic*, 169(3):177–215, 2018.
- 15 J. Väänänen. *Dependence logic: A new approach to independence friendly logic*, volume 70. Cambridge University Press, 2007.
- 16 W. Walkoe. Finite partially-ordered quantification. *Journal of Symbolic Logic*, 35:535–555, 1970.

A Appendix

A.1 The Expressive Power of $\text{FO}(\subseteq_{\approx}, |\approx)$

Proof of Theorem 8. By induction:

Case $\varphi = \bar{v} \mid_{\approx} \bar{w}$: Then $\varphi^* := R_{\varphi} \bar{v} \wedge \neg R_{\varphi} \bar{w}$. Let $R_{\varphi} := (X(\bar{v}))_{\approx}$, which is the \approx -closure of $X(\bar{v})$. Now we observe that

$$\begin{aligned} \mathfrak{A} \models_X \bar{v} \mid_{\approx} \bar{w} &\iff \text{for every } s, s' \in X : s(\bar{v}) \not\approx s'(\bar{w}) \\ &\iff \text{for every } s \in X : s(\bar{v}) \in R_{\varphi} \text{ and } s(\bar{w}) \notin R_{\varphi} \\ &\iff (\mathfrak{A}, R_{\varphi}) \models_X R_{\varphi} \bar{v} \wedge \neg R_{\varphi} \bar{w}. \end{aligned}$$

Case φ is some FO-literal or an \subseteq_{\approx} -atom: Then we have $\varphi^* = \varphi$ and there is nothing to prove, because the relation symbols \bar{R} do not occur in φ^* .

Case $\varphi = \varphi_0 \vee \varphi_1$: Let $\vartheta_1^{(j)}, \dots, \vartheta_{k_j}^{(j)}$ be the exclusion atoms that occur in φ_j .

“(i) \implies (ii)”: First, we assume that $\mathfrak{A} \models_X \varphi_0 \vee \varphi_1$. Then there are teams X_0, X_1 such that $X = X_0 \cup X_1$ and $\mathfrak{A} \models_{X_j} \varphi_j$ for $j \in \{0, 1\}$. By induction hypothesis, there exists two tuples of \approx -closed relations $\bar{R}_j = (R_{\vartheta_1^{(j)}}, \dots, R_{\vartheta_{k_j}^{(j)}})$ such that $(\mathfrak{A}, \bar{R}_j) \models_{X_j} \varphi_j^*$ (for $j \in \{0, 1\}$).

We define

$$\bar{R} := (R_{\vartheta_1^{(0)}}, \dots, R_{\vartheta_{k_0}^{(0)}}, R_{\vartheta_1^{(1)}}, \dots, R_{\vartheta_{k_1}^{(1)}})$$

and, since the relations \bar{R}_j do not occur in φ_{1-j} , it follows that $(\mathfrak{A}, \bar{R}) \models_{X_j} \varphi_j^*$ for $j \in \{0, 1\}$. Therefore, $(\mathfrak{A}, \bar{R}) \models_X \varphi^*$.

“(i) \impliedby (ii)”: For the other direction, let there be \approx -closed relations \bar{R} such that $(\mathfrak{A}, \bar{R}) \models_X \varphi^*$ and, hence, there are teams X_0, X_1 such that $X = X_0 \cup X_1$ and $(\mathfrak{A}, \bar{R}) \models_{X_j} \varphi_j^*$ for $j \in \{0, 1\}$. By induction hypothesis, this implies that $\mathfrak{A} \models_{X_j} \varphi_j$ for $j \in \{0, 1\}$, whence it follows that $\mathfrak{A} \models_X \varphi$.

The case where $\varphi = \varphi_0 \wedge \varphi_1$ is similar to the previous one, and the cases where $\varphi = \forall x \psi$ or $\varphi = \exists x \psi$ are trivial. \blacktriangleleft

Proof of Lemma 9. Let $\mathfrak{A} \in (\tau, \approx)$ and X be some team over \mathfrak{A} with $\text{free}(\varphi) \subseteq \text{dom}(X)$. Recall that $\vartheta_1, \dots, \vartheta_k$ are the inclusion atoms that occur in φ . Our goal is now to prove that the following statements are equivalent for every subformula ϑ of φ :

(1) $\mathfrak{A} \models_X \vartheta$

(2) There are \approx -closed relations $\bar{R} = (R_{\vartheta_1}, \dots, R_{\vartheta_k})$ over \mathfrak{A} such that $(\mathfrak{A}, \bar{R}) \models_X \vartheta^*$ and for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_{\vartheta_i}$ there exists some $s_{i, \bar{a}} \in X$ such that $(\mathfrak{A}, \bar{R}) \models_{s_{i, \bar{a}}[\bar{z}_i \rightarrow \bar{a}]} \vartheta^{(i)}$.

Whenever we are proving that (2) holds, we can often use that $\vartheta^{(i)}$ and ϑ^* are equivalent, if $\vartheta_i = \bar{x}_i \subseteq_{\approx} \bar{y}_i$ does not occur in ϑ . Hence, we only need to prove that $\mathfrak{A} \models_X \vartheta^*$ holds and that for every $i \in \{1, \dots, k\}$ such that ϑ_i occurs in ϑ and every $\bar{a} \in R_{\vartheta_i}$ there exists some $s_{i, \bar{a}} \in X$ with $\mathfrak{A} \models_{s_{i, \bar{a}}[\bar{z}_i \rightarrow \bar{a}]} \vartheta^{(i)}(\bar{z}_i)$.

For the empty team $X = \emptyset$, there is nothing to prove, because $\mathfrak{A} \models_{\emptyset} \vartheta$ follows from the empty team property and the empty relations trivially satisfy the conditions stated in (2). From now on we only consider non-empty teams X in this proof, which proceeds now by induction:

Case $\vartheta = \bar{v} \subseteq_{\approx} \bar{w}$: Then there exists a unique $\ell \in \{1, \dots, k\}$ such that $\vartheta_{\ell} = \vartheta$ and $R_{\vartheta_{\ell}} = R_{\vartheta}$. Recall that we have defined $\vartheta^* := R_{\vartheta} \bar{v}$ and

$$\vartheta^{(i)} := \begin{cases} \vartheta^*, & \text{if } i \neq \ell \\ \vartheta^* \wedge \bar{w} \approx \bar{z}_i, & \text{if } i = \ell. \end{cases}$$

“(1) \implies (2)”: Suppose $\mathfrak{A} \models_X \bar{v} \subseteq_{\approx} \bar{w}$. Then $X(\bar{v}) \subseteq_{\approx} X(\bar{w})$ and, this is why, setting $R_{\vartheta} := X(\bar{w})_{\approx}$ leads to $(\mathfrak{A}, R_{\vartheta}) \models_X R_{\vartheta} \bar{v} = \vartheta^*$. Moreover, for every $\bar{a} \in R_{\vartheta} = X(\bar{w})_{\approx}$ there exists, by definition of $X(\bar{w})_{\approx}$, an assignment $s \in X$ such that $s(\bar{w}) \approx \bar{a}$ and, hence, it holds that $(\mathfrak{A}, R_{\vartheta}) \models_{s[\bar{z}_\ell \mapsto \bar{a}]} \bar{w} \approx \bar{z}_\ell$, which leads to $(\mathfrak{A}, R_{\vartheta}) \models_{s[\bar{z}_\ell \mapsto \bar{a}]} \vartheta^{(\ell)}$. Since the other relations R_{ϑ_i} for $i \neq \ell$ occur neither in ϑ^* nor $\vartheta^{(i)}$, it does not matter what values they contain.

“(1) \longleftarrow (2)”: For the converse direction, we assume that there are \approx -closed relations \bar{R} such that $(\mathfrak{A}, \bar{R}) \models_X R_{\vartheta_\ell} \bar{v}$ and that for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_{\vartheta_i}$ there is some $s_{i, \bar{a}} \in X$ such that $(\mathfrak{A}, \bar{R}) \models_{s_{i, \bar{a}}[\bar{z}_i \mapsto \bar{a}]} \vartheta^{(i)}$. In particular, for every $\bar{a} \in R_{\vartheta_\ell}$ holds

$$(\mathfrak{A}, \bar{R}) \models_{s_{\ell, \bar{a}}[\bar{z}_\ell \mapsto \bar{a}]} \vartheta^{(\ell)} = R_{\vartheta_\ell} \bar{v} \wedge \bar{w} \approx \bar{z}_\ell.$$

Our goal is to prove that $\mathfrak{A} \models_X \bar{v} \subseteq_{\approx} \bar{w}$. Towards this end, let $s \in X$. Because $(\mathfrak{A}, \bar{R}) \models_X R_{\vartheta_\ell} \bar{v}$, it follows that $s(\bar{v}) \in R_{\vartheta_\ell}$. So for $s' := s_{\ell, s(\bar{v})} \in X$ holds $(\mathfrak{A}, \bar{R}) \models_{s'[\bar{z}_\ell \mapsto s(\bar{v})]} \bar{w} \approx \bar{z}_\ell$. Therefore, $s'(\bar{w}) \approx s(\bar{v})$. Since $s \in X$ was chosen arbitrarily, this proves that $\mathfrak{A} \models_X \bar{v} \subseteq_{\approx} \bar{w}$.

Case ϑ is an FO-literal: Then we have $\vartheta^* := \vartheta =: \vartheta^{(i)}$. For arbitrary (not necessarily \approx -closed) relations \bar{R} holds

$$\mathfrak{A} \models_X \vartheta \iff_{X \neq \emptyset} (\mathfrak{A}, \bar{R}) \models_s \vartheta \text{ for every } s \in X \text{ and}$$

for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_{\vartheta_i}$ exists $s \in X$

such that $(\mathfrak{A}, \bar{R}) \models_{s[\bar{z}_i \mapsto \bar{a}]} \vartheta = \vartheta^{(i)}$

Case $\vartheta = \psi_0 \vee \psi_1$: Let $\vartheta_1^{(j)}, \dots, \vartheta_k^{(j)}$ be the inclusion atoms that occur in ψ_j .

“(1) \implies (2)”: First we assume that $\mathfrak{A} \models_X \vartheta$. Then there are two teams Y_0, Y_1 such that $X = Y_0 \cup Y_1$ and $\mathfrak{A} \models_{Y_j} \psi_j$ for $j \in \{0, 1\}$.

By induction hypothesis, there are tuples of \approx -closed relations $\bar{R}^{(j)} = (R_{\vartheta_1}^{(j)}, \dots, R_{\vartheta_k}^{(j)})$ such that $(\mathfrak{A}, \bar{R}^{(j)}) \models_{Y_j} \psi_j$ (for $j \in \{0, 1\}$) and for every $i \in \{1, \dots, k\}$ and every $\bar{a} \in R_{\vartheta_i}^{(j)}$ there exists some $s \in Y_j$ such that $(\mathfrak{A}, \bar{R}^{(j)}) \models_{s[\bar{z}_i \mapsto \bar{a}]} \psi_j^{(i)}$. Let $\bar{R} = (R_{\vartheta_1}, \dots, R_{\vartheta_k})$ be a tuple of \approx -closed relations such that $R_{\vartheta_i} = R_{\vartheta_i}^{(j)} \iff \vartheta_i$ occurs in ψ_j .³

We are going to prove that $(\mathfrak{A}, \bar{R}) \models_X \vartheta^*$ and that for every $i \in \{1, \dots, k\}$ such that ϑ_i occurs in ϑ and every $\bar{a} \in R_{\vartheta_i}$, there exists some $s \in X$ such that $(\mathfrak{A}, \bar{R}) \models_{s[\bar{z}_i \mapsto \bar{a}]} \vartheta^{(i)}$. Since $(\mathfrak{A}, \bar{R}^{(j)}) \models_{Y_j} \psi_j^*$, we also have $(\mathfrak{A}, \bar{R}) \models_{Y_j} \psi_j^*$, because whenever a relation symbol R_{ϑ_i} occurs in ψ_j , it must be the case that ϑ_i occurs in ψ_j and, hence, $R_{\vartheta_i} = R_{\vartheta_i}^{(j)}$. Additionally we still have $X = Y_0 \cup Y_1$ and, thus, $(\mathfrak{A}, \bar{R}) \models_X \vartheta^*$.

Towards proving the second part, let $i \in \{1, \dots, k\}$ such that ϑ_i occurs in ϑ and $\bar{a} \in R_{\vartheta_i}$. There must be some (unique) $j \in \{0, 1\}$ such that ϑ_i occurs in ψ_j and $R_{\vartheta_i} = R_{\vartheta_i}^{(j)}$. We know already that there exists some $s \in Y_j$ that satisfies $(\mathfrak{A}, \bar{R}^{(j)}) \models_{s[\bar{z}_i \mapsto \bar{a}]} \psi_j^{(i)}$, which implies that $(\mathfrak{A}, \bar{R}) \models_{s[\bar{z}_i \mapsto \bar{a}]} \psi_j^{(i)}$. Furthermore, we have that $\vartheta^{(i)} := \psi_j^{(i)}$ and, consequently, it follows that $(\mathfrak{A}, \bar{R}) \models_{s[\bar{z}_i \mapsto \bar{a}]} \vartheta^{(i)}$, which is exactly what we wanted to achieve.

“(1) \longleftarrow (2)”: Suppose that there are \approx -closed relations \bar{R} such that $(\mathfrak{A}, \bar{R}) \models_X \vartheta^*$ and that for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_{\vartheta_i}$ there exists some $s_{i, \bar{a}} \in X$ such that $(\mathfrak{A}, \bar{R}) \models_{s_{i, \bar{a}}[\bar{z}_i \mapsto \bar{a}]} \vartheta^{(i)}$. Then there are some teams Y_0, Y_1 such that $Y = Y_0 \cup Y_1$ and $(\mathfrak{A}, \bar{R}) \models_{Y_j} \psi_j^*$ for $j \in \{0, 1\}$. Furthermore, by definition of $\vartheta^{(i)}$, for every $i \in \{1, \dots, k\}$ such that ϑ_i occurs in ϑ , there exists a (unique) $j(i) \in \{0, 1\}$ with $\vartheta^{(i)} = \psi_{j(i)}^{(i)}$. For $j \in \{0, 1\}$ let

$$Y'_j := \{s_{i, \bar{a}} : i \in \{1, \dots, k\}, \vartheta_i \text{ occurs in } \vartheta, \bar{a} \in R_{\vartheta_i} \text{ and } j(i) = j\} (\subseteq X).$$

³ Such \bar{R} exists, because ϑ_i cannot occur in both ψ_0 and ψ_1 .

It follows that $(\mathfrak{A}, \overline{R}) \models_s \psi_j^{(i)}$ for every $s \in Y_j'$, because every $s \in Y_j$ has the form $s = s_{i,\bar{a}}$ and we have that $(\mathfrak{A}, \overline{R}) \models_{s_{i,\bar{a}}} \vartheta^{(i)} = \psi_j^{(i)}$ ($= \psi_j^{(i)}$). Since $\psi_j^{(i)} \models \psi_j^*$, we can conclude that also $(\mathfrak{A}, \overline{R}) \models_{s_{i,\bar{a}}[\bar{z}_i \mapsto \bar{a}]} \psi_j^*$. This, together with the flatness property of FO, implies that $(\mathfrak{A}, \overline{R}) \models_{Z_j} \psi_j^*$ where $Z_j := Y_j \cup Y_j'$.

For $j \in \{0, 1\}$ let $\overline{R}_{(j)} = (R_{\vartheta_1}^{(j)}, \dots, R_{\vartheta_k}^{(j)})$ be given by

$$R_{\vartheta_i}^{(j)} := \begin{cases} R_{\vartheta_i}, & \text{if } \vartheta_i \text{ is a subformula of } \psi_j \\ \emptyset, & \text{otherwise.} \end{cases}$$

Because the relation symbol R_{ϑ_i} occurs in ψ_j^* if, and only if ϑ_i is a subformula of ψ_j , we still have $(\mathfrak{A}, \overline{R}_{(j)}) \models_{Z_j} \psi_j^*$ for $j \in \{0, 1\}$. Furthermore, for every $j \in \{0, 1\}$, every $i \in \{1, \dots, k\}$ and every $\bar{a} \in R_{\vartheta_i}^{(j)}$ it must be the case that ϑ_i is a subformula of ψ_j (otherwise we would have $R_{\vartheta_i}^{(j)} = \emptyset$, but this contradicts $\bar{a} \in R_{\vartheta_i}^{(j)}$) and, thus, it follows that $\vartheta^{(i)} = \psi_j^{(i)}$ and, therefore, $(\mathfrak{A}, \overline{R}_{(j)}) \models_{s_{i,\bar{a}}[\bar{z}_i \mapsto \bar{a}]} \psi_j^{(i)}$, because we have $(\mathfrak{A}, \overline{R}) \models_{s_{i,\bar{a}}[\bar{z}_i \mapsto \bar{a}]} \vartheta^{(i)}$ and $s_{i,\bar{a}} \in Y_j' \subseteq Z_j$.

This is the reason, why we are allowed to use the induction hypothesis, which yields us that $\mathfrak{A} \models_{Z_j} \psi_j$ for $j \in \{0, 1\}$. Consequently, it follows that $\mathfrak{A} \models_{Z_0 \cup Z_1} \vartheta$.

It is easy to observe that $Z_0 \cup Z_1 = Y_0 \cup Y_1 \cup Y_0' \cup Y_1' = X$, because $Y_0', Y_1' \subseteq X$ and $X = Y_0 \cup Y_1$. As a result, we obtain that $\mathfrak{A} \models_X \vartheta$.

Case $\vartheta = \psi_0 \wedge \psi_1$: Similar and even easier than the previous case!

Case $\vartheta = \exists x\psi$: Recall that we have defined $\vartheta^* := \exists x\psi^*$ and $\vartheta^{(i)} = \exists x\psi^{(i)}$ for every $i \in \{1, \dots, k\}$. We only prove “(1) \Leftarrow (2)”, since the other direction is quite trivial.

Suppose that there are \approx -closed relations \overline{R} such that $(\mathfrak{A}, \overline{R}) \models_X \exists x\psi^*$ and for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_{\vartheta_i}$ there exists an $s_{i,\bar{a}} \in X$ such that $(\mathfrak{A}, \overline{R}) \models_{s_{i,\bar{a}}[\bar{z}_i \mapsto \bar{a}]} \exists x\psi^{(i)}$. Then there is a function $F : X \rightarrow \mathcal{P}(A) \setminus \{\emptyset\}$ such that for $Y := X[x \mapsto F]$ holds $(\mathfrak{A}, \overline{R}) \models_Y \psi^*$ and for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_{\vartheta_i}$ there exists some $b_{i,\bar{a}} \in A$ such that $(\mathfrak{A}, \overline{R}) \models_{s'_{i,\bar{a}}[\bar{z}_i \mapsto \bar{a}]} \psi^{(i)}$ where $s'_{i,\bar{a}} = s_{i,\bar{a}}[x \mapsto b_{i,\bar{a}}]$. Let $Y' := \{s'_{i,\bar{a}} : i \in \{1, \dots, k\}, \bar{a} \in R_{\vartheta_i}\}$. Due to $\psi^{(i)} \models \psi^*$ and the flatness property of FO, it follows that $(\mathfrak{A}, \overline{R}) \models_Z \psi^*$ for $Z := Y \cup Y'$. Furthermore, for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_{\vartheta_i}$ we have $s'_{i,\bar{a}} \in Y' \subseteq Z$ with $(\mathfrak{A}, \overline{R}) \models_{s'_{i,\bar{a}}[\bar{z}_i \mapsto \bar{a}]} \psi^{(i)}$. Thus, we can apply the induction hypothesis with Z and $\tilde{\vartheta}$ to obtain that $\mathfrak{A} \models_Z \psi$. By definition of Z we have $Z \upharpoonright \text{dom}(X) = (Y \upharpoonright \text{dom}(X)) \cup (Y' \upharpoonright \text{dom}(X))$. Furthermore, it is the case that $Y \upharpoonright \text{dom}(X) = X[x \mapsto F] \upharpoonright \text{dom}(X) = X$ (recall that we assume that no variable is quantified twice and that $\text{dom}(X) = \text{free}(\vartheta)$) and $Y' \upharpoonright \text{dom}(X) \subseteq X$, because every $s' \in Y'$ has the form $s' = s'_{i,\bar{a}} = s_{i,\bar{a}}[x \mapsto b_{i,\bar{a}}]$ where $s_{i,\bar{a}} \in X$. Therefore, $Z \upharpoonright \text{dom}(X) = X$ and, hence, it follows that $\mathfrak{A} \models_X \exists x\psi = \vartheta$.

Case $\vartheta = \forall x\psi$: Recall that we have defined $\vartheta^* := \forall x\psi^*$ and $\vartheta^{(i)} := \exists x(\psi^{(i)} \wedge \forall x(\psi^*))$.

“(1) \implies (2)”: Let $\mathfrak{A} \models_X \forall x\psi$. Then $\mathfrak{A} \models_Y \psi$ where $Y := X[x \mapsto A]$. By induction hypothesis, there are \approx -closed relations \overline{R} such that $(\mathfrak{A}, \overline{R}) \models_Y \psi^*$ and for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_{\vartheta_i}$ there exists an $s'_{i,\bar{a}} \in Y$ that satisfies $(\mathfrak{A}, \overline{R}) \models_{s'_{i,\bar{a}}[\bar{z}_i \mapsto \bar{a}]} \psi^{(i)}$.

Since $Y = X[x \mapsto A]$, it follows that $(\mathfrak{A}, \overline{R}) \models_X \forall x\psi^* = \vartheta^*$. We have already mentioned above that $\vartheta^* \in \text{FO}$. So we can use the flatness property, which leads to $(\mathfrak{A}, \overline{R}) \models_s \forall x\psi^*$ for every $s \in X$. In particular, this holds for the assignments $s_{i,\bar{a}} := (s'_{i,\bar{a}} \upharpoonright \text{dom}(X)) \in X$. This is why, we have $(\mathfrak{A}, \overline{R}) \models_{s_{i,\bar{a}}} \forall x\psi^*$. Furthermore, from $(\mathfrak{A}, \overline{R}) \models_{s'_{i,\bar{a}}[\bar{z}_i \mapsto \bar{a}]} \psi^{(i)}$ follows that $(\mathfrak{A}, \overline{R}) \models_{s_{i,\bar{a}}[\bar{z}_i \mapsto \bar{a}]} \exists x\psi^{(i)}$. Consequently, we can conclude that $(\mathfrak{A}, \overline{R}) \models_{s_{i,\bar{a}}} \exists x(\psi^{(i)} \wedge \forall x(\psi^*)) = \vartheta^{(i)}$ for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_{\vartheta_i}$ and we have that $(\mathfrak{A}, \overline{R}) \models_X \vartheta^*$.

“(1) \Leftarrow (2)”: Suppose that there are \approx -closed relations \bar{R} satisfying $(\mathfrak{A}, \bar{R}) \models_X \forall x \psi^*$ and for every $i \in \{1, \dots, k\}$, $\bar{a} \in R_{\vartheta_i}$ there exists some $s_{i, \bar{a}} \in X$ such that

$$(\mathfrak{A}, \bar{R}) \models_{s_{i, \bar{a}}[\bar{z}_i \mapsto \bar{a}]} \vartheta^{(i)} = \exists x (\psi^{(i)} \wedge \vartheta^*.$$

Let $Y := X[x \mapsto A]$. Then we have $(\mathfrak{A}, \bar{R}) \models_Y \psi^*$ (because $(\mathfrak{A}, \bar{R}) \models_X \forall x \psi^*$). Furthermore, for every $i \in \{1, \dots, k\}$, $\bar{a} \in A$ there exists some $b_{i, \bar{a}} \in A$ such that $(\mathfrak{A}, \bar{R}) \models_{s'_{i, \bar{a}}[\bar{z}_i \mapsto \bar{a}]} \vartheta^{(i)}$ where $s'_{i, \bar{a}} := s_{i, \bar{a}}[x \mapsto b_{i, \bar{a}}] \in Y$ (because $(\mathfrak{A}, \bar{R}) \models_{s_{i, \bar{a}}[\bar{z}_i \mapsto \bar{a}]} \exists x \psi^{(i)}$). So, by induction hypothesis, it follows that $\mathfrak{A} \models_Y \psi$ and, because of $Y = X[x \mapsto A]$, we obtain that $\mathfrak{A} \models_X \forall x \psi = \vartheta$. \blacktriangleleft

A.2 $\Sigma_1^1(\approx)$ on restricted classes of structures

Proof of Theorem 25. It suffices to prove this for $p = 2$. Let $\tau = \{E, \approx\}$. Consider the following problem: $\mathcal{C} := \{\mathfrak{A} \in \mathcal{K}_{\leq 2} : (A, E^{\mathfrak{A}}) \text{ is not connected}\}$. By using the method of Ehrenfeucht-Fraïssé we will show that \mathcal{C} is not definable in $\text{FO}(\subseteq_{\approx}, |\approx)$.

For every $m > 3$ let $\mathfrak{A}_m := (A_m, E^{\mathfrak{A}_m}, \approx)$ and $\mathfrak{B}_m := (B_m, E^{\mathfrak{B}_m}, \approx)$ where $A_m := \{0, \dots, m-1\} \cup \{0', \dots, (m-1)'\} =: B_m$ and $E^{\mathfrak{A}_m} := E_+^{\mathfrak{A}_m} \cup E_-^{\mathfrak{A}_m}$ with

$$E_+^{\mathfrak{A}_m} := \{(i, j), (i', j') : j = i + 1 \pmod{m}\}$$

and $E_-^{\mathfrak{A}_m} := \{(w, v) : (v, w) \in E_+^{\mathfrak{A}_m}\}$. Similarly, $E^{\mathfrak{B}_m} := E_+^{\mathfrak{B}_m} \cup E_-^{\mathfrak{B}_m}$ where $E_+^{\mathfrak{B}_m} := \{(0, 1), (1, 2), \dots, (m-2, m-1), (m-1, 0'), (0', 1'), \dots, ((m-2)', (m-1)'), ((m-1)', 0)\}$ and $E_-^{\mathfrak{B}_m} := \{(w, v) : (v, w) \in E_+^{\mathfrak{B}_m}\}$. \approx is in both structures defined such that $[i]_{\approx} = \{i, i'\}$ for every $i \in \{0, \dots, m-1\}$. In other words, \mathfrak{A}_m consists of two cycles $(0, 1, \dots, m-1, 0)$ and $(0', 1', \dots, (m-1)', 0')$ of length m , while \mathfrak{B}_m is a single cycle $(0, 1, \dots, m-1, 0', 1', \dots, (m-1)', 0)$ of length $2m$.

For every $v \in \{0, 1, \dots, m-1, 0', 1', \dots, (m-1)'\}$ there are uniquely determined $s^{\mathfrak{A}_m}(v)$ and $s^{\mathfrak{B}_m}(v)$ such that $(v, s^{\mathfrak{A}_m}(v)) \in E_+^{\mathfrak{A}_m}$ and $(v, s^{\mathfrak{B}_m}(v)) \in E_+^{\mathfrak{B}_m}$. Similarly, there exists uniquely determined predecessors $(s^{\mathfrak{A}_m})^{-1}(v)$ and $(s^{\mathfrak{B}_m})^{-1}(v)$ with $(v, (s^{\mathfrak{A}_m})^{-1}(v)) \in E_-^{\mathfrak{A}_m}$ and $(v, (s^{\mathfrak{B}_m})^{-1}(v)) \in E_-^{\mathfrak{B}_m}$. We define for every $v \in A_m$, every $w \in B_m$ and every $k \in \mathbb{Z}$

$$v +_{\mathfrak{A}_m} k := (s^{\mathfrak{A}_m})^k(v) \text{ and } w +_{\mathfrak{B}_m} k := (s^{\mathfrak{B}_m})^k(w).$$

We are going omit \mathfrak{A}_m and \mathfrak{B}_m as a subscript, when it is clear from the context that v belongs to \mathfrak{A}_m resp. \mathfrak{B}_m .

For $v, w \in A_m$ we define $\text{dist}_{\mathfrak{A}_m}(v, w)$ to be the minimal number $n \in \mathbb{N}$ such that $v + n = w$ or $v - n = w$, or ∞ , if no such number $n \in \mathbb{N}$ exists. $\text{dist}_{\mathfrak{B}_m}(v, w)$ is defined analogously. Please note, that $\text{dist}_{\mathfrak{A}_m}(v, w) = \text{dist}_{\mathfrak{A}_m}(w, v)$ and $\text{dist}_{\mathfrak{B}_m}(v, w) = \text{dist}_{\mathfrak{B}_m}(w, v)$. Furthermore, for every $a \in \{0, 1, \dots, m-1, 0', 1', \dots, (m-1)'\}$ and every $b, c \in \mathbb{Z}$ holds,

$$(a +_{\mathfrak{A}_m} b) +_{\mathfrak{A}_m} c = a +_{\mathfrak{A}_m} (b + c) \text{ and } (a +_{\mathfrak{B}_m} b) +_{\mathfrak{B}_m} c = a +_{\mathfrak{B}_m} (b + c).$$

It is easy to see that $\text{dist}(v_1, v_3) \leq \text{dist}(v_1, v_2) + \text{dist}(v_2, v_3)$ for every v_1, v_2, v_3 from A_m or B_m . Furthermore, $v \approx w$ implies that $s^{\mathfrak{A}_m}(v) \approx s^{\mathfrak{B}_m}(v)$ and $(s^{\mathfrak{A}_m})^{-1}(v) \approx (s^{\mathfrak{B}_m})^{-1}(v)$. This observation leads to the following corollary.

► **Claim 26.** Let $v \in A_m, w \in B_m$ with $v \approx w$. Then $v + k \approx w + k$ for every $k \in \mathbb{Z}$.

For every $i, j, q \in \mathbb{N}$ we write $i \approx_q j$ if, and only if $i = j$ or $i \geq q \leq j$. Given two assignments $s : \{x_1, \dots, x_\ell\} \rightarrow A_m$ and $t : \{x_1, \dots, x_\ell\} \rightarrow B_m$, we write $s \approx_q t$ if, and only if $s(x_i) \approx t(x_i)$ (which is equivalent to: $s(x_i), t(x_i) \in \{n, n'\}$ for some $n \in \{0, \dots, m-1\}$) and $\text{dist}_{\mathfrak{A}_m}(s(x_i), s(x_j)) \approx_q \text{dist}_{\mathfrak{B}_m}(t(x_i), t(x_j))$ holds for every $i, j \in \{1, \dots, \ell\}$.

► **Lemma 27.** *Let $m > 2^{n+2}$ and $0 \leq \ell \leq k < n$. Furthermore, let $s : \{x_1, \dots, x_\ell\} \rightarrow A_m$ and $t : \{x_1, \dots, x_\ell\} \rightarrow B_m$ be two assignments with $s \approx_{2^{n+1-k}} t$. Then:*

(1) *For every $a \in A_m$ there exists some $b = b(s, t, a) \in B_m$ such that*

$$s' := s[x_{\ell+1} \mapsto a] \approx_{2^{n-k}} t[x_{\ell+1} \mapsto b] =: t'.$$

(2) *For every $b \in B_m$ there exists some $a = a(s, t, b) \in A_m$ such that*

$$s' := s[x_{\ell+1} \mapsto a] \approx_{2^{n-k}} t[x_{\ell+1} \mapsto b] =: t'.$$

Furthermore, for two teams X, Y over $\mathfrak{A}_m, \mathfrak{B}_m$ with $\text{dom}(X) = \{x_1, \dots, x_\ell\} = \text{dom}(Y)$ we write $X \approx_q Y$ if, and only if for every $s \in X$ there exists some $t \in Y$ and, conversely, for every $t \in Y$ there exists some $s \in X$ such that $s \approx_q t$.

► **Claim 28.** Let $n, m \in \mathbb{N}$ with $m > 2^{n+2}$. Duplicator has a winning strategy in $\mathcal{G}_n(\mathfrak{A}_m, \mathfrak{B}_m)$.

Thus we have $\mathfrak{A}_m \equiv_n \mathfrak{B}_m$ for every $m > 2^{n+2}$. Using very similar arguments, it is possible to prove that $\mathfrak{B}_m \equiv_n \mathfrak{A}_m$. Furthermore, we have $\mathfrak{A}_m \in \mathcal{C}$ and $\mathfrak{B}_m \notin \mathcal{C}$. This proves that \mathcal{C} is not definable in $\text{FO}(\subseteq_{\approx}, |\approx)$ (because φ is unable to distinguish between \mathfrak{A}_m and \mathfrak{B}_m for every $m > 2^{\text{depth}(\varphi)+2}$). On the other hand, \mathcal{C} is definable in Σ_1^1 by the sentence $\exists X \exists x \exists y (Xx \wedge \neg Xy \wedge \forall u \forall v (Xu \wedge Euv \rightarrow Xv))$. This concludes the proof of $\text{FO}(\subseteq_{\approx}, |\approx) < \Sigma_1^1$. $\text{FO} < \text{FO}(\subseteq_{\approx}, |\approx)$ follows from the fact that $\text{FO}(|\approx) \equiv \text{FO}(\text{dep}_{\approx})$ and that the sentence

$$\forall x \exists y \forall x' \exists y' (\text{dep}_{\approx}(x, y) \wedge \text{dep}_{\approx}(x', y') \wedge x \not\approx y \wedge (x \not\approx x' \vee y \approx y') \wedge (x \not\approx y' \vee y \approx x'))$$

expresses that a even number of equivalence classes exists. Using standard Ehrenfeucht-Fraïssé arguments, it is not difficult to prove, that \mathcal{C} is not FO-definable. ◀

Finite Bisimulations for Dynamical Systems with Overlapping Trajectories

Béatrice Bérard

Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6,
LIP6, F-75005 Paris, France
beatrice.berard@lip6.fr

Patricia Bouyer¹

LSV, CNRS, ENS Paris-Saclay, Univ. Paris-Saclay, France
bouyer@lsv.fr

Vincent Jugé

Université Paris-Est, LIGM (UMR 8049), CNRS, ENPC, ESIEE, UPEM, F-77454,
Marne-la-Vallée, France
vincent.juge@u-pem.fr

Abstract

Having a finite bisimulation is a good feature for a dynamical system, since it can lead to the decidability of the verification of reachability properties. We investigate a new class of o-minimal dynamical systems with very general flows, where the classical restrictions on trajectory intersections are partly lifted. We identify conditions, that we call *Finite* and *Uniform Crossing*: When Finite Crossing holds, the time-abstract bisimulation is computable and, under the stronger Uniform Crossing assumption, this bisimulation is finite and definable.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Reachability properties, dynamical systems, o-minimal structures, intersecting trajectories, finite bisimulations

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.26

Acknowledgements We would like to thank Françoise Point, who pointed out to us reference [33], and anonymous referees for providing additional references.

1 Introduction

Hybrid automata. Hybrid systems [16] combine continuous dynamics, i.e. evolution of variables according to flow functions (possibly described by differential equations) in control locations, and discrete jumps between these locations, equipped with guards and variable updates. For this very expressive class of models, where the associated transition system has an uncountable state space, most verification questions are undecidable [19, 4], in particular the reachability of some error states. For the last twenty-five years, a large amount of research has been devoted to approximation methods [34, 12] and to the identification of subclasses with decidable properties obtained by restricting the continuous dynamics and/or the discrete behaviour of the systems [2]. Among these subclasses lie the well-known timed automata [1], where all variables are *clocks* evolving with rate 1 with respect to a global time, guards are comparisons of clocks with rational constants, and updates are resets. Decidability results

¹ Supported by ERC project EQualIS.



were also obtained for larger classes (see [18, 14, 25, 2, 8, 15, 10, 9, 3]), usually (but not always) by building a finite abstraction based on some *bisimulation* equivalence, preserving a specific class of properties, like reachability or those expressed by temporal logic formulas.

Ingredients for decidability results. We now describe the restrictions mentioned above. The first one consists in constraining the variable updates on discrete transitions between locations by some “strong reinitialization”, to make the dynamics of locations independent from each other, hence decoupling the discrete and continuous components. Considering a single location with its dynamics is then sufficient; in the next step, the aim is to identify subclasses of the dynamical systems governing the variables on a fixed location, for which a finite bisimulation can be found.

With the decoupling conditions, powerful flows, like the linear flows considered in [17], become possible. The approach in [2] handles o-minimal hybrid systems, using o-minimal structures over the reals as time and variable domains. The first-order theory of reals is then exploited to produce a finite bisimulation. This direction was further explored in [25, 8, 22, 10, 9], where analytical or algebraic methods are proposed to extend the set of flow functions as well as the underlying o-minimal structures. In [10, 9], decidability of reachability is even obtained with the theory of reals while no finite bisimulation may exist. The work of [15] explores how to slightly lift the hypothesis on strong reinitialization.

A few cases feature hybrid automata with no decoupling between the discrete and the continuous parts, at the price of very simple dynamical systems: the first one is the class of timed automata, where clocks describe the most basic flow functions, and the second one is the (incomparable) class of Interrupt Timed Automata with polynomial constraints [3], where the variables are stopwatches (with rate 0 or 1 depending on the location) organized along hierarchical levels. In this latter case, classical polyhedron-based abstractions are not sufficient and the finite bisimulation is obtained via an adaptation of the cylindrical algebraic decomposition algorithm [13].

Contribution. We investigate a new class of o-minimal dynamical systems, where some classical restrictions on the trajectories are lifted: overlapping trajectories are possible, as depicted for instance in Figure 1. Our method involves a classification of intersection points, similar to the cylindrical decomposition, producing a time-abstract bisimulation leading to a finite abstraction under suitable hypotheses.

Outline. In Section 2, we recall the base properties of o-minimal structures used in our developments; we then define the dynamical systems we will study; we also define the technical tool of time-abstract bisimulation which is used to build a finite abstraction of the dynamical systems; we end up this section with a discussion on related work. In Section 3, we present the graph construction, which leads to abstract the original dynamical system with some partition of the state-space, on which we are able to check time-abstract bisimulation. In Section 4, we discuss definability and decidability issues, and show how our approach can be used to recover the original work [25]. We end up with some perspectives.

2 Definitions

We consider linearly ordered structures $\mathcal{M} = \langle M, <, \dots \rangle$. These structures can be dense or discrete (or mixed), with or without endpoints (i.e. minimum or maximum). Classical examples without endpoints are the set \mathbb{Z} of integers or the real line \mathbb{R} , while the sets \mathbb{N}

of natural numbers and \mathbb{R}_+ of the non negative real numbers have 0 as left endpoint. We will consider the first-order theory associated with \mathcal{M} : we say that some relation, subset or function is *definable* when it is first-order definable in the structure \mathcal{M} . Next we may abusively identify the structure \mathcal{M} with its first-order theory. A general reference for first-order logic is [20]. Moreover, we will assume that the theory of \mathcal{M} is o-minimal and we recall here the definition of o-minimality (references are [31, 21, 32, 33, 36]).

2.1 O-minimal structures

Recall that *intervals* of $\mathcal{M} = \langle M, <, \dots \rangle$ are convex sets with either a supremum in M or no upper bound, and either an infimum in M or no lower bound.

► **Definition 1.** A linearly ordered structure $\mathcal{M} = \langle M, <, \dots \rangle$ has an *o-minimal* theory if every definable subset of M is a finite union of intervals.

In other words, the definable subsets of M are the simplest possible. This assumption implies that definable subsets of M^n (in the sense of \mathcal{M}) admit very nice structure theorems (like the *cell decomposition* [21, 32]). Classical o-minimal structures are: the ordered group of rationals $\langle \mathbb{Q}, <, +, 0, 1 \rangle$, the ordered field of reals $\langle \mathbb{R}, <, +, \cdot, 0, 1 \rangle$, the field of reals with exponential function, the field of reals expanded by restricted pfaffian functions and the exponential function, and many more interesting structures (see [36, 37]). An example of non o-minimal structure is given by $\langle \mathbb{R}, <, \sin, 0 \rangle$, since the definable set $\{x \mid \sin(x) = 0\}$ is not a finite union of intervals. However, note that the structure² $\langle \mathbb{R}, +, \cdot, 0, 1, <, \sin_{|[0,2\pi]}, \cos_{|[0,2\pi]} \rangle$ is o-minimal (see [35]).

We recall here a standard base result of o-minimal structures, used to build the cell decomposition, and which will be useful in the subsequent developments. While initially proved for dense structures [31, Theorem 4.2], a version for discrete structures is provided in [32, Lemmas 1.3 and 1.5], and the result holds for general mixed structures as a consequence of [33, Proposition 2.3].

► **Theorem 2.** Let $\mathcal{M} = \langle M, <, \dots \rangle$ be a linearly ordered structure with an o-minimal theory. Let $f : M \mapsto M$ be a definable function. The set M can be partitioned into finitely many intervals I_1, \dots, I_k such that, for every interval I_j , (i) the restriction $f|_{I_j}$ is either constant or one-to-one and monotonic, and (ii) the set $f(I_j)$ is an interval of M .

The other result on o-minimal structures used in the sequel is the following, restated from [33, Section 2], which provides a uniform bound on the partition size:

► **Theorem 3.** Let $\mathcal{M} = \langle M, <, \dots \rangle$ be a linearly ordered structure with an o-minimal theory. Let φ be a formula with k variables. Then there exists an integer \mathbf{N}_φ such that, for all $b_2, \dots, b_k \in M$, the set $\{a \in M \mid (a, b_2, \dots, b_k) \models \varphi\}$ can be partitioned into at most \mathbf{N}_φ intervals.

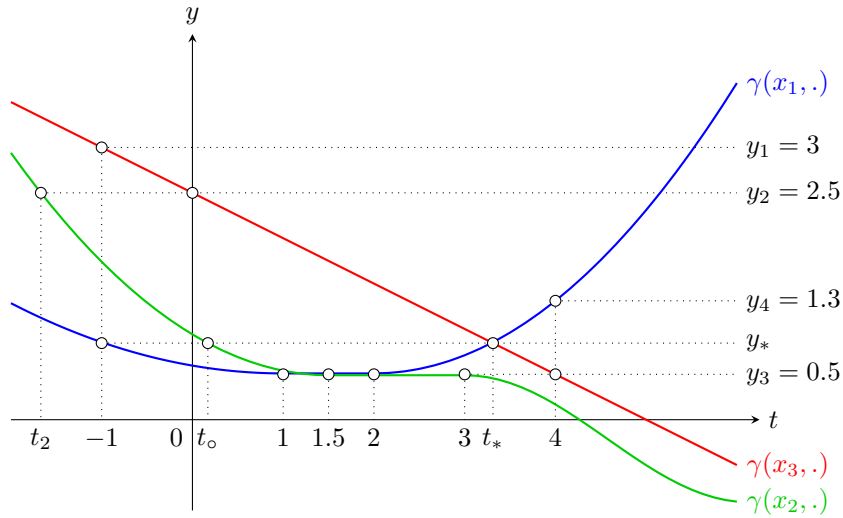
2.2 Dynamical systems

► **Definition 4.** A *dynamical system* is a pair (\mathcal{M}, γ) where:

- $\mathcal{M} = \langle M, <, \dots \rangle$ is a linearly ordered structure,
- $\gamma : V_1 \times V \rightarrow V_2$ is a function definable in \mathcal{M} (where $V_1 \subseteq M^{k_1}$, $V \subseteq M$ and $V_2 \subseteq M^{k_2}$ are definable subsets).³

² $\sin_{|[0,2\pi]}$ and $\cos_{|[0,2\pi]}$ correspond to the sine and cosine functions restricted to interval $[0, 2\pi]$.

³ We use these notations in the rest of the paper.



■ **Figure 1** A dynamical system with three trajectories.

The function γ is called the *dynamics* of the system and (\mathcal{M}, γ) is said to be o-minimal when the theory of \mathcal{M} is itself o-minimal.

Classically, we see V as the time, V_1 as the input space, or set of parameters, $V_1 \times V$ as the space-time and V_2 as the output, or geometrical, space.

► **Definition 5.** For a dynamical system (\mathcal{M}, γ) , if we fix a point $x \in V_1$, the set $\Gamma_x = \{\gamma(x, t) \mid t \in V\} \subseteq V_2$ is called the *trajectory* determined by x .

We define a transition system associated with the dynamical system. This definition is an adaptation to our context of the classical *continuous transition system* in the case of hybrid systems (see [25] for example).

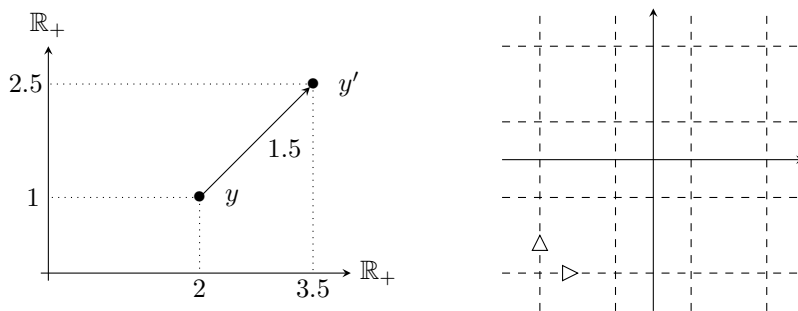
► **Definition 6.** Given (\mathcal{M}, γ) a dynamical system, the associated *transition system* $T_\gamma = (Q, \rightarrow)$ is defined by:

- its set of states $Q = V_2$;
- its transition relation \rightarrow , which is defined by: $y \rightarrow y'$ if $\exists x \in V_1, \exists t, t' \in V$ such that $t \leq t'$ and $\gamma(x, t) = y, \gamma(x, t') = y'$.

As usual, an execution is a sequence of consecutive transitions. Note that it is possible to switch between trajectories, as illustrated below.

► **Example 7.** The dynamical system depicted in Figure 1 is composed of three trajectories (with $\gamma(x_1, \cdot)$ in blue, $\gamma(x_2, \cdot)$ in green and $\gamma(x_3, \cdot)$ in red), with set of parameters $V_1 = \{x_1, x_2, x_3\}$ and $V = V_2 = \mathbb{R}$. Executions take place in \mathbb{R} , according to the trajectories. For instance: $y_1 \rightarrow y_2 \rightarrow y_3 \rightarrow y_4$ with $y_1 = 3 = \gamma(x_3, -1)$ and $y_2 = 2.5 = \gamma(x_3, 0)$ on the red curve, then switching to the green curve since $y_2 = \gamma(x_2, t_2)$ for some $t_2 < 0$, $y_3 = 0.5 = \gamma(x_2, 2)$, and finally jumping to the blue curve since $y_3 = \gamma(x_1, 2)$, leading to $y_4 = \gamma(x_1, 4)$.

The definition of *dynamical system* encompasses a lot of different behaviours, examples of which can be obtained with structures enriched by additional operations like addition, multiplication (or the exponential function).



■ **Figure 2** Dynamical systems: timed automata in dimension 2 (left) and example from [6] (right).

► **Example 8.** A classical one is the continuous dynamics of *timed automata* [1]: In this case, $\mathcal{M} = \langle \mathbb{R}, <, + \rangle$ and the dynamics $\gamma : \mathbb{R}_+^n \times [0, +\infty[\rightarrow \mathbb{R}_+^n$ is defined by $\gamma(x_1, \dots, x_n, t) = (x_1 + t, \dots, x_n + t)$. The standard graphical view, displayed in Figure 2 left, represents the dynamical system directly on the output space: $y \rightarrow y'$ with $y = (2, 1) = \gamma((2, 1), 0)$ and $y' = (3.5, 2.5) = \gamma((2, 1), 1.5)$.

► **Example 9.** Another example, borrowed from [6] and illustrated in Figure 2 right, features a dynamical system where each point of the plane has two possible behaviours: going right or going up. The dynamics $\gamma : \mathbb{R}^2 \times \{-1, +1\} \times \mathbb{R} \rightarrow \mathbb{R}^2$ is defined by:

$$\gamma(x_1, x_2, p, t) = \begin{cases} (x_1 + t, x_2) & \text{if } p = +1 \\ (x_1, x_2 + t) & \text{if } p = -1 \end{cases}$$

Then $y_1 \rightarrow y_2 \rightarrow y_3$ for the three points $y_1 = (0, 0)$, $y_2 = (0, 1)$ and $y_3 = (1, 1)$, since $y_1 = \gamma(0, 0, -1, 0)$, $y_2 = \gamma(0, 0, -1, 1) = \gamma(0, 1, 1, 0)$, and $y_3 = \gamma(0, 1, 1, 1)$.

In hybrid automata, such behaviours are combined with a finite set of discrete locations, each one having its own dynamics with respect to a common structure \mathcal{M} ; jumps between locations are constrained by guards and equipped with updates. As mentioned in the introduction, basic verification problems like reachability checking are undecidable in the general case, and solutions to recover decidability are often to impose strong reinitializations of trajectories at jumps (we will come back to that in subsection 2.4), which amounts to concentrating on the analysis of a single dynamical system.

2.3 Time-abstract bisimulation

Time-abstract bisimulation [18, 14, 2, 25] is a behavioural relation often used to obtain a quotient of the original transition system. When this quotient is finite, a large class of properties can be verified, notably reachability properties.

We associate with a dynamical system (\mathcal{M}, γ) a finite set G of guards, which are definable subsets of V_2 . For every $y \in V_2$, we define the set $G_y \stackrel{\text{def}}{=} \{g \in G \mid y \in g\}$ of guards that are “satisfied” by y , thus producing a finite partition of V_2 into subsets satisfying the same sets of guards.

► **Definition 10.** Consider a dynamical system (\mathcal{M}, γ) , a finite set G of definable guards and an integer $k \geq -1$. A *k-step time-abstract bisimulation* is an equivalence relation $R_k \subseteq V_2 \times V_2$ such that either (i) $k = -1$, or (ii) $k \geq 0$ and there exists a $(k - 1)$ -step time-abstract bisimulation R_{k-1} such that, if $(y_1, y_2) \in R_k$, then:

- (a) $G_{y_1} = G_{y_2}$;
- (b) if $y_1 \rightarrow y'_1$ then there exists y'_2 such that $y_2 \rightarrow y'_2$ and $(y'_1, y'_2) \in R_{k-1}$;
- (c) if $y_2 \rightarrow y'_2$ then there exists y'_1 such that $y_1 \rightarrow y'_1$ and $(y'_1, y'_2) \in R_{k-1}$.

We further say that an equivalence relation $R \subseteq V_2 \times V_2$ is a *time-abstract bisimulation* if R is a k -step time-abstract bisimulation for all $k \geq -1$. We also say that y_1 and y_2 are (k -step) *time-abstract bisimilar* whenever there is a (k -step) time-abstract bisimulation $R \subseteq V_2 \times V_2$ such that $(y_1, y_2) \in R$.

Note that, for every k , the class of k -step time-abstract bisimulations is closed under union, and therefore there is a largest k -step time-abstract bisimulation, which can be obtained as the union of all such relations. In particular, the relation R_{k-1} used in items (b) and (c) when defining R_k can be taken as the largest $(k-1)$ -step time-abstract bisimulation. Similarly, there is a largest time-abstract bisimulation.

2.4 Problem and existing results

We focus here on the construction of finite (time-abstract) bisimulation relations, which is a standard and powerful tool to prove decidability of classes of hybrid systems [18].

Existence of such relations is, for instance, the key property satisfied by timed automata [1], a well-established model for real-time systems. However, for hybrid systems with more complex dynamics, proving that there is a finite bisimulation might be difficult and is not possible in general. In several works willing to better understand rich continuous dynamics in a system, the idea has been to decouple the continuous and the discrete parts of the system by assuming (possibly non-deterministic) reinitializations of the trajectories when a jump between locations is performed, see e.g. [14, 24, 26, 23, 25]. This leads to only focus on bisimulation relations within a discrete location. In this work, we follow this idea, and therefore only focus on bisimulations generated by a single dynamical system.

A standard methodology for proving that there is a finite time-abstract bisimulation is to compute successive approximations of the bisimulation relation (see [18, 24, 26, 23, 25, 7, 5]), and show that the procedure terminates. In (almost) all the references mentioned below, this is the way the problem is attacked. While the methodology seems rather universal, it is amazing to see the variety of arguments which are used to show termination of the procedure. They range from analytical and geometrical arguments [24, 26, 25] to model theory arguments [23], algebraic and topological arguments [14] or, more recently, arguments based on word combinatorics [7, 5].

While the precise domains of applicability of the approaches might vary, in most mentioned related works (except [8, 7, 5]), time-determinism is assumed, in the sense that there is a single trajectory going through some point of the output space. In [8, 7, 5], several trajectories may intersect or self-intersect, but rather strong assumptions need to be made. For instance, in the *suffix-determinism* assumption, all trajectories starting from a given point of the output space visit the same pieces of the initial partition in a similar way; in the *loop-determinism* assumption, two trajectories cannot intersect each other, but a trajectory can intersect itself in finitely many points.

In our work, lots of self-intersecting and overlapping trajectories are possible, but we bound the number of trajectories one can reach by switching between them (we will formalize this later). For example, the dynamical system of Figure 1 does not satisfy any of the above assumptions, but typically fits our framework.

The generic symbolic approach of [5] (and in particular the 2-subword refinement procedure) is a semi-procedure for finding finite bisimulations of o-minimal dynamical systems: it finds a finite bisimulation relation if there is one, but cannot tell that there is no finite

bisimulation. But only the two above-mentioned assumptions (suffix-determinism and loop-determinism) guarantee termination of the computation. For instance, even though it does not satisfy any of the sufficient conditions above, the system of Example 9 has a finite bisimulation, which can be computed by the refinement procedure with respect to the single guard $y = (0, 0)$. Since there is no bound on the intersections of trajectories, this system does not belong to our class. On the other hand, both the present work and the approach of [5] encompass the original result [25].

Also, while the theory of o-minimality has been developed in any linearly ordered structure [30, 31, 21, 33], initial settings [25, 14] assume expansion of the reals. Here, similarly to [5], our results hold in the general setting.

In this paper we provide a method which is only based on geometrical properties of o-minimal systems. It does not assume the field of real numbers, nor dense or discrete structures. Furthermore, we are able to deal not only with (self-)crossing trajectories but also with partly stationary trajectories.

3 The graph construction

In this section, we fix an o-minimal dynamical system (\mathcal{M}, γ) and a finite set G of guards as defined above, and we build a graph representing the time-abstract behaviour of γ .

We define a relation \sim on V_1 , where $x \sim x'$ if and only if the trajectories Γ_x and $\Gamma_{x'}$ cross each other, i.e. if there exist $t, t' \in V$ such that $\gamma(x, t) = \gamma(x', t')$. We also set $V_1(x) \stackrel{\text{def}}{=} \{x' \in V_1 : x \sim x'\}$.

To build the graph we distinguish between points of V_2 with (at most) finitely many predecessors by γ on any trajectory and points of V_2 with infinitely many predecessors on some trajectory. We will show that those two sets are definable, and that they can be used to provide a nice finite decomposition of the state-space, fine enough to characterize the time-abstract bisimulation. After defining suitable notions of intervals, we independently provide a finite decomposition result and the construction of the graph itself.

3.1 Towards a decomposition

In what follows, we need to distinguish two kinds of intervals: singletons, i.e. intervals with one unique element, and intervals with at least two elements, which we call *large* intervals.

► **Definition 11.** An interval $I \subseteq V$ is called *x-static* if either (i) I is large and $|\gamma(x, I)| = 1$, or (ii) I is a singleton and there exist a parameter $x' \in V_1$ and a large interval $J \subseteq V$ such that $\gamma(x', J) = \gamma(x, I)$. We further say that an element t of V is *x-static* if t belongs to some *x-static* interval, and that a state $y \in V_2$ is *static* if there exists $x \in V_1$ and $t \in V$ such that t is *x-static* and $y = \gamma(x, t)$.

On the contrary, we say that an element t of V is *x-dynamic* if t is not *x-static*, and we say that an interval I is *x-dynamic* if every element of I is *x-dynamic*. We further say that I is *x-suitable* if (i) I is *x-dynamic*, (ii) the function $t \mapsto G_{\gamma(x, t)}$ is constant on I , and (iii) the function $\gamma(x, \cdot)$ is one-to-one on I .

This produces a classification of points in V_2 : static, if some trajectory stops at that position, or dynamic. It also induces a classification of timepoints and intervals along trajectories: a static point y of V_2 generates *x-static* timepoints on Γ_x , even though the trajectory Γ_x may not be responsible for making y static.

► **Example 12.** We illustrate the various notions on the example of Figure 1. Value $y = 0.5$ is static, because of x_1 and x_2 . In particular, interval $(1, 2)$ is x_1 -static and interval $(1.5, 3)$ is x_2 -static. Time $t = 4$ is x_3 -static because $\gamma(x_3, 4) = 0.5$ is static, even though $\gamma(x_3, \cdot)$ itself crosses $y = 0.5$ only in one point. And thus, interval $\{4\}$ is also x_3 -static but not large.

Note that $y = y_*$ is dynamic, since no trajectory of the dynamical system is constant on a large interval on which its value is y_* .

Assuming no guard in the system (or a single guard $y = 0.5$), the intervals $(-\infty, 1)$ and $(2, +\infty)$ are x_1 -suitable (and maximal for that condition). Similarly, the intervals $(-\infty, 1.5)$ and $(3, +\infty)$ are x_2 -suitable; the intervals $(-\infty, 4)$ and $(4, +\infty)$ are x_3 -suitable.

Then, since we want a finite representation of important points of the dynamical system, we need to get uniform (definable) descriptions of the above classification of points.

First, we gather all portions of trajectories corresponding to dynamic parts of the system. Note that such trajectories, while they visit the same state-space (in V_2), might follow different directions (hence the value $\varepsilon = \pm 1$ below).

► **Definition 13.** Consider two parameters $x, x' \in V_1$, one x -suitable interval $I \subseteq V$ and one x' -suitable interval $I' \subseteq V$. We say that the pairs (x, I) and (x', I') are *adapted* to each other if:

- (i) the sets $\{\gamma(x, t) \mid t \in I\}$ and $\{\gamma(x', t') \mid t' \in I'\}$ are equal to each other;
- (ii) there exists $\varepsilon = \pm 1$ such that: for all $t, u \in I$ with $t < u$, there exist $t', u' \in I'$ such that $\gamma(x, t) = \gamma(x', t')$, $\gamma(x, u) = \gamma(x', u')$, and $t' < u' \Leftrightarrow \varepsilon = 1$.

In general, we say that a family of pairs $(x_k, I_k)_{k \in K}$ is *strongly adapted* if:

- (iii) every two pairs (x_k, I_k) and (x_ℓ, I_ℓ) are adapted to each other;
- (iv) for all $k \in K$, $\{(x, t) \in V_1 \times M \mid \gamma(x, t) \in \gamma(x_k, I_k)\} = \bigcup_{\ell \in K} \{x_\ell\} \times I_\ell$.

Finally, we say that an interval I is *x -adaptable* if the pair (x, I) belongs to a strongly adapted family.

► **Example 14.** Going back to the previous example:

- the pairs $(x_1, (-\infty, 1))$ and $(x_2, (-\infty, 1.5))$ are adapted to each other (with $\varepsilon = +1$);
- the pairs $(x_1, (2, +\infty))$ and $(x_2, (-\infty, 1.5))$ are also adapted to each other (with $\varepsilon = -1$);
- the pairs $(x_2, (3, +\infty))$ and $(x_3, (4, +\infty))$ are adapted to each other (with $\varepsilon = +1$).

By extension, we get that:

- the pairs $(x_1, (-\infty, 1))$, $(x_1, (2, +\infty))$, $(x_2, (-\infty, 1.5))$ and $(x_3, (-\infty, 4))$ form a strongly adapted family;
- the interval $(2, +\infty)$ is x_1 -adaptable, due to the strongly adapted family above;
- the interval $(3, +\infty)$ is x_2 -adaptable, due to the family formed of $(x_2, (3, +\infty))$ and $(x_3, (4, +\infty))$;
- the singleton $\{t_*\}$ is both x_1 -adaptable and x_3 -adaptable, due to the family formed of $(x_1, \{-1\})$, $(x_2, \{t_o\})$, $(x_1, \{t_*\})$ and $(x_3, \{t_*\})$.

An interval I is said maximal x -static (resp. maximal x -adaptable), whenever it is x -static (resp. x -adaptable), and is contained in no larger x -static (resp. x -adaptable) interval.

It turns out that maximal x -static and x -adaptable intervals form a covering of the time domain V .

► **Lemma 15.** Consider a parameter $x \in V_1$ and a timepoint $t \in V$. There exists an interval $I \subseteq V$, which contains t , and such that I is a maximal x -static interval (if t is x -static) or a maximal x -adaptable interval (if t is x -dynamic).

Proof. If t is x -static, then the singleton $\{t\}$ is x -static. If t is x -dynamic, then the family of pairs $(x', \{t'\})$ such that $\gamma(x', t') = \gamma(x, t)$ is strongly adapted, and therefore the singleton $\{t\}$ is x -adaptable. Moreover, both the class of x -static intervals and the class of x -adaptable intervals are closed under increasing union: this is clear for x -static intervals, and can be argued as follows for x -adaptable intervals.

Let $(I^\alpha)_\alpha$ be an increasing family of x -adaptable intervals. For every α , let $\mathcal{F}^\alpha = (x_k^\alpha, I_k^\alpha)_{k \in K^\alpha}$ be a corresponding strongly adapted family. There is an obvious one-to-one correspondence between elements of \mathcal{F}^α and elements of $\mathcal{F}^{\alpha'}$ for any pair of indices (α, α') , hence one can rewrite the family \mathcal{F}^α uniformly as $(x_k, I_k^\alpha)_{k \in K}$. One can therefore take $\mathcal{F} = (x_k, \bigcup_\alpha I_k^\alpha)_{k \in K}$ as a strongly adapted family for $(x, \bigcup_\alpha I^\alpha)$. The result follows. ◀

3.2 Finite decomposition result

Our goal here is to prove the following decomposition, which refines Lemma 15.

► **Proposition 16.** *Consider a parameter $x \in V_1$ such that $V_1(x)$ is finite. Then, the set V is a finite, disjoint and definable union of intervals I_1, \dots, I_k such that every interval I_j is either*

1. *a maximal x -static interval, or*
2. *a maximal x -adaptable interval.*

We first focus on static (geometrical, i.e. in V_2) points and show that there can only be finitely many such points along a trajectory.

► **Lemma 17.** *There exists an integer \mathbf{K} such that, for every parameter $x \in V_1$, there exist at most \mathbf{K} large maximal x -static intervals.*

Proof. We first observe that, if I_1 and I_2 are maximal large x -static intervals, with $I_1 \neq I_2$, then $I_1 \cap I_2 = \emptyset$. Otherwise, the union $I_1 \cup I_2$ would also be x -static, contradicting the maximality of I_1 and I_2 . Henceforth, we denote by \prec the linear order on maximal large x -static intervals, defined by

$$I_1 \prec I_2 \text{ if and only if } \forall t \in I_1, \forall t' \in I_2, t < t'.$$

If $I_1 \prec I_2$, and if I_1 and I_2 have respective lower bounds ℓ_1 and ℓ_2 , then $t \leq \ell_2$ for all $t \in I_1$ and therefore $\ell_1 < \ell_2$ (since I_1 is large). Consequently, if $\ell_2 \in I_1$, then I_1 must have a maximal element, and $\ell_2 = \max(I_1)$.

Now, let $\mathcal{L}(x)$ be the set of lower bounds of maximal x -static intervals. Observe that $\mathcal{L}(x)$ is definable, and therefore by Theorem 3, there exists an integer \mathbf{K}_1 such that, for all $x \in V_1$, $\mathcal{L}(x)$ is a disjoint union of at most \mathbf{K}_1 intervals. We claim that each of these intervals has (strictly) less than three elements.

Assume on the contrary that there exists a sub-interval J of $\mathcal{L}(x)$ containing three elements $\ell_1 < \ell_2 < \ell_3$. For all $t \in J$, we denote by $I(t)$ the maximal large x -static interval with lower bound t . Since $I(\ell_1)$ is large, it contains some element t such that $\ell_1 < t$. Up to replacing both t and ℓ_2 by $\min\{t, \ell_2\}$, we assume that $t = \ell_2$. It follows, as noted above, that $\ell_2 = \max(I(\ell_1))$. Since $I(\ell_2)$ is large too, consider some element u of $I(\ell_2)$ that is not maximal in $I(\ell_2)$. Since $\ell_1 \in I(\ell_1)$ and $I(\ell_1) \prec I(\ell_2)$, we know that $\ell_2 < u$. Up to replacing both u and ℓ_3 by $\min\{u, \ell_3\}$, we also assume that $u = \ell_3$, hence that $\ell_3 \in I(\ell_2)$. However, since $\ell_2 < \ell_3$, our initial remark proves that $u = \ell_3$ must be the maximal element of $I(\ell_2)$, contradicting the definition of u . This proves our claim.

The set $\mathcal{L}(x)$ is therefore of cardinality at most $2\mathbf{K}_1$. Observing that at most one maximal large x -static interval has no lower bound proves that there exist at most \mathbf{K} large maximal x -static intervals, where $\mathbf{K} = 2\mathbf{K}_1 + 1$. ◀

► **Lemma 18.** *There exists an integer \mathbf{L} such that, for every parameter $x \in V_1$, the set of x -static elements of V is a disjoint union of at most $\mathbf{L} |V_1(x)|$ maximal x -static intervals.*

Proof. Fix $x \in V_1$. Let \mathcal{S} denote the set of static elements of $\gamma(x, V)$. With each element y of \mathcal{S} we can associate a pair (x', I) , where $x' \in V_1(x)$ and I is a maximal large x' -static interval such that $\gamma(x', I) = \{y\}$. This association is one-to-one, and therefore $|\mathcal{S}| \leq \mathbf{K} |V_1(x)|$.

Moreover, there exists an integer \mathbf{L}_1 such that, for every $y \in V_2$, the definable set $\{t \in V \mid \gamma(x, t) = y\}$ is a finite union of at most \mathbf{L}_1 intervals (Theorem 3). Assuming, without loss of generality, that these intervals are pairwise disjoint, proves Lemma 18 for $\mathbf{L} = \mathbf{K} \mathbf{L}_1$. ◀

We now turn to the case of dynamic elements. We start with the following combinatorial lemma, whose proof is immediate by induction on $k + \ell$.

► **Lemma 19.** *Let $\mathcal{I} = (I_1, \dots, I_k)$ and $\mathcal{J} = (J_1, \dots, J_\ell)$ be two partitions of V into sub-intervals. There exists a partition $\mathcal{K} = (K_1, \dots, K_m)$ of V into sub-intervals that refines both \mathcal{I} and \mathcal{J} , and such that $m + 1 \leq k + \ell$.*

► **Lemma 20.** *There exists an integer \mathbf{M} such that, for every parameter $x \in V_1$, every maximal x -dynamic interval of V is a disjoint union of at most $\mathbf{M}(1 + |V_1(x)|)$ maximal x -adaptable intervals.*

Proof. First, recall that there exists an integer \mathbf{L}_1 such that, for all $x \in V_1$ and all $y \in V_2$, the definable set $\{t \in V \mid \gamma(x', t) = y\}$ is a disjoint union of at most \mathbf{L}_1 intervals. If y is not static, then these intervals must be singletons, and therefore $|\{t \in V \mid \gamma(x', t) = y\}| \leq \mathbf{L}_1$.

Now, for all $t \in V$ and $x, x' \in V_1$, we denote by $f_1(x, x', t) < \dots < f_{\mathbf{L}_1}(x, x', t)$ the elements of the set $\{t' \in V \mid \gamma(x, t) = \gamma(x', t')\}$, where $f_i(x, x', t)$ is undefined if $|\{t' \in V \mid \gamma(x, t) = \gamma(x', t')\}|$ is either smaller than i or greater than \mathbf{L}_1 (in the latter case, $\gamma(x, t)$ must be static). Observe that every function f_i is definable. Consequently, there exists an integer \mathbf{M}_1 such that, for all $x \in V_1$ and $x' \in V_1(x)$, there exists a partition $\mathcal{P}_i(x, x')$ of V into at most \mathbf{M}_1 intervals on which the function $t \mapsto f_i(x, x', t)$ is either undefined, constant, or continuous and strictly monotonic (Theorems 2 and 3).

Similarly, since the function $(x, t) \mapsto G_{\gamma(x, t)}$ is definable, there exists an integer \mathbf{M}_2 such that, for all $x \in V_1$, there exists a partition $\mathcal{P}'(x)$ of V in at most \mathbf{M}_2 intervals on which $t \mapsto G_{\gamma(x, t)}$ is constant.

Now, consider some $x \in V_1$. By Lemma 19, there exists a partition \mathcal{P} of V , which refines $\mathcal{P}'(x)$ and every partition $\mathcal{P}_i(x, x')$, for $i \leq \mathbf{L}_1$ and $x' \in V_1(x)$, and which contains at most $\mathbf{M}_1 \mathbf{L}_1 |V_1(x)| + \mathbf{M}_2$ intervals. By construction, every interval of the partition \mathcal{P} is either x -adaptable or x -static, and by choosing \mathcal{P} to contain as few intervals as possible, these intervals are guaranteed to be maximal x -adaptable intervals. Lemma 20 follows, by choosing $\mathbf{M} = \max\{\mathbf{M}_1 \mathbf{L}_1, \mathbf{M}_2\}$. ◀

Since maximal x -adaptable intervals and maximal x -static intervals are definable, we derive from Lemmas 18 and 20 the targeted Proposition 16.

3.3 Construction of the graph

► **Definition 21.** We call *bisimulation graph* for the o-minimal dynamical system (\mathcal{M}, γ) and the set of definable guards G the (possibly infinite) labeled graph with ε -transitions $\mathcal{G} = (N, E, E_\varepsilon, L)$ defined as follows:

- the set of nodes is

$$N = \{(x, I) : x \in V_1, I \text{ is a maximal } x\text{-static or } x\text{-adaptable interval}\};$$

- the set of edges is

$$E = \{((x, I), (x, J)) \in N \times N : \exists t \in I, \exists t' \in J, t \leq t'\};$$

- the set of ε -transitions is

$$E_\varepsilon = \{((x, I), (x', I')) \in N \times N : \exists t \in I, \exists t' \in I', \gamma(x, t) = \gamma(x', t')\};$$

- the labeling function is $L : (x, I) \mapsto \{g : \exists t \in I, g \in G_{\gamma(x, t)}\}$.

Next, we write \rightarrow (resp. \rightarrow_ε) the transition relation defined by E (resp. E_ε), and we denote by \rightsquigarrow the relation defined by: $n_1 \rightsquigarrow n_4$ if there exist nodes n_2 and n_3 such that $n_1 \rightarrow_\varepsilon n_2 \rightarrow n_3 \rightarrow_\varepsilon n_4$.

► **Definition 22.** Consider an integer $k \geq -1$. A *k-step ε -bisimulation* is an equivalence relation $\mathfrak{R}_k \subseteq N \times N$ such that either (i) $k = -1$, or (ii) $k \geq 0$ and there exists a $(k-1)$ -step ε -bisimulation \mathfrak{R}_{k-1} such that, if $n_1 \mathfrak{R}_k n_2$, then:

(a) $L(n_1) = L(n_2)$;

(b) if $n_1 \rightsquigarrow n'_1$ then there exists n'_2 such that $n_2 \rightsquigarrow n'_2$ and $n'_1 \mathfrak{R}_{k-1} n'_2$;

(c) if $n_2 \rightsquigarrow n'_2$ then there exists n'_1 such that $n_1 \rightsquigarrow n'_1$ and $n'_1 \mathfrak{R}_{k-1} n'_2$.

We further say that an equivalence relation $\mathfrak{R} \subseteq N \times N$ is a *ε -bisimulation* if \mathfrak{R} is a k -step ε -bisimulation for all $k \geq -1$. We also say that two nodes n_1 and n_2 are *(k-step) ε -bisimilar* whenever there is a $(k$ -step) ε -bisimulation $\mathfrak{R} \subseteq N \times N$ such that $n_1 \mathfrak{R} n_2$.

Like time-abstract bisimulation, the class of $(k$ -step) ε -bisimulations is closed under union, hence there is a largest $(k$ -step) ε -bisimulation, which can be obtained as the union of all such relations. In particular, the relation \mathfrak{R}_{k-1} used in items (b) and (c) when defining \mathfrak{R}_k can be taken as the largest $(k-1)$ -step ε -bisimulation.

► **Lemma 23.** Let $n = (x, I)$ and $n' = (x', I')$ be nodes of the bisimulation graph \mathcal{G} . The following statements are equivalent: (i) $n \rightarrow_\varepsilon n'$, (ii) $\gamma(x, I) \cap \gamma(x', I') \neq \emptyset$, and (iii) $\gamma(x, I) = \gamma(x', I')$.

Proof. The equivalence between (i) and (ii) follows directly from the definition of the set E_ε of ε -transitions, and the implication (iii) \Rightarrow (ii) is obvious.

It remains to prove (iii), under the assumption that (ii) holds. If I is x -static, then $\gamma(x, I)$ is a singleton, hence I' contains an x' -static element, and therefore I' is not x' -suitable. This proves that I' is x' -static, hence that $\gamma(x', I')$ is a singleton too, and (iii) follows.

If I is maximal x -adaptable, then I' cannot be x' -static, hence I' is maximal x' -adaptable too. Let I'' be an interval such that (x, I) and (x', I'') are adapted, with $I' \cap I'' \neq \emptyset$. Since maximal x' -adaptable intervals are disjoint, it follows that $I'' \subseteq I'$, whence $\gamma(x, I) \subseteq \gamma(x', I')$. Similarly, we have $\gamma(x', I') \subseteq \gamma(x, I)$, which completes the proof. ◀

► **Lemma 24.** *Let $n = (x, I)$ and $n' = (x', I')$ be nodes of the bisimulation graph \mathcal{G} . The following statements are equivalent: (i) $n \rightsquigarrow n'$, (ii) $\exists y \in \gamma(x, I), \exists y' \in \gamma(x', I')$ s.t. $y \rightarrow y'$, and (iii) $\forall y \in \gamma(x, I), \exists y' \in \gamma(x', I')$ s.t. $y \rightarrow y'$.*

Proof. We first prove that (i) \Rightarrow (iii). Assume that $n \rightsquigarrow n'$, and let $n_1 = (x_1, I_1), n_2 = (x_2, I_2)$ be nodes such that $n \rightarrow_\varepsilon n_1 \rightarrow n_2 \rightarrow_\varepsilon n'$. Let also $y \in \gamma(x, I)$. By Lemma 23, there exists $t \in I_1$ such that $y = \gamma(x_1, t)$. Let us prove that there exists $t' \in I_2$ such that $t \leq t'$. Indeed, if $I_1 = I_2$, we may choose $t' = t$. Otherwise, recall that I_1 and I_2 , as maximal x_1 -static or x_1 -adaptable intervals, must be disjoint, and that there exist $t_1 \in I_1$ and $t_2 \in I_2$ such that $t_1 \leq t_2$; this proves in fact that $t_1 < t_2$ for all $t_1 \in I_1$ and $t_1 \in I_2$, and therefore that every $t' \in I_2$ is greater than t . Finally, let $y' = \gamma(x_2, t')$. Since $x_1 = x_2$ and $t \leq t'$, we know that $y \rightarrow y'$, and since $n_2 \rightarrow_\varepsilon n'$, Lemma 23 proves that $y' \in \gamma(x', I')$, which proves (iii).

Second, observe that the implication (iii) \Rightarrow (ii) is immediate. It remains to prove that (ii) \Rightarrow (i). Assume that (ii) holds. Let $x_1 \in V_1$ be a parameter, and $t_1 \leq t_2$ be elements of V such that $y = \gamma(x_1, t_1)$ and $y' = \gamma(x_1, t_2)$. Let I_1 and I_2 be the maximal x_1 -static or x_1 -adaptable intervals to which belong t_1 and t_2 , and let $n_1 = (x_1, t_1)$ and $n_2 = (x_1, t_2)$. By construction, and using Lemma 23, we have $n \rightarrow_\varepsilon n_1 \rightarrow n_2 \rightarrow_\varepsilon n'$, which proves (i). ◀

► **Theorem 25.** *For all integers $k \geq -1$, two elements y_1 and y_2 in V_2 are (k -step) time-abstract bisimilar if and only if there exist (k -step) ε -bisimilar nodes $n_1 = (x_1, I_1)$ and $n_2 = (x_2, I_2)$ of the bisimulation graph \mathcal{G} such that $y_i \in \gamma(x_i, I_i)$.*

Proof. In the following, we conveniently write $\gamma(n)$ instead of $\gamma(x, I)$ when n is the node (x, I) .

For every $k \geq -1$, define R_k as the largest k -step time-abstract bisimulation over V_2 . We define the relation \mathfrak{R}_k over N as follows:

$$n_1 \mathfrak{R}_k n_2 \text{ iff } \exists y_i \in \gamma(n_i) \text{ such that } y_1 R_k y_2.$$

Let us prove, by induction on k , that \mathfrak{R}_k is a k -step ε -bisimulation relation. The case $k = -1$ is immediate, hence we assume that $k \geq 0$ and that \mathfrak{R}_{k-1} is a $(k-1)$ -step time-abstract bisimulation.

Let $n_1 = (x_1, I_1)$ and $n_2 = (x_2, I_2)$ be two nodes such that $n_1 \mathfrak{R}_k n_2$, and let $y_1 \in \gamma(n_1)$ and $y_2 \in \gamma(n_2)$ such that $y_1 R_k y_2$. First, since I_1 is either x_1 -static or x_1 -suitable, we know that the function $t \mapsto G_{\gamma(x_1, t)}$ is constant on I_1 . Similarly, the function $t \mapsto G_{\gamma(x_2, t)}$ is constant on I_2 and therefore $L(n_1) = G_y = L(n_2)$.

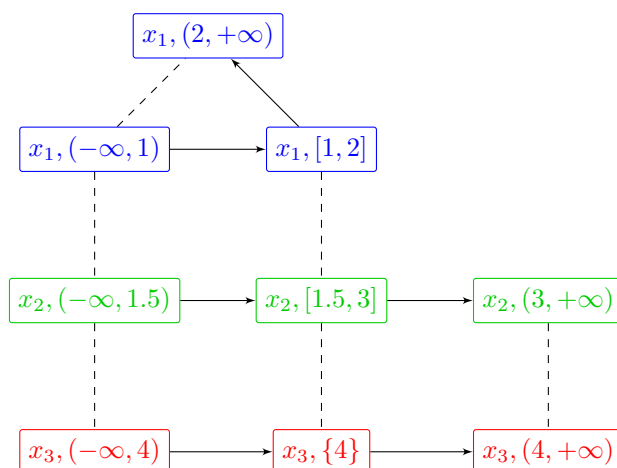
Then, let $n'_1 = (x'_1, I'_1)$ be a node such that $n_1 \rightsquigarrow n'_1$. By Lemma 24, there exists $y'_1 \in \gamma(n'_1)$ such that $y_1 \rightarrow y'_1$. Since $y_1 R_k y_2$, there also exists y'_2 such that $y_2 \rightarrow y'_2$ and $y'_1 R_{k-1} y'_2$. Let $n'_2 = (x'_2, I'_2)$ be a node such that $y'_2 \in \gamma(n'_2)$. By construction, we have $y'_1 \mathfrak{R}_{k-1} y'_2$. Since n_1 and n_2 play symmetric roles, \mathfrak{R}_k is a k -step ε -bisimulation relation.

Likewise, if R is the largest time-abstract bisimulation over V_2 , the relation \mathfrak{R} over N defined by $n_1 \mathfrak{R} n_2$ iff $\exists y_i \in \gamma(n_i)$ such that $y_1 R y_2$ is an ε -bisimulation relation.

Consequently, if y_1 and y_2 are (k -step) time-abstract bisimilar, constructing the relation (\mathfrak{R}_k or) \mathfrak{R} as above proves that there exist (k -step) ε -bisimilar nodes n_1 and n_2 of the bisimulation graph \mathcal{G} such that $y_i \in \gamma(n_i)$.

Conversely, for every $k \geq -1$, define \mathfrak{R}_k as the largest k -step ε -bisimulation over N . We define the relation R_k over V_2 as follows:

$$y_1 R_k y_2 \text{ iff } \exists n_i \in N \text{ such that } y_i \in \gamma(n_i) \text{ and } n_1 \mathfrak{R}_k n_2.$$



■ **Figure 3** The bisimulation graph of the previous example (\rightarrow_ε is obtained by reflexive and transitive closure of dashed lines; \rightarrow is represented by normal edges).

Let us prove, by induction on k , that R_k is a k -step ε -bisimulation relation. The case $k = -1$ is immediate, hence we assume that $k \geq 0$ and that R_{k-1} is a $(k-1)$ -step time-abstract bisimulation.

Consider two states $y_1, y_2 \in V_2$ such that $y_1 R_k y_2$, and let $n_1 = (x_1, I_1)$ and $n_2 = (x_2, I_2)$ be two nodes such that $y_i \in \gamma(n_i)$ and $n_1 \mathfrak{R}_k n_2$. Once again, the function $t \mapsto G_{\gamma(x_1, t)}$ is constant on I_1 , and $t \mapsto G_{\gamma(x_2, t)}$ is constant on I_2 , hence $G_{y_1} = L(n_1) = L(n_2) = G_{y_2}$.

Then, let y'_1 be a state such that $y_1 \rightarrow y'_1$, and let $n'_1 = (x'_1, I'_1)$ be a node such that $y'_1 \in \gamma(n'_1)$. Lemma 24 proves that $n_1 \rightsquigarrow n'_1$, and since \mathfrak{R}_k is a k -step ε -bisimulation relation there exists a node $n'_2 = (x'_2, I'_2)$ such that $n_2 \rightsquigarrow n'_2$ and $n'_1 \mathfrak{R}_{k-1} n'_2$. Lemma 24 proves that $y_2 \rightarrow y'_2$ for some $y'_2 \in \gamma(n'_2)$, and we have $y'_1 R_{k-1} y'_2$ by construction. Since y_1 and y_2 play symmetric roles, \mathfrak{R}_k is a k -step ε -bisimulation relation.

Likewise, if \mathfrak{R} is the largest ε -bisimulation over N , the relation R over V_2 defined by $y_1 R y_2$ iff $\exists n_i \in N$ such that $y_i \in \gamma(n_i)$ and $n_1 \mathfrak{R} n_2$ is a time-abstract bisimulation relation. In particular, if \mathfrak{R} is an ε -bisimulation relation, then R is a time-abstract bisimulation relation.

Consequently, if n_1 and n_2 are $(k$ -step) ε -bisimilar, constructing the relation (R_k or) R as above proves that, for all states $y_i \in \gamma(n_i)$, y_1 and y_2 are $(k$ -step) time-abstract bisimilar, which completes the proof. ◀

► **Example 26.** The bisimulation graph for the dynamical system of Figure 1 is depicted on Figure 3. We infer that:

- all points of the interval $(-\infty, y_3) = \gamma(x_2, (3, +\infty)) = \gamma(x_3, (4, +\infty))$ are time-abstract bisimilar;
- the singleton $\{y_3\} = \gamma(x_1, [1, 2]) = \gamma(x_2, [1.5, 3]) = \gamma(x_3, \{4\})$ forms a class of the time-abstract bisimulation;
- all points of the interval $(y_3, +\infty) = \gamma(x_1, (-\infty, 1)) = \gamma(x_1, (2, +\infty)) = \gamma(x_2, (-\infty, 1.5)) = \gamma(x_3, (-\infty, 4))$ are time-abstract bisimilar.

4 Definability and decidability

In this section, we discuss definability and decidability issues.

We say that a theory $\mathcal{M} = \langle M, <, \dots \rangle$ is *decidable* whenever for every first-order formula φ , for every $t \in M$, one can decide whether $t \models \varphi$ holds.

So far we have not assumed any decidability of the structures, and, indeed, not all o-minimal structures are decidable. For instance, it is not known whether the o-minimal structure $\langle \mathbb{R}, <, 0, 1, +, \cdot, \exp \rangle$ is decidable [27, 29]. Alternatively, if ω is a non-computable real number, such as Chaitin's constant [11], then the structure $\langle \mathbb{R}, <, 0, 1, \omega, + \rangle$ is o-minimal but not decidable.

In this section, we consider the relation \sim^* , which is the (reflexive and) transitive closure of \sim , with $V_1^*(x) \stackrel{\text{def}}{=} \{x' \in V_1 : x \sim^* x'\}$. We introduce the following assumption, called *Finite Crossing*: every equivalence class of the relation \sim^* (i.e. every set $V_1^*(x)$) is finite. The stronger condition obtained when there is a uniform bound on the size of equivalence classes is called *Uniform Crossing*.

► **Theorem 27.** *Let (\mathcal{M}, γ) be an o-minimal dynamical system. Under the Uniform Crossing assumption, the relation of time-abstract bisimulation is definable, and it contains finitely many equivalence classes.*

Proof. Let y_1, y_2 be elements of V_2 and let $x_1, x_2 \in V_1$ be parameters such that $y_i \in \Gamma_{x_i}$. Let also \mathbf{P} be a positive integer such that $|V_1^*(x)| \leq \mathbf{P}$ for all $x \in V_1$. Consider the sub-graph \mathcal{G}' of the bisimulation graph \mathcal{G} that consists in those nodes (x', I) with $x' \in V_1^*(x_1) \cup V_1^*(x_2)$. This sub-graph is finite, and Lemmas 18 and 20 prove that it contains at most $k(\mathbf{L} + \mathbf{M} + \mathbf{M}k)$ nodes, where $k = |V_1^*(x_1) \cup V_1^*(x_2)|$. Since $k \leq 2\mathbf{P}$, it follows that \mathcal{G}' contains at most $\mathbf{N} = 2\mathbf{P}(\mathbf{L} + \mathbf{M} + 2\mathbf{M}\mathbf{P})$ nodes.

It is well-known that, on \mathcal{G}' , the relations of ε -bisimulation and of \mathbf{N} -step ε -bisimulation are equal to each other. Hence, it follows from Theorem 25 that y_1 and y_2 are time-abstract bisimilar if and only if they are \mathbf{N} -step time-abstract bisimilar. In particular, the latter relation has finitely many equivalence classes, and is definable, which proves Theorem 27. ◀

► **Theorem 28.** *Let (\mathcal{M}, γ) be a decidable o-minimal dynamical system. Under the Finite Crossing assumption, the relation of time-abstract bisimulation is decidable: given $y_1, y_2 \in V_2$, one can decide whether y_1 and y_2 are time-abstract bisimilar.*

Proof. For all $k \geq 0$ and $x \in V_1$, let $\mathbf{V}(k, x) = \{x_k \in V_1 : \exists x_1, \dots, x_k \in V_1 \text{ s.t. } x \sim x_1, x_1 \sim x_2, \dots, x_{k-1} \sim x_k\}$, where we recall that the relation \sim is defined by: $x \sim x'$ iff $\Gamma_x \cap \Gamma_{x'} \neq \emptyset$. By construction, the set $\mathbf{V}(k, x)$ is definable and is a subset of $V_1^*(x)$. Moreover, since $V_1^*(x)$ is finite, there exists a minimal integer $k \geq 0$ such that $\mathbf{V}(k, x) = \mathbf{V}(k+1, x)$, and we have $V_1^*(x) = \mathbf{V}(k, x)$. Since the equality of definable sets is decidable, the set $V_1^*(x)$ is therefore computable for every parameter $x \in V_1$.

Now, let y_1, y_2 be elements of V_2 and let $x_1, x_2 \in V_1$ be parameters such that $y_i \in \Gamma_{x_i}$. We just showed how to compute the set $V_1' = V_1^*(x_1) \cup V_1^*(x_2)$. Then, let \mathcal{R} and \mathcal{R}' be the respective time-abstract bisimulation relations in (\mathcal{M}, γ) and of (\mathcal{M}, γ') , where γ' is the restriction of γ to the set $V_1' \times V$. Since \mathcal{R}' coincides with the restriction of \mathcal{R} to $\{y \in V_2 : \exists x \in V_1', y \in \Gamma_x\}$, it remains to compute the relation \mathcal{R}' .

Since V_1' is finite, we may apply Theorem 27 to (\mathcal{M}, γ') . We thereby prove that \mathcal{R}' has finitely many equivalence classes, and therefore is equal to the \mathbf{N} -step time-abstract bisimulation in (\mathcal{M}, γ') , for some integer \mathbf{N} . Consequently, the standard partition refinement procedure (see e.g. [7, p. 6]) will terminate, since there are finitely many classes, and we

will be able to detect termination, since the theory is decidable. The partition refinement procedure is therefore an effective algorithm which allows to compute the time-abstract bisimulation \mathcal{R}' , which completes the proof. ◀

Remark that all results still hold if we replace the conditions on the sets $V_1^*(x)$, $x \in V_1$ by a finer semantical definition:

$$V_1^*(x) = \{x' \in V_1 : \exists y_1, \dots, y_k \in V_2 \text{ s.t. } y_1 \in \Gamma_x, y_k \in \Gamma_{x'} \text{ and } y_1 \rightarrow \dots \rightarrow y_k\}.$$

Notice, however, that the assumption on the size of $V_1^*(x)$ could not be relaxed, due to the undecidability result of [4, Theorem 3.1].

Recovering the main result of [25]

The use of restricted dynamical systems also allows us to encompass the main result of [25].

► **Theorem 29.** *Let $\mathcal{M} = \langle \mathbb{R}, <, \dots \rangle$ be an expansion of the ordered set of the reals with an o-minimal theory, and let $V = \mathbb{R}$ and $V_1 = V_2 = \mathbb{R}^n$ for some integer $n \geq 1$. Assume that there exists a smooth, complete vector field F over \mathbb{R}^n such that the dynamics (called flow in [25]) $\gamma : (x, t) \rightarrow \gamma(x, t)$, which is defined by: $\gamma(x, 0) = x$ and $\frac{d}{dt}\gamma(x, t) = F(\gamma(x, t))$, is definable in \mathcal{M} . Then, the relation of time-abstract bisimulation is definable, and it contains finitely many equivalence classes.*

Proof. By construction, if two trajectories Γ_x and $\Gamma_{x'}$ have a non-empty intersection, then there exists a real number t such that $x' = \gamma(x, t)$, and we have $\gamma(x', u) = \gamma(x, t + u)$ for all $u \in \mathbb{R}$, so that the trajectories Γ_x and $\Gamma_{x'}$ are equal to each other. Hence, the relation \sim is an equivalence relation.

Then, due to [28, Corollary 3.3.28], there exists a definable set V_1' such that every equivalence class of \sim contains a unique element in V_1' . Consider the restricted dynamical system (\mathcal{M}, γ') , where γ' is the restriction of γ to the set $V_1' \times V$. This restricted dynamical system satisfies the hypothesis of Theorem 27, and therefore there exists an integer $\mathbf{N} \geq 0$ such that the time-abstract bisimulation relation and the \mathbf{N} -step time-abstract bisimulation relation in (\mathcal{M}, γ') are equal to each other. Since the transition systems associated with (\mathcal{M}, γ) and (\mathcal{M}, γ') are equal to each other, the result follows. ◀

5 Conclusion

In this paper, we have proposed a new approach for the analysis of o-minimal dynamical systems. Our approach allows us to treat trajectories with overlapping portions, and with possibly rich intersections. There is however a restriction, which is that trajectory switches should remain within a finite family of trajectories, once the initial trajectory has been chosen. It is important to notice that, as mentioned in the end of Section 4, it would not be possible to arbitrarily relax that assumption, since the reachability problem is undecidable for dynamical systems allowing arbitrarily many switches, as proved in [4, Theorem 3.1].

Adding the standard decoupling hypothesis, where jumps between locations reinitialize trajectories, we obtain a decidable class of hybrid systems.

Our future work will consist in trying to adapt the idea of interrupt timed automata of [3], where no reinitialization is assumed, to systems with richer (o-minimal) dynamics.

References

- 1 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- 2 Rajeev Alur, Thomas A. Henzinger, Gerardo Lafferriere, and George J. Pappas. Discrete abstractions of hybrid systems. *Proc. of the IEEE*, 88:971–984, 2000.
- 3 Béatrice Bérard, Serge Haddad, Claudine Picaronny, Mohab Safey El Din, and Mathieu Sassolas. Polynomial Interrupt Timed Automata. In *Proceedings of the 9th International Workshop on Reachability problems (RP'15)*, volume 9328 of *LNCS*, pages 20–32. Springer, 2015.
- 4 Thomas Brihaye. A note on the undecidability of the reachability problem for o-minimal dynamical systems. *Math. Log. Q.*, 52(2):165–170, 2006. doi:10.1002/malq.200510024.
- 5 Thomas Brihaye. *Verification and Control of O-Minimal Hybrid Systems and Weighted Timed Automata*. PhD thesis, Université de Mons-Hainaut, Belgium, 2006.
- 6 Thomas Brihaye. Words and bisimulation of dynamical systems. *Discrete Mathematics and Theoretical Computer Science*, 9(2):11–31, 2007.
- 7 Thomas Brihaye and Christian Michaux. On the expressiveness and decidability of o-minimal hybrid systems. *Journal of Complexity*, 21(4):447–478, 2005.
- 8 Thomas Brihaye, Christian Michaux, Cédric Rivière, and Christophe Troestler. On o-minimal hybrid systems. In *Proc. 7th International Workshop on Hybrid Systems: Computation and Control (HSCC'04)*, volume 2993 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2004.
- 9 Alberto Casagrande, Pietro Corvaja, Carla Piazza, and Bud Mishra. Decidable compositions of o-minimal automata. In Sung Deok Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan, editors, *Proc. of 6th International Symposium on Automated Technology for Verification and Analysis, ATVA 2008*, volume 5311 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2008. doi:10.1007/978-3-540-88387-6_25.
- 10 Alberto Casagrande, Carla Piazza, Alberto Policriti, and Bud Mishra. Inclusion dynamics hybrid automata. *Inf. Comput.*, 206(12):1394–1424, 2008. doi:10.1016/j.ic.2008.09.001.
- 11 Gregory J Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM (JACM)*, 22(3):329–340, 1975.
- 12 Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium, RTSS 2012*, pages 183–192. IEEE Computer Society, 2012. doi:10.1109/RTSS.2012.70.
- 13 G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages 2nd GI Conference*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer, 1975.
- 14 Jennifer M. Davoren. Topologies, continuity and bisimulations. *Informatique Théorique et Applications*, 33(4-5):357–382, 1999.
- 15 Raffaella Gentilini. Reachability problems on extended o-minimal hybrid automata. In *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 2005.
- 16 R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid systems*, volume 736 of *LNCS*. Springer, 1993.
- 17 Emmanuel Hainry. Reachability in linear dynamical systems. In *Logics and Theory of Algorithms, Proc. 4th Conference on Computability in Europe (CiE'08)*, volume 5028 of *Lecture Notes in Computer Science*, pages 241–250. Springer, 2008.

- 18 Thomas A. Henzinger. Hybrid automata with finite bisimulations. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP'95)*, volume 944 of *Lecture Notes in Computer Science*, pages 324–335. Springer, 1995.
- 19 Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.
- 20 Wilfrid Hodges. *A Shorter Model Theory*. Cambridge University Press, 1997.
- 21 Julia F. Knight, Anand Pillay, and Charles Steinhorn. Definable sets in ordered structures. II. *Transactions of the American Mathematical Society*, 295(2):593–605, 1986.
- 22 Margarita V. Korovina and Nicolai Vorobjov. Upper and lower bounds on sizes of finite bisimulations of Pfaffian hybrid systems. In *CiE*, volume 3988 of *Lecture Notes in Computer Science*, pages 267–276. Springer, 2006.
- 23 Gerardo Lafferriere, George J. Pappas, and Shankar Sastry. Hybrid systems with finite bisimulations. In *Proc. Hybrid Systems V: Verification and Control (1997)*, volume 1567 of *Lecture Notes in Computer Science*, pages 186–203. Springer, 1997.
- 24 Gerardo Lafferriere, George J. Pappas, and Shankar Sastry. Subanalytic stratifications and bisimulations. In *Proc. 1st International Workshop on Hybrid Systems: Computation and Control (HSCC'98)*, volume 1386 of *Lecture Notes in Computer Science*, pages 205–220. Springer, 1998.
- 25 Gerardo Lafferriere, George J. Pappas, and Shankar Sastry. O-minimal hybrid systems. *Mathematics of Control, Signals, and Systems*, 13(1):1–21, 2000.
- 26 Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. A new class of decidable hybrid systems. In *Proc. 2nd International Workshop on Hybrid Systems: Computation and Control (HSCC'99)*, volume 1569 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 1999.
- 27 Angus Macintyre and Alex J. Wilkie. On the decidability of the real exponential field. In *Kreiseliana*, pages 441–467. A. K. Peters, 1996.
- 28 David Marker. *Model theory: an introduction*, volume 217. Springer Science & Business Media, 2006.
- 29 Daniel J Miller. Constructing o-minimal structures with decidable theories using generic families of functions from quasianalytic classes. Research Report math.LO/1008.2575, arXiv, 2010.
- 30 Anand Pillay and Charles Steinhorn. Definable sets in ordered structures. *Bulletin of the American Mathematical Society*, 11(1), 1984.
- 31 Anand Pillay and Charles Steinhorn. Definable sets in ordered structures. I. *Transactions of the American Mathematical Society*, 295(2):565–592, 1986.
- 32 Anand Pillay and Charles Steinhorn. Discrete o-minimal structures. *Ann. Pure Appl. Logic*, 34(3):275–289, 1987. doi:10.1016/0168-0072(87)90004-2.
- 33 Anand Pillay and Charles Steinhorn. Definable sets in ordered structures. III. *Transactions of the American Mathematical Society*, 309(2):469–476, 1988.
- 34 Ashish Tiwari and Gaurav Khanna. Series of abstractions for hybrid automata. In C. J. Tomlin and M. R. Greenstreet, editors, *Hybrid Systems: Computation and Control HSCC*, volume 2289 of *LNCS*, pages 465–478. Springer, 2002.
- 35 Lou van den Dries. O-minimal structures. In *Proc. Logic, From Foundations to Applications*, pages 137–185. Oxford University Press, 1996.
- 36 Lou van den Dries. *Tame Topology and O-Minimal Structures*, volume 248 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1998.
- 37 Alex J. Wilkie. Model completeness results for expansions of the ordered field of real numbers by restricted Pfaffian functions and the exponential function. *Journal of the AMS*, 9(4):1051–1094, 1996.

A Contextual Reconstruction of Monadic Reflection

Toru Kawata

Department of Computer Science, The University of Tokyo, Tokyo, Japan

kawata@lyon.is.s.u-tokyo.ac.jp

Abstract

With the help of an idea of contextual modal logic, we define a logical system λ^{refl} that incorporates monadic reflection, and then investigate delimited continuations through the lens of monadic reflection. Technically, we firstly prove a certain universality of continuation monad, making the character of monadic reflection a little more clear. Next, moving focus to delimited continuations, we present a macro definition of shift/reset by monadic reflection. We then prove that $\lambda_{2\text{cont}}^{\text{refl}}$, a restriction of λ^{refl} , has exactly the same provability as $\lambda_{\text{pure}}^{\text{s/r}}$, a system that incorporates shift/reset. Our reconstruction of monadic reflection opens up a path for investigation of delimited continuations with familiar monadic language.

2012 ACM Subject Classification Theory of computation \rightarrow Type theory

Keywords and phrases Monadic Reflection, Delimited Continuations, shift/reset, Contextual Modal Logic, Curry-Howard Isomorphism

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.27

Acknowledgements I would like to thank Yoshihiko Kakutani, Yuichi Nishiwaki, and Yuito Murase for their valuable comments on contextual modality. I also thank the anonymous reviewers for their valuable comments.

1 Introduction

Every light is accompanied by darkness. Every term is accompanied by its continuation. Suppose that the part indicated by the underline in the following term is being evaluated:

$$1 + \underline{2 \times 3} - 4$$

In this case, the continuation of 2×3 is intuitively $\mathcal{A}(1 + [\] - 4)$, where $\mathcal{A}(M)$ means “compute M and then quit”. In other words, a continuation is the rest of the computation, and is therefore expressed as a term with just one “hole”. Some calculi[5][21] can turn a continuation into a function and use it in terms as if it were an ordinary function. Such manipulations of continuations are well-understood today, and indeed it is a well-known fact that these can be characterized as classical inferences via the Curry-Howard Isomorphism[10][21]. In the correspondence, the type of $\mathcal{A}(1 + [\] - 4)$ is understood to be $\mathbb{N} \rightarrow \perp$, assuming that \mathbb{N} is the type of natural numbers.

Delimited continuation[4][6][11] is a variant of continuation. In operational terms, a delimited continuation is explained as the rest of the computation *up to the nearest enclosing delimiter*. For example, consider the following term:

$$[1 + \underline{2 \times 3}] - 4$$

where we denote the delimiter by $[-]$. In this case, the delimited continuation of 2×3 is $1 + [\]$. As in the case of ordinary continuations, there are calculi that can turn a delimited



© Toru Kawata;

licensed under Creative Commons License CC-BY

27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 27; pp. 27:1–27:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

continuation into a function[4][15]. This perhaps seemingly trivial modification is in fact significant, enhancing the expressibility of continuations drastically. Delimited continuations are so expressive that it can realize, for example, coroutines, non-determinism, and statically typed printf[2]. This expressibility essentially comes from the fact that delimited continuations are composable as functions, whereas ordinary continuations are not since their codomains are \perp .

In addition to the above intriguing applications of delimited continuations, there exists one more elegant use of it: *monadic reflection*[7][8]. Monadic reflections allow programmers to write programs that involve monads as if those monads are “built-in” to the language, or in Haskell terminology, without `do`-notation. A little more specifically, in a system with monadic reflection, a monadic term $M : TA$ can be turned into a non-monadic term $\text{reflect}^T M : A$ without disrupting the intuitive behavior of M , easing the trouble of writing effectful programs in a purely-functional language.

On the other hand, however, delimited continuations are logically complicated. In contrast to the fact that ordinary continuations correspond to classical inferences, there does not seem to exist such a simple correspondence for delimited ones. Although there exists, for example, an interesting work that embeds a system with delimited continuations into a variant of classical logic[1], the embedding is still not “the end of the story” in that the original system with delimited continuations[4] does not allow classical inferences.

Thus, to sum up the plot, an elegant application is being derived from a rather opaque starting point. In this paper, with the help of an idea of contextual modal logic, we invert the direction of this storyline: We directly define a logical system λ^{eff} that incorporates monadic reflection, and then investigate delimited continuations through the lens of monadic reflection, obtaining a better understanding of delimited continuations. Our contribution in this paper is now summarized as follows:

- A modal logical system that incorporates monadic reflection,
- The universality of continuation monad with respect to monadic reflection,
- Logical understanding of delimited continuations via “quasi-double negation”,
- Equivalence of monadic reflection and delimited continuation in provability.

In the next section, we overview the idea of Fitch-style contextual modal logic to obtain the foundational idea of context layering. In the section, we also briefly review ordinary modal logic since it might be an unfamiliar concept for researchers on delimited continuations. Readers who are familiar with modal logic can safely skip the first part of the section. Using the idea of contextual modal logic, in Section 3, we define a system λ^{eff} that incorporates monadic reflection, and show the universality of continuation monad with respect to reflection, making the character of monadic reflection a little more clear. We also present a macro definition of shift/reset in the section. Section 4 is dedicated to an investigation of delimited continuations via the lens of monadic reflection. We show certain equivalence of these two concepts with respect to provability in the section. In Section 5, we put our contribution into perspective by comparing our work to related work. Section 6 concludes this paper with discussion of future work.

2 Fitch-Style Contextual Modal Logic

2.1 A Quick Tour of Modal Logic

Modal logic is a logic in which we can add certain modalities, or “flavors”, to propositions. In modal logic, one can state that a proposition is “necessarily” true, “possibly” true, or “should be” true, etc. Let us focus on necessity and possibility here. Typically, when a

proposition A is necessarily or possibly true under an assumption Γ , one writes $\Gamma \vdash \Box A$ or $\Gamma \vdash \Diamond A$, respectively. Using these operators, one can extend ordinary logic into modal one.

But what after all is necessity? What is possibility? The critical problem of modal logic in its early days was the very fact that people did not share any firm consensus on these concepts. Is a proposition $\Box A \rightarrow \Box \Box A$ true? How about $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$? Since people did not have any precise definitions of modalities, the answer naturally differed among people, which resulted in a lot of different variant of modal logics.

Fortunately, around 1960, this situation was recovered by Kripke[16][17], who presented a semantics for modal logic. He presented formal definitions of the two modalities in his semantics which employs a concept that would excite Sci-Fi enthusiasts and even others: possible worlds. His semantics introduces a set of possible worlds and an “accessibility relation” between them. Under this framework, for example, the meaning of necessity is defined as follows: $\Box A$ is true in a possible world w if and only if A is true in any world w' that is accessible from the world w . The various aspects of necessity are then characterized by how this accessibility relation is defined. For instance, if the accessibility relation under consideration is reflexive, the box modality validates $\Box A \rightarrow A$. If the relation is transitive, the modality validates $\Box A \rightarrow \Box \Box A$, etc. In this way, he gave formal definitions of “necessity” and “possibility”, or whatever they are called, in modal logic.

2.2 Fitch-Style Contextual Modal Logic

There still remains a pitfall. Modalities become tricky when they are combined with bindings. Let us take the example by Quine[23]. The sentence $\Box(8 > 7)$ is true as long as our box modality considered here validates $A \rightarrow \Box A$. Now, we know that the number of the planets in our solar system is 8. Let us define

$$N := (\text{the number of the planets in our solar system}).$$

Then $N = 8$ is true. Now, however, the statement $\Box(N > 7)$ is intuitively false, since we can consider a possible world in which we have, for instance, only 5 planets in our solar system. This phenomenon suggests that box modality has a delicate character with respect to bindings. Some might think that the box modality should contain information about its argument just like universal/existential quantifiers.

And here comes contextual modal logic. Contextual modal logic[19] is an attempt to generalize the modalities with contextual information. In this logic, the box modality is generalized from $\Box A$ to $[\Gamma]A$. The latter proposition intuitively reads as “ A is necessarily true under Γ ”. $[\Gamma]A$ degenerates to the ordinary box modality when Γ is empty. By generalizing box modality in this way, one can obtain a more fine-grained modality.

Multi-level Fitch-style contextual modal logic[20] is a variant of contextual modal logic. In the logic, the form of judgments are generalized to $\Gamma_1; \dots; \Gamma_n \vdash A$. Intuitively, this reads as “ A is true under Γ_n , is true under Γ_{n-1} , ..., is true under Γ_1 ”. In other words, this logic extends our vocabulary in the system, allowing statements of the form not only

$$A \text{ is true under } \Gamma$$

but also

$$\text{“}A \text{ is true under } \Gamma_2\text{” is true under } \Gamma_1.$$

By internalizing the meaning of this statement, we can incorporate quotation to our logic. For example, we can consider the following inference rules

$$\frac{\vec{\Pi}; \Gamma \vdash A}{\vec{\Pi} \vdash [\Gamma]A} \square_i \quad \frac{\vec{\Pi} \vdash [\Gamma]A}{\vec{\Pi}; \Gamma \vdash A} \square_e$$

which internalize quotation, or the relative pronoun “that”, as the box modality. It would be worth noting that this box modality defined above characterizes the so-called K modality. Furthermore, by extending these rules in a certain way, we can also characterize T, K4, and S4 modalities. Interested readers are referred to [18][20]. In this paper, however, we think that this quick explanation is sufficient for our purpose, and therefore do not go into further details here.

3 A Contextual Reconstruction of Monadic Reflection

3.1 A Logical System with Monadic Reflection

3.1.1 Syntax

The syntax of λ^{refl} is defined as follows. Here, the x and t in the following definition are a variable and a type-variable, respectively. We assume that the set of variables and that of type-variables are distinct.

$$\begin{aligned} A, B &::= t \mid A \rightarrow A \\ T &::= [] \mid t \mid T \rightarrow T \\ M, N, P, Q &::= x \mid \lambda x. M \mid M @ M \mid \text{reflect}^T M \mid \text{reify}^T M \\ \Gamma, \Delta &::= \emptyset \mid x : A, \Gamma \\ \vec{\Pi} &::= \cdot \mid \Gamma; \vec{\Pi} \\ \vec{L} &::= \cdot \mid T; \vec{L} \end{aligned}$$

We define $T[A]$ to be the proposition obtained by replacing all the occurrences of $[]$ in T with A . For example, when $T = [] \rightarrow B$, $T[A]$ is $A \rightarrow B$. Although the definition of T in the rule above is somewhat restricted, we can easily generalize it if necessary. We denote $T[A]$ by TA . We also abbreviate $\Gamma_1; \dots; \Gamma_n; \cdot$ as $\Gamma_1; \dots; \Gamma_n$, and $T_1; \dots; T_n; \cdot$ as $T_1; \dots; T_n$. Every judgment of λ^{refl} is of the form $\vec{L} \mid \vec{\Pi} \vdash M : A$.

3.1.2 Logic

The type system of λ^{refl} is as follows.

$$\begin{aligned} &\frac{}{\vec{L}; T \mid \vec{\Pi}; \Gamma, x : A \vdash x : A} \text{var} \\ &\frac{\vec{L} \mid \vec{\Pi}; \Gamma, x : A \vdash M : B}{\vec{L} \mid \vec{\Pi}; \Gamma \vdash \lambda x. M : A \rightarrow B} \rightarrow_i \quad \frac{\vec{L} \mid \vec{\Pi} \vdash M : A \rightarrow B \quad \vec{L} \mid \vec{\Pi} \vdash N : A}{\vec{L} \mid \vec{\Pi} \vdash M @ N : B} \rightarrow_e \\ &\frac{T; \vec{L} \mid \emptyset; \vec{\Pi} \vdash M : A}{\vec{L} \mid \vec{\Pi} \vdash \text{reify}^T M : TA} \text{reify} \quad \frac{\vec{L} \mid \vec{\Pi} \vdash M : TA}{T; \vec{L} \mid \Gamma; \vec{\Pi} \vdash \text{reflect}^T M : A} \text{reflect} \end{aligned}$$

There exist two side-conditions. Firstly, the rule \rightarrow_i can be applied only when M does not have any `reflects` that are not encapsulated by `reify`. This side-condition is required to define the reduction of this system in the ordinary call-by-value way. At the same time, however, this side-condition can seemingly be dropped by allowing reduction in abstraction, and we will revisit this point later. Secondly, when the rules `reify` or `reflect` are applied, the following two rules must be admissible without appealing `reify` and `reflect`:

$$\frac{\vec{L} \mid \vec{\Pi} \vdash M : A}{\vec{L} \mid \vec{\Pi} \vdash \text{return}^T M : TA} \text{ return}$$

$$\frac{\vec{L} \mid \vec{\Pi}; \Gamma \vdash M : TA \quad \vec{L} \mid \vec{\Pi}; \Gamma, x : A \vdash N : TB}{\vec{L} \mid \vec{\Pi}; \Gamma \vdash M \triangleright_x^T N : TB} \text{ bind}$$

where $\text{return}^T M$ and $M \triangleright_x^T N$ are “macros” defined by terms in the system¹. For example, suppose $T = []$. For this identity effect T , the rule `return` is vacuously admissible. The rule `bind` is also admissible for this effect since we have the following derivation tree:

$$\frac{\frac{\vec{L} \mid \vec{\Pi}; \Gamma, x : A \vdash N : B}{\vec{L} \mid \vec{\Pi}; \Gamma \vdash \lambda x. N : A \rightarrow B} \quad \vec{L} \mid \vec{\Pi}; \Gamma \vdash M : A}{\vec{L} \mid \vec{\Pi}; \Gamma \vdash (\lambda x. N)@M : B}$$

Thus, we are allowed to apply the rules `reify` and `reflect` when $T = []$. In this case, $\text{return}^{[]} M$ and $M \triangleright_x^[] N$ are understood as macros for M and $(\lambda x. N)@M$, respectively.

► **Definition 1.** A proposition A is *provable in λ^{refl}* when there exists a term N such that $[] \mid \emptyset \vdash N : A$ is derivable in λ^{refl} . Such N is said to be *well-typed*.

We write $\lambda^{\text{refl}} \vdash A$ when A is provable in λ^{refl} . We also write $\lambda^{\text{refl}} \vdash \mathcal{J}$ when \mathcal{J} is derivable in λ^{refl} . We use these notations for subsystems of λ^{refl} that we will define later, too.

3.1.3 Reduction

Values V , contexts E , and pure-contexts F are defined as follows.

$$\begin{aligned} V &::= x \mid \lambda x. M \\ E &::= [] \mid V@E \mid E@M \mid \text{reify}^T E \\ F &::= [] \mid V@F \mid F@M \end{aligned}$$

We define $E[M]$ in the same way as $T[A]$. The reduction of this system is defined as follows.

$$\begin{aligned} E[(\lambda x. M)@V] &\rightsquigarrow E[M\{x := V\}] \\ E[\text{reify}^T F[\text{reflect}^T M]] &\rightsquigarrow E[M \triangleright_x^T \text{reify}^T F[x]] \\ E[\text{reify}^T V] &\rightsquigarrow E[\text{return}^T V] \end{aligned}$$

Here, the second rule “factors out” a detour resulted from `reify/reflect`. Specifically, the rule translates a derivation tree

¹ Note that we do not impose the laws of monads here.

$$\frac{\frac{\frac{(\dots)}{\vec{L} \mid \vec{\Pi} \vdash M : TA}}{T; \vec{L} \mid \emptyset; \vec{\Pi} \vdash \text{reflect}^T M : A}}{(\dots)}}{T; \vec{L} \mid \emptyset; \vec{\Pi} \vdash F[\text{reflect}^T M] : B}}{\vec{L} \mid \vec{\Pi} \vdash \text{reify}^T F[\text{reflect}^T M] : TB}$$

into

$$\frac{\frac{(\dots)}{\vec{L} \mid \vec{\Pi} \vdash M : TA} \quad \frac{\frac{\frac{T; \vec{L} \mid \emptyset; \vec{\Pi}, x : A \vdash x : A}{(\dots)}}{T; \vec{L} \mid \emptyset; \vec{\Pi}, x : A \vdash F[x] : B}}{\vec{L} \mid \vec{\Pi}, x : A \vdash \text{reify}^T F[x] : TB}}{\vec{L} \mid \vec{\Pi} \vdash M \triangleright_x^T \text{reify}^T F[x] : TB}$$

The following elementary properties can be shown by ordinary induction.

► **Proposition 2.** *If a judgment $\vec{L} \mid \vec{\Pi}; \Gamma \vdash M : A$ is derivable in λ^{refl} , $\vec{L} \mid \vec{\Pi}; \Gamma, x : B \vdash M : A$ is also derivable for any fresh variable x and any type B .*

► **Proposition 3.** *If M is a well-typed term of type A , $M \rightsquigarrow N$ implies $N : A$.*

► **Proposition 4.** *If $\vec{L} \mid \vec{\emptyset} \vdash M : A$ is derivable, M is one of the followings:*

- $M = \lambda x. P$
- $M = E[R]$
- $M = F[\text{reflect}^T P]$

where R is one of $(\lambda x. P)@V$, $\text{reify}^T F[\text{reflect}^T P]$, $\text{reify}^T V$.

The last proposition implies the progress property:

► **Corollary 5.** *Every well-typed term M is either a value, or can be uniquely written as $E[R]$, where R is one of $(\lambda x. P)@V$, $\text{reify}^T F[\text{reflect}^T P]$, $\text{reify}^T V$.*

3.2 λ_2^{refl} , $\lambda_{2\text{cont}}^{\text{refl}}$: Restrictions of λ^{refl}

Now, let us investigate the character of continuation monad. The system λ_2^{refl} is defined to be the system obtained from λ^{refl} by restricting the form of judgments to the following two:

- $[\] \mid \Gamma \vdash M : A$
- $T; [\] \mid \emptyset; \Gamma \vdash M : A$

The system $\lambda_{2\text{cont}}^{\text{refl}}$ is also defined to be the system obtained by restricting the form of judgments to the following two:

- $[\] \mid \Gamma \vdash M : A$
- $\text{cont}_B; [\] \mid \emptyset; \Gamma \vdash M : A$

where cont_A stands for $([\] \rightarrow A) \rightarrow A$. For this $\text{cont}_{(-)}$, the rule `return` is admissible by the following derivation:

$$\frac{\frac{\vec{L} \mid \vec{\Pi}; \Gamma, k : A \rightarrow B \vdash k : A \rightarrow B}{} \quad \frac{\vec{L} \mid \vec{\Pi}; \Gamma, k : A \rightarrow B \vdash M : A}{}{\vec{L} \mid \vec{\Pi}; \Gamma, k : A \rightarrow B \vdash k @ M : B}}{\vec{L} \mid \vec{\Pi}; \Gamma \vdash \lambda k. k @ M : \text{cont}_B A}$$

and the rule `bind` is also admissible as follows:

$$\frac{\frac{\frac{\vec{L} \mid \vec{\Pi}; \Gamma, x : A \vdash N : \text{cont}_C B}{\vec{L} \mid \vec{\Pi}; \Delta \vdash N : \text{cont}_C B} \quad \frac{}{\vec{L} \mid \vec{\Pi}; \Delta \vdash w : B \rightarrow C}}{\vec{L} \mid \vec{\Pi}; \Delta \vdash N @ w : C}}{\frac{\vec{L} \mid \vec{\Pi}; \Gamma \vdash M : \text{cont}_C A \quad \vec{L} \mid \vec{\Pi}; \Gamma, w : B \rightarrow C \vdash \lambda x. N @ w : A \rightarrow C}{\vec{L} \mid \vec{\Pi}; \Gamma, w : B \rightarrow C \vdash M @ (\lambda x. N @ w) : C}}{\vec{L} \mid \vec{\Pi}; \Gamma \vdash \lambda w. M @ (\lambda x. N @ w) : \text{cont}_C B}}$$

where Δ is a shorthand for $\Gamma, x : A, w : B \rightarrow C$. These admissibilities give us a license to apply `reify` and `reflect` for this effect.

We will write $\text{reify}^A M$ for $\text{reify}^{\text{cont}^A} M$ and $\text{reflect}^A M$ for $\text{reflect}^{\text{cont}^A} M$.

3.3 Universality of Continuation Monad

The system $\lambda_{2\text{cont}}^{\text{refl}}$ is obviously a subsystem of λ_2^{refl} , and therefore any proposition that can be proved in $\lambda_{2\text{cont}}^{\text{refl}}$ can also be proved in λ_2^{refl} . Here, we show that the other direction is in fact also true.

► **Theorem 6.** *Let \mathcal{J} be a derivable judgment in λ_2^{refl} . If \mathcal{J} is of the form $[\] \mid \Gamma \vdash M : A$, there exists a term N such that $[\] \mid \Gamma \vdash N : A$ is derivable in $\lambda_{2\text{cont}}^{\text{refl}}$. If \mathcal{J} is of the form $T; [\] \mid \emptyset; \Gamma \vdash M : A$, for any type B , there exists a term N such that the judgment $\text{cont}_{TB}; [\] \mid \emptyset; \Gamma \vdash N : A$ is derivable in $\lambda_{2\text{cont}}^{\text{refl}}$.*

Proof. We prove the statement by induction on the derivation of \mathcal{J} . When \mathcal{J} is derived from `var`, `→i`, or `→e`, the proofs are routine. Suppose that \mathcal{J} is derived from `reflect` and of the form $T; [\] \mid \emptyset; \Gamma \vdash \text{reflect}^T M : A$. In this case we have $[\] \mid \Gamma \vdash M : TA$. By the induction hypothesis, there exists a term N such that $[\] \mid \Gamma \vdash N : TA$ is derivable in $\lambda_{2\text{cont}}^{\text{refl}}$. Now, we can construct the following valid derivation tree for any type B :

$$\frac{\frac{\frac{[\] \mid \Gamma \vdash N : TA \quad \frac{[\] \mid \Delta \vdash k : A \rightarrow TB \quad [\] \mid \Delta \vdash x : A}{[\] \mid \Gamma, k : A \rightarrow TB \vdash N : TA}}{[\] \mid \Gamma, k : A \rightarrow TB \vdash N \triangleright_x^T k @ x : TB}}{[\] \mid \Gamma \vdash \lambda k. N \triangleright_x^T k @ x : (A \rightarrow TB) \rightarrow TB}}{\text{cont}_{TB}; [\] \mid \emptyset; \Gamma \vdash \text{reflect}^{TB}(\lambda k. N \triangleright_x^T k @ x) : A}}$$

where the Δ is a shorthand for $\Gamma, k : A \rightarrow TB, x : A$.

Suppose that \mathcal{J} is derived from `reify` and of the form $[\] \mid \Gamma \vdash \text{reify}^T M : TA$. In this case, we have $T; [\] \mid \emptyset; \Gamma \vdash M : A$. Just as in the previous case, for any type B , there exists a term N such that $\text{cont}_{TB}; [\] \mid \emptyset; \Gamma \vdash N : A$. Since the B in this judgment is arbitrary, we can take it to be A . In other words, we have $\text{cont}_{TA}; [\] \mid \emptyset; \Gamma \vdash N : A$. Now we just need to construct the following derivation tree:

$$\frac{\frac{\text{cont}_{TA}; [\] \mid \emptyset; \Gamma \vdash N : A \quad \frac{[\] \mid \Gamma, x : A \vdash x : A}{[\] \mid \Gamma, x : A \vdash \text{return}^T x : TA}}{[\] \mid \Gamma \vdash \text{reify}^{TA} N : (A \rightarrow TA) \rightarrow TA} \quad \frac{}{[\] \mid \Gamma \vdash \lambda x. \text{return}^T x : A \rightarrow TA}}{[\] \mid \Gamma \vdash (\text{reify}^{TA} N) @ (\lambda x. \text{return}^T x) : TA}$$

► **Corollary 7.** $\lambda_2^{\text{refl}} \vdash A$ iff $\lambda_{2\text{cont}}^{\text{refl}} \vdash A$.

Note that we had to state the theorem only for provability, and not for term conversion. One might wonder why we did not define, for example, a translation by

$$\begin{aligned}
x^\natural &= x \\
(\lambda x. M)^\natural &= \lambda x. M^\natural \\
(M @ N)^\natural &= M^\natural @ N^\natural \\
(\text{reflect}^T M)^\natural &= \text{reflect}^{TB}(\lambda k. M^\natural \triangleright_x^T k @ x) \\
(\text{reify}^T M)^\natural &= (\text{reify}^{TA} M^\natural) @ (\lambda x. \text{return}^T x)
\end{aligned}$$

and state that $M \rightsquigarrow N \Leftrightarrow M^\natural \rightsquigarrow N^\natural$. This is because the translation makes the correspondence of redexes unclear. For example, in the following diagram, we cannot reduce the upper-right term into the lower-right term because the “redex” $1 + 1$ is encapsulated in the lambda abstraction.

$$\begin{array}{ccc}
\text{reflect}^T F[1 + 1] & \rightsquigarrow^\natural & \text{reflect}^{TB}(\lambda k. F^\natural[1 + 1] \triangleright_x^T k @ x) \\
\text{reduce} \downarrow & & \downarrow ? \\
\text{reflect}^T F[2] & \rightsquigarrow^\natural & \text{reflect}^{TB}(\lambda k. F^\natural[2] \triangleright_x^T k @ x)
\end{array}$$

This fact complicates discussion on term conversion. Faced with the complexity, we had to state the theorem only for provability, albeit we believe that our results can be strengthened to include an account of term conversion. In Section 4, we state the equivalence of monadic reflection and delimited continuations again “up to provability”. It is also because of this.

3.4 Deriving shift/reset from Monadic Reflection

When we instantiate the T in the rule `reflect` with `contB`, the rule behaves as if it were a rule of double-negation:

$$\frac{\vec{L} \mid \vec{\Pi} \vdash M : (A \rightarrow B) \rightarrow B}{\text{cont}_B; \vec{L} \mid \Gamma; \vec{\Pi} \vdash \text{reflect}^B M : A} \text{reflect}$$

Of course, this is not a properly classical inference since we do have a “debt” `contB`. Still, it would be natural to expect that we might be able to handle continuations in some form using this “quasi-double negation”, considering that classical inferences correspond to manipulations of continuations via the Curry-Howard Isomorphism.

And this is in fact true. In λ^{refl} , we can define the following macros:

$$\begin{aligned}
\lceil M \rceil^A &= (\text{reify}^A M) @ (\lambda z. z) \\
\mathcal{S}^A k. M &= \text{reflect}^A(\lambda k. \lceil M \rceil^A)
\end{aligned}$$

These macros behave in exactly the same way as specified in the ordinary operational semantics of shift/reset. Namely, we can easily check the followings:

$$\begin{aligned}
\lceil F[\mathcal{S}k. M] \rceil &\rightsquigarrow^* \lceil M\{k := \lambda x. \lceil F[x] \rceil\} \rceil, \\
\lceil V \rceil &\rightsquigarrow^* V.
\end{aligned}$$

Specifically, the former is justified by:

$$\begin{aligned}
& [F[\mathcal{S}k. M]] \\
&= (\text{reify } (F[\text{reflect } (\lambda k. [M])]))@(\lambda z. z) \\
&\rightsquigarrow ((\lambda k. [M]) \triangleright_x \text{reify } (F[x]))@(\lambda z. z) \\
&= (\lambda w. (\lambda k. [M])@(\lambda x. (\text{reify } (F[x]))@w))@(\lambda z. z) \\
&\rightsquigarrow (\lambda k. [M])@(\lambda x. (\text{reify } (F[x]))@(\lambda z. z)) \\
&= (\lambda k. [M])@(\lambda x. [F[x]]) \\
&\rightsquigarrow [M\{k := \lambda x. [F[x]}\}]
\end{aligned}$$

and the latter by:

$$[V] = (\text{reify } V)@(\lambda z. z) \rightsquigarrow (\text{return } V)@(\lambda z. z) = (\lambda k. k@V)@(\lambda z. z) \rightsquigarrow^* V.$$

4 Investigating Delimited Continuations with Monadic Reflection

4.1 A Logical System with Delimited Continuations

Our logical system with delimited continuations, $\lambda_{\text{pure}}^{\text{s/r}}$, is obtained from $\lambda_{\text{let}}^{\text{s/r}}$ [3], by

- fixing the answer types,
- restricting lambda abstractions to be pure, and
- making the application of the rule `exp` explicit.

4.1.1 Syntax

The syntax of $\lambda_{\text{pure}}^{\text{s/r}}$ is defined as follows.

$$\begin{aligned}
A, B &::= t \mid A \rightarrow A \\
M, N &::= x \mid \lambda x. M \mid M@N \mid \mathcal{S}^A k. M \mid [M]^A \mid [M]^A \\
\Gamma, \Delta &::= \emptyset \mid x : A, \Gamma
\end{aligned}$$

We often omit the type annotation in $\mathcal{S}^A k. M$, $[M]^A$, and $[M]^A$. The form of the judgments of $\lambda_{\text{pure}}^{\text{s/r}}$ is either $\Gamma \vdash M : A$ or $B \mid \Gamma \vdash M : A$.

4.1.2 Logic

The type system of $\lambda_{\text{pure}}^{\text{s/r}}$ is as follows.

$$\begin{array}{c}
\frac{}{\Gamma, x : A \vdash x : A} \text{var} \\
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B} \rightarrow_i \quad \frac{C \mid \Gamma \vdash M : A \rightarrow B \quad C \mid \Gamma \vdash N : A}{C \mid \Gamma \vdash M@N : B} \rightarrow_e \\
\frac{B \mid \Gamma, k : A \rightarrow B \vdash M : B}{B \mid \Gamma \vdash \mathcal{S}^B k. M : A} \text{shift} \quad \frac{A \mid \Gamma \vdash M : A}{\Gamma \vdash [M]^A : A} \text{reset} \quad \frac{\Gamma \vdash M : A}{B \mid \Gamma \vdash [M]^B : A} \text{exp}
\end{array}$$

► **Definition 8.** A proposition A is *provable* in $\lambda_{\text{pure}}^{\text{s/r}}$ if there exists a term N such that $\emptyset \vdash N : A$. We also say that such N is *well-typed*.

We define $\lambda_{\text{pure}}^{\text{s/r}} \vdash A$ and $\lambda_{\text{pure}}^{\text{s/r}} \vdash \mathcal{J}$ as in λ^{refl} .

4.2 Translations

We will compare $\lambda_{\text{pure}}^{\text{s/r}}$ with $\lambda_{2\text{cont}}^{\text{refl}}$. Towards that end, we syntactically distinguish the applications of $\lambda_{2\text{cont}}^{\text{refl}}$ at different levels. In other words, we separate the rule \rightarrow_e in $\lambda_{2\text{cont}}^{\text{refl}}$ into the following two:

$$\frac{[\] \mid \Gamma \vdash M : A \rightarrow B \quad [\] \mid \Gamma \vdash N : A}{[\] \mid \Gamma \vdash M \diamond^B N : B} \rightarrow_e^{\text{pure}}$$

$$\frac{\text{cont}_C; [\] \mid \emptyset; \Gamma \vdash M : A \rightarrow B \quad \text{cont}_C; [\] \mid \emptyset; \Gamma \vdash N : A}{\text{cont}_C; [\] \mid \emptyset; \Gamma \vdash M @ N : B} \rightarrow_e$$

We sometimes omit the annotation in the former application for brevity.

With the preparation above, we define two translations $(-)^{\sharp} : \lambda_{\text{pure}}^{\text{s/r}} \rightarrow \lambda_{2\text{cont}}^{\text{refl}}$ and $(-)^{\flat} : \lambda_{2\text{cont}}^{\text{refl}} \rightarrow \lambda_{\text{pure}}^{\text{s/r}}$ as follows:

$$\begin{array}{ll} x^{\flat} = x & x^{\sharp} = x \\ (\lambda x. M)^{\flat} = \lambda x. M^{\flat} & (\lambda x. M)^{\sharp} = \lambda x. M^{\sharp} \\ (M @ N)^{\flat} = M^{\flat} @ N^{\flat} & (M @ N)^{\sharp} = M^{\sharp} @ N^{\sharp} \\ (M \diamond^A N)^{\flat} = \llbracket [M^{\flat}]^A @ [N^{\flat}]^A \rrbracket^A & - \\ (\text{reflect}^A M)^{\flat} = \mathcal{S}^A k. \llbracket [M^{\flat}]^A @ [k]^A \rrbracket^A & (\mathcal{S}^A k. M)^{\sharp} = \text{reflect}^A (\lambda k. (\text{reify}^A M^{\sharp}) \diamond^A (\lambda x. x)) \\ (\text{reify}^A M)^{\flat} = \lambda k. \llbracket [k]^A @ M^{\flat} \rrbracket^A & (\llbracket M \rrbracket^A)^{\sharp} = (\text{reify}^A M^{\sharp}) \diamond^A (\lambda x. x) \\ - & (\llbracket M \rrbracket^A)^{\flat} = \text{reflect}^A (\lambda k. k \diamond^A M^{\sharp}) \end{array}$$

We extend these translations for $\lambda_{\text{pure}}^{\text{s/r}}$ -judgments

$$\begin{aligned} (\Gamma \vdash M : A)^{\flat} &= [\] \mid \Gamma \vdash M^{\flat} : A \\ (B \mid \Gamma \vdash M : A)^{\flat} &= \text{cont}_B; [\] \mid \emptyset; \Gamma \vdash M^{\flat} : A \end{aligned}$$

and for $\lambda_{2\text{cont}}^{\text{refl}}$ -judgments

$$\begin{aligned} ([\] \mid \Gamma \vdash M : A)^{\sharp} &= \Gamma \vdash M^{\sharp} : A \\ (\text{cont}_B; [\] \mid \emptyset; \Gamma \vdash M : A)^{\sharp} &= B \mid \Gamma \vdash M^{\sharp} : A. \end{aligned}$$

4.3 Equivalence of the Two Systems in Provability

Now, we present the equivalence of monadic reflection and delimited continuation.

► **Theorem 9.** $\lambda_{2\text{cont}}^{\text{refl}}$ and $\lambda_{\text{pure}}^{\text{s/r}}$ possess exactly the same provability. Specifically,

1. $\lambda_{\text{pure}}^{\text{s/r}} \vdash \mathcal{J}$ implies $\lambda_{2\text{cont}}^{\text{refl}} \vdash \mathcal{J}^{\sharp}$
2. $\lambda_{2\text{cont}}^{\text{refl}} \vdash \mathcal{J}$ implies $\lambda_{\text{pure}}^{\text{s/r}} \vdash \mathcal{J}^{\flat}$

Proof. (1) We prove the statement by the induction on the derivation of \mathcal{J} . The proofs for var , \rightarrow_i , $\rightarrow_e^{\text{pure}}$, \rightarrow_e are routine. Assume that \mathcal{J} is derived using shift and of the form $B \mid \Gamma \vdash \mathcal{S}^B k. M : A$. In this case, we have $B \mid \Gamma, k : A \rightarrow B \vdash M : B$. Applying the translation, we obtain a judgment $\text{cont}_B; [\] \mid \emptyset; \Gamma, k : A \rightarrow B \vdash M^{\sharp} : B$, which is justified by the induction hypothesis. Now we have the following derivation tree

$$\frac{\frac{\text{cont}_B; [] | \emptyset; \Gamma, k : A \rightarrow B \vdash M^\sharp : B}{[] | \Gamma, k : A \rightarrow B \vdash \text{reify}^B M^\sharp : (B \rightarrow B) \rightarrow B} \quad \frac{[] | \Gamma, k : A \rightarrow B, x : B \vdash x : B}{[] | \Gamma, k : A \rightarrow B \vdash \lambda x. x : B \rightarrow B}}{[] | \Gamma, k : A \rightarrow B \vdash (\text{reify}^B M^\sharp) \diamond^B (\lambda x. x)} \\
\frac{[] | \Gamma \vdash \lambda k. (\text{reify}^B M^\sharp) \diamond^B (\lambda x. x) : (A \rightarrow B) \rightarrow B}{\text{cont}_B; [] | \emptyset; \Gamma \vdash \text{reflect}^B (\lambda k. (\text{reify}^B M^\sharp) \diamond^B (\lambda x. x)) : A}$$

where x is a variable which does not occur in M^\sharp, Γ . The conclusion of this derivation tree is equal to the result of the translation of $B | \Gamma \vdash \mathcal{S}^B k. M : A$.

Assume that \mathcal{J} is derived using `reset` and of the form $\Gamma \vdash [M]^A : A$. In this case, we have $A | \Gamma \vdash M : A$. Translating this judgment, we obtain a valid judgment $\text{cont}_A; [] | \emptyset; \Gamma \vdash M^\sharp : A$. Now, we can construct the following tree:

$$\frac{\frac{\text{cont}_A; [] | \emptyset; \Gamma \vdash M^\sharp : A}{[] | \Gamma \vdash \text{reify}^A M^\sharp : (A \rightarrow A) \rightarrow A} \quad \frac{[] | \Gamma, x : A \vdash x : A}{[] | \Gamma \vdash \lambda x. x : A \rightarrow A}}{[] | \Gamma \vdash (\text{reify}^A M^\sharp) \diamond^A (\lambda x. x) : A}$$

Hence we have $[] | \Gamma \vdash (\text{reify}^A M^\sharp) \diamond^A (\lambda x. x) : A$, which is equal to the result of the translation of $\Gamma \vdash [M]^A : A$.

Assume that \mathcal{J} is derived using `exp` and of the form $B | \Gamma \vdash [M]^B : A$. In this case, we have $\Gamma \vdash M : A$, and therefore $[] | \Gamma \vdash M^\sharp : A$. Now we have the following derivation:

$$\frac{\frac{[] | \Gamma, k : A \rightarrow B \vdash k : A \rightarrow B}{[] | \Gamma, k : A \rightarrow B \vdash k \diamond^B M^\sharp : B} \quad \frac{[] | \Gamma \vdash M^\sharp : A}{[] | \Gamma, k : A \rightarrow B \vdash M^\sharp : A}}{[] | \Gamma, k : A \rightarrow B \vdash \lambda k. k \diamond^B M^\sharp : (A \rightarrow B) \rightarrow B} \\
\frac{[] | \Gamma, k : A \rightarrow B \vdash \lambda k. k \diamond^B M^\sharp : (A \rightarrow B) \rightarrow B}{\text{cont}_B; [] | \emptyset; \Gamma \vdash \text{reflect}^B (\lambda k. k \diamond^B M^\sharp) : A}$$

which concludes our proof for (1).

(2) Again, we prove the statement on the derivation of \mathcal{J} . We present proofs only for non-trivial cases: `reflect`, `reify`, and $\rightarrow_e^{\text{pure}}$. Assume that \mathcal{J} is derived from `reflect` and of the form $\text{cont}_B; [] | \emptyset; \Gamma \vdash \text{reflect}^B M : A$. Then we have $[] | \Gamma \vdash M : (A \rightarrow B) \rightarrow B$, which implies $\Gamma \vdash M^b : (A \rightarrow B) \rightarrow B$ by the induction hypothesis. Now we have the following derivation tree:

$$\frac{\frac{\Gamma \vdash M^b : (A \rightarrow B) \rightarrow B}{B | \Gamma \vdash [M^b]^B : (A \rightarrow B) \rightarrow B} \quad \frac{\Gamma, k : A \rightarrow B \vdash k : A \rightarrow B}{B | \Gamma, k : A \rightarrow B \vdash [k]^B : A \rightarrow B}}{B | \Gamma, k : A \rightarrow B \vdash [M^b]^B \@[k]^B : B} \\
\frac{B | \Gamma, k : A \rightarrow B \vdash [M^b]^B \@[k]^B : B}{B | \Gamma \vdash \mathcal{S}^B k. [M^b]^B \@[k]^B : A}$$

The conclusion of this tree is equal to $(\text{cont}_B; [] | \emptyset; \Gamma \vdash \text{reflect}^B M : A)^b$.

Assume that \mathcal{J} is derived from `reify` and of the form $[] | \Gamma \vdash \text{reify}^B M : (A \rightarrow B) \rightarrow B$. In this case, we have $\text{cont}_B; [] | \emptyset; \Gamma \vdash M : A$, which implies $B | \Gamma \vdash M^b : A$. The tree that we need to construct is the following one:

$$\frac{\frac{\Gamma, k : A \rightarrow B \vdash k : A \rightarrow B}{B \mid \Gamma, k : A \rightarrow B \vdash [k]^B : A \rightarrow B} \quad \frac{B \mid \Gamma \vdash M^b : A}{B \mid \Gamma, k : A \rightarrow B \vdash M^b : A}}{\frac{B \mid \Gamma, k : A \rightarrow B \vdash [k]^B @ M^b : B}{\Gamma, k : A \rightarrow B \vdash \llbracket [k]^B @ M^b \rrbracket^B : B}}}{\Gamma \vdash \lambda k. \llbracket [k]^B @ M^b \rrbracket^B : (A \rightarrow B) \rightarrow B}$$

Finally, assume that \mathcal{J} is derived from $\rightarrow_e^{\text{pure}}$ and of the form $[\] \mid \Gamma \vdash M \diamond^B N : B$. In this case, we have $[\] \mid \Gamma \vdash M : A \rightarrow B$ and $[\] \mid \Gamma \vdash N : A$ for some A . We can therefore construct the following derivation tree:

$$\frac{\frac{\Gamma \vdash M^b : A \rightarrow B}{B \mid \Gamma \vdash [M^b]^B : A \rightarrow B} \quad \frac{\Gamma \vdash N^b : A}{B \mid \Gamma \vdash [N^b]^B : A}}{\frac{B \mid \Gamma \vdash [M^b]^B @ [N^b]^B : B}{\Gamma \vdash \llbracket [M^b]^B @ [N^b]^B \rrbracket^B : B}}$$

which concludes our proof for (2). ◀

► **Corollary 10.** $\lambda_{\text{pure}}^{s/r} \vdash A$ iff $\lambda_{2\text{cont}}^{\text{refl}} \vdash A$.

5 Related Work

5.1 Monadic Reflection

In his seminal work, Filinski[7] introduced the idea of monadic reflection with its implementation by shift/reset. His work is one of our main motivations of this paper. It would be worth to note that the type system of the calculus with shift/reset in his paper is different from the original one[4], and therefore from the one that we have discussed in this paper. Indeed, for example, the original calculus is known to be strongly normalizing, whereas his calculus is not[14].

Forster et al.[9] compares the expressibility of effect handlers, monadic reflection, and delimited continuations. In the paper, among others, they show that delimited continuations and monadic reflection can express each other in untyped setting. At the same time, they show that their translation from delimited continuations to monadic reflection preserves types, whereas the translation of the opposite direction does not. Comparing to their work, our reconstruction can be understood as a proposal of an answer to the question of how we can preserve types in both directions of these translations.

Zeilberger[24] discusses delimited continuations in his somewhat non-standard calculus which incorporates polarity. He claims in the paper that monadic reflection is essentially the isomorphism in the Yoneda lemma. His observation might play an important role when we construct a categorical semantics of our system.

5.2 Logical Meaning of Delimited Continuations

Ariola et al.[1] explains the logical meaning of delimited continuations by translating a system with a dynamic variable, which can interpret shift/reset with answer-type modification, into a logical system with *subtraction*. In terms of classical logic, subtraction $A - B$ is dual to implication $A \rightarrow B$. Namely, $A - B = A \wedge (\neg B)$. A virtue of their system would be the fact that it explains the meaning of answer-type modification. At the same time, however, it should be noted that the subtractive system allows classical inference, whereas the original system with shift/reset does not allow double-negation.

Kameyama [12] covers a variant of delimited continuation operators which differs a little from shift/reset. In his work, he considers a side-condition that the operation that corresponds to shift can be used only when the operation that corresponds to reset will appear later. Through the lens of our reconstruction, this condition seems to be closely related to the role of the context stack in our system.

6 Conclusion

We have reconstructed monadic reflection using the idea of contextual modal logic, and exploited it to investigate delimited continuations, obtaining the equivalence of these two concepts in provability.

In this paper, we have focused on delimited continuations with pure abstractions. It might be possible to extend our work and encompass impure abstractions. Indeed, we can extend the pure-contexts in λ^{refl} as follows:

$$F ::= [] \mid V@F \mid F@M \mid \lambda x. F$$

and drop the side-condition of \rightarrow_i that we imposed in Section 3. With some modification to the definition of the values, we have the following reduction, for example:

$$E[\text{reify}^T(\lambda x. \text{reflect}^T M)] \rightsquigarrow E[M \triangleright_y^T \text{reify}^T(\lambda x. y)].$$

Note that our layered context ensures that M does not have x as free variable. By exploiting this fact and the macro-definition of shift/reset in λ^{refl} , we might be able to realize impure functions.

Another direction of future work is categorical semantics of λ^{refl} . Ordinary contextual modal logic already has a categorical semantics based on iterative enrichment[20]. It might be possible to explain λ^{refl} based on their calculus.

Decomposing the effect T into contextual modalities is also an interesting topic. In [22], the authors decompose the modality that corresponds to T in our system into $\diamond\Box$. Using the contextual possibility, it might be possible to decompose our T and derive the rule `reify`, `reflect`.

We have restricted the depth of the context stack of $\lambda_{2\text{cont}}^{\text{refl}}$ to be 2. It might also be possible to generalize the equivalence between delimited continuation and monadic reflection to the “iterative” one by dropping this restriction, clarifying the character of `shiftn/resetn` in the CPS hierarchy[13].

References

- 1 Zena Ariola, Hugo Herbelin, and Amr Sabry. A Type-Theoretic Foundation of Delimited Continuations. *Higher-Order and Symbolic Computation*, 2007.
- 2 Kenichi Asai. On typing delimited continuations: three new solutions to the printf problem. *Higher-Order and Symbolic Computation*, 22(3):275–291, Sep 2009. doi:10.1007/s10990-009-9049-5.
- 3 Kenichi Asai and Yuki Yoshi Kameyama. Polymorphic delimited continuations. In *Programming Languages and Systems*, pages 239–254. Springer Berlin Heidelberg, 2007. doi:10.1007/978-3-540-76637-7_16.
- 4 Olivier Danvy and Andrzej Filinski. A functional abstraction of typed contexts. Technical report, Institute of Datalogy, University of Copenhagen, 1989.
- 5 Matthias Felleisen, Daniel Friedman, Eugene Kohlbecker, and Bruce Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52(3):205–237, 1987. doi:10.1016/0304-3975(87)90109-5.

- 6 Mattias Felleisen. The theory and practice of first-class prompts. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '88, pages 180–190, New York, NY, USA, 1988. ACM. doi:10.1145/73560.73576.
- 7 Andrzej Filinski. Representing monads. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '94, pages 446–457. ACM, 1994. doi:10.1145/174675.178047.
- 8 Andrzej Filinski. Monads in action. *SIGPLAN Not.*, 45(1):483–494, 2010. doi:10.1145/1707801.1706354.
- 9 Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. On the expressive power of user-defined effects: Effect handlers, monadic reflection, delimited control. *Proceedings of the ACM on Programming Languages*, 1(ICFP):13:1–13:29, 2017. doi:10.1145/3110257.
- 10 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.
- 11 Gregory Johnson. GL – a denotational testbed with continuations and partial continuations as first-class objects. *SIGPLAN Not.*, 22(7):165–176, 1987. doi:10.1145/960114.29668.
- 12 Yuki Yoshi Kameyama. Towards logical understanding of delimited continuations. In *Continuations Workshop*, pages 27–33, 2001.
- 13 Yuki Yoshi Kameyama. Axioms for delimited continuations in the CPS hierarchy. In *Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 20-24, 2004, Proceedings*, pages 442–457, 2004. doi:10.1007/978-3-540-30124-0_34.
- 14 Yuki Yoshi Kameyama and Kenichi Asai. Strong normalization of polymorphic calculus for delimited continuations. In *Symbolic Computation in Software Science*, 2008.
- 15 Oleg Kiselyov and Chung-chieh Shan. A substructural type system for delimited continuations. In *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, pages 223–239, 2007. doi:10.1007/978-3-540-73228-0_17.
- 16 Saul Kripke. A completeness theorem in modal logic. *The Journal of Symbolic Logic*, 24(1):1–14, 1959. URL: <http://www.jstor.org/stable/2964568>.
- 17 Saul Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16(1963):83–94, 1963.
- 18 Yuito Murase. Kripke-style contextual modal type theory. Work-in-progress report at Logical Frameworks and Meta-Languages, 2017.
- 19 Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):23:1–23:49, 2008. doi:10.1145/1352582.1352591.
- 20 Yuichi Nishiwaki, Yoshihiko Kakutani, and Yuito Murase. Modality via iterated enrichment, 2018. arXiv:1804.02809. arXiv:arXiv:1804.02809.
- 21 Michel Parigot. $\lambda\mu$ -calculus: An algorithmic interpretation of classical natural deduction. *Lecture Notes in Computer Science*, 624:190–201, 1992. doi:10.1007/BFb0013061.
- 22 Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, aug 2001. doi:10.1017/S0960129501003322.
- 23 Willard Quine. Reference and modality. In *Journal of Symbolic Logic*, pages 139–159. Harvard University Press, 1953.
- 24 Noam Zeilberger. Polarity and the logic of delimited continuations. In *Proceedings of the 2010 25th Annual IEEE Symposium on Logic in Computer Science*, LICS '10, pages 219–227, Washington, DC, USA, 2010. IEEE Computer Society. doi:10.1109/LICS.2010.23.

An Algebraic Decision Procedure for Two-Variable Logic with a Between Relation

Andreas Krebs

Universität Tübingen, Germany

Kamal Lodaya

The Institute of Mathematical Sciences, Chennai, India

Paritosh K. Pandya

Tata Institute of Fundamental Research, Mumbai, India

Howard Straubing

Boston College, USA

Abstract

In earlier work (LICS 2016), the authors introduced two-variable first-order logic supplemented by a binary relation that allows one to say that a letter appears between two positions. We found an effective algebraic criterion that is a necessary condition for definability in this logic, and conjectured that the criterion is also sufficient, although we proved this only in the case of two-letter alphabets. Here we prove the general conjecture. The proof is quite different from the arguments in the earlier work, and required the development of novel techniques concerning factorizations of words. We extend the results to binary relations specifying that a factor appears between two positions.

2012 ACM Subject Classification Theory of computation → Algebraic language theory, Theory of computation → Finite Model Theory

Keywords and phrases two-variable logic, finite model theory, algebraic automata theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.28

Acknowledgements We would like to thank Boston College, IMSc and TIFR for hosting our collaborative visits.

1 Introduction

In this paper we work with finite word models. The first-order definable languages – those definable in the logic $FO[<]$ – were shown equivalent to starfree expressions by the work of Schützenberger [14], McNaughton and Papert [9]. The algebraic viewpoint established decidability of the definability question, that is, whether a given regular language is first-order definable. The first level of the quantifier alternation hierarchy was characterized by Knast [7]. Recently Place and Zeitoun characterized some more levels of the hierarchy [12, 13]. Two-variable logic was algebraically characterized by Thérien and Wilke [20]. They also showed decidability of its definability, and also of levels of the until hierarchy of temporal logic LTL , which was shown equivalent to first-order logic by Kamp [6].

In our earlier paper [8] we extended two-variable logic over finite words with between relations and studied this logic $FO^2[<, \text{bet}]$ and associated temporal logics. A between relation $a(x, y)$, for letters a of the finite alphabet, says that there is a position z labelled with the letter a such that $x < z < y$. The monoid variety $\mathbf{M}_e\mathbf{DA}$ is obtained by applying an operation M_e (see Section 2) to the variety \mathbf{DA} of two-variable logic [15]. We showed that



© Andreas Krebs, Kamal Lodaya, Paritosh Pandya, and Howard Straubing; licensed under Creative Commons License CC-BY

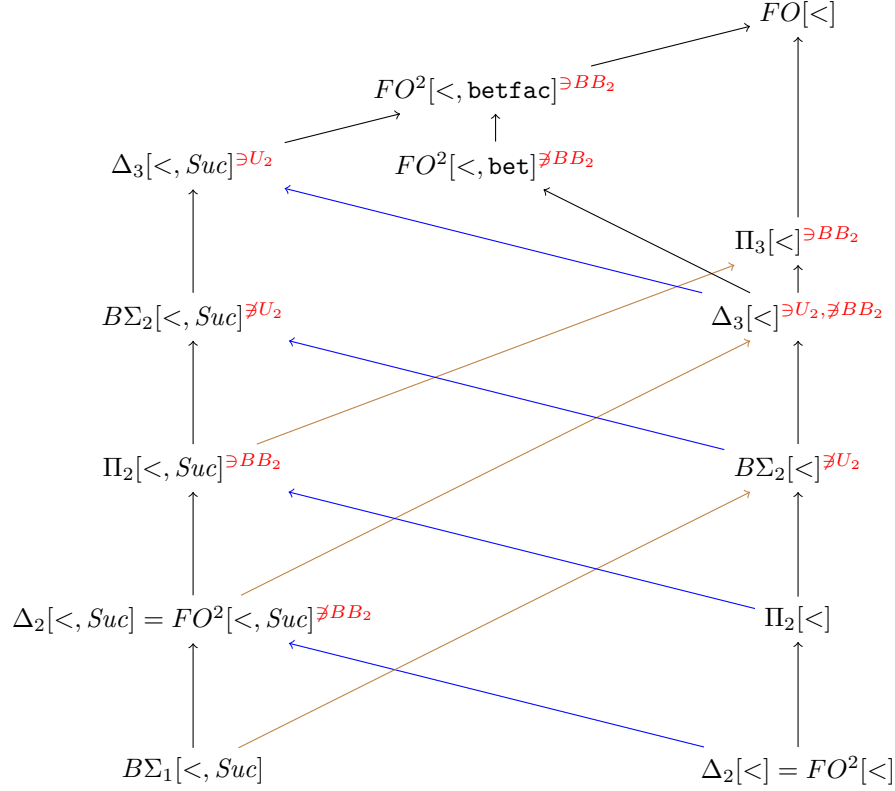
27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 28; pp. 28:1–28:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Dot depth and quantifier alternation hierarchies. The language U_2 over alphabet $A = \{a, b, c\}$ is $(A^* \setminus (A^*ac^*aA^*)) \cup (A^* \setminus (A^*bc^*bA^*))ac^*aA^*$, it consists of words which have no occurrence of bc^*b before an occurrence of ac^*a . The language BB_2 over $\{a, b\}$ is $(a(ab)^*b)^*$.

$\mathbf{M}_e\mathbf{DA}$ is an upper bound for $FO^2[<, \text{bet}]$, cutting across the quantifier alternation and until nesting depth hierarchies. We conjectured that this bound is tight and were able to show this for alphabets of size two. In this paper we establish this conjecture. Hence definability of a regular language in $FO^2[<, \text{bet}]$ is decidable. The variety $\mathbf{M}_e\mathbf{DA}$ first appeared in a paper by Weil [22]. Thus we provide a logical characterization of this variety.

The proof is somewhat intricate. We develop new techniques of factorization which are amenable to simulation using logic. At the end we rely on some hard algebra: the theorem on the locality of variety \mathbf{DA} , first shown by Almeida [1, 11]. Building on these techniques we show another main result, that the semigroup variety $\mathbf{M}_e\mathbf{DA} * \mathbf{D}$, obtained by applying to $\mathbf{M}_e\mathbf{DA}$ semidirect products with semigroups for the definite languages, characterizes a simple extension of our logic $FO^2[<, \text{bet}]$ to between relations $\langle u \rangle(x, y)$, for words u over the alphabet, which say that u is a factor, or substring, contained at positions between x and y . (So there are infinitely many between relations in this extended logic $FO^2[<, \text{betfac}]$.) The techniques we use here come from early work on providing a “delay” bound to varieties such as $\mathbf{DA} * \mathbf{D}$ [17, 21].

For the reader familiar with the lower levels of the quantifier alternation hierarchy of first-order logic (see [4] for a survey), these are the classes on the right in Figure 1. Those on the left are the classes of the original dot depth hierarchy of Cohen and Brzozowski [3]. The logics which we have introduced in [8] and in this paper are at top centre. They have a nonempty intersection with every level of both the hierarchies. The two example languages have played a prominent role in our work.

We also gave in [8] a tight complexity of *Expspace* for satisfiability of $FO^2[<, \text{bet}]$. The techniques extend to provide the same complexity bounds for $FO^2[<, \text{betfac}]$. This is an exponential blowup over *LTL*, but as noted in our earlier paper, these logics allow succinct binary representation of threshold constraints.

2 Setup

We denote by $FO^2[<, \text{Suc}]$ two-variable first-order logic with the successor relation *Suc* and the order relation $<$, interpreted in finite words over a finite alphabet A . (As usual, this stands for both the set of formulas, and the family of languages over A defined by such formulas.) Variables in first-order formulas are interpreted as positions in a word, and for each letter $a \in A$ there is a unary relation $a(x)$, interpreted to mean ‘the letter in position x is a ’. Thus sentences in this logic define properties of words, or, what is the same thing, languages $L \subseteq A^*$. Two-variable logic over words has been extensively studied, and has many equivalent characterizations in terms of temporal logic, regular languages, and the algebra of finite semigroups. (See, for instance, [19] and the many references cited therein.)

In [8] we extended $FO^2[<, \text{Suc}]$ to express ‘betweenness’ with only two variables. More precisely, predicates

$$a(x, y) = \exists z(a(z) \wedge x < z \wedge z < y),$$

which assert that there is an occurrence of the letter a strictly between x and y , were added to form the logic $FO^2[<, \text{bet}]$. We also showed that counting the number of occurrences of the letter between x and y upto a threshold is definable in $FO^2[<, \text{bet}]$. In Section 7 we will consider a further extension of this logic where we allow specification of factors between positions x and y .

We showed that languages defined by sentences of this logic satisfy an algebraic condition, which we explain next. For further background on the basic algebraic notions in this section, see Pin [10].

A *semigroup* is a set together with an associative multiplication. It is a *monoid* if it also has a multiplicative identity 1.

All of the languages defined by sentences of $FO[<]$ are regular languages. Our characterization of languages in these logics is based on properties of the *syntactic semigroup* $S(L)$ (resp. *syntactic monoid* $M(L)$) of a regular language L . This is the transition semigroup (monoid) of the minimal deterministic automaton recognizing L , and therefore finite. Equivalently, $S(L)$ is the smallest semigroup S that *recognizes* L , that is: There is a homomorphism $h : A^+ \rightarrow S$ and a subset $X \subseteq S$ such that $L = h^{-1}(X)$ (and similarly for monoids).

Let S be a finite semigroup. An *idempotent* $e \in S$ is an element satisfying $e^2 = e$.

If S is a finite semigroup and $s_1, s_2 \in S$, we write $s_1 \leq_{\mathcal{J}} s_2$ if $s_1 = rs_2t$ for some $r, t \in S$. This is a preorder, the so-called \mathcal{J} -ordering on S . Let $E(S)$ denote the set of all idempotents in S . If $e \in E(S)$, then M_e denotes the submonoid of M generated by the set $\{s \in S : e \leq_{\mathcal{J}} s\}$. The operation M_e appears in an unpublished memo of Schützenberger cited by Brzozowski [2]. He uses the submonoid generated by the generators of an idempotent element e of a semigroup. For example, if abc mapped to an idempotent element e , M_e would correspond to the language $(a + b + c)^*$.

The operation can be used at the level of semigroup and monoid classes. Thus the variety $\mathbf{M}_e\mathbf{DA}$ has monoids M , all of whose submonoids of the form $eM_e e$ for $e \in E(M)$, are in the variety \mathbf{DA} . Our main result is:

► **Theorem 1.** *Let $L \subseteq A^*$. L is definable in $FO^2[<, \text{bet}]$ iff $M(L) \in \mathbf{M}_e\mathbf{DA}$.*

In our earlier paper [8], we proved necessity of the algebraic condition, but only proved sufficiency in the case $|A| = 2$. Sections 3 to 6 are devoted to the proof of sufficiency for general alphabets.

The logical machinery we will use is quite standard (see [18]). In our paper [8], we defined Ehrenfeucht-Fraïssé games for the logic $FO^2[<, \mathbf{bet}]$. We use the games in this paper to prove the existence of an $FO^2[<, \mathbf{bet}]$ formula θ , by the equivalent formulation that there is an integer $k > 0$ such that, if $(w, i), (w', j)$ are marked words in which i and j are inequivalent, then Player 1 has a winning strategy in the k -round game for $FO^2[<, \mathbf{bet}]$ in $(w, i), (w', j)$. That is, $(w, i) \models \theta, (w', j) \not\models \theta$, so Player 1 has a winning strategy in the k -round game, where k is the quantifier depth of θ . Conversely, suppose Player 1 always has such a winning strategy. Consider all marked words (w, i) , and take the union of all the \equiv_k -classes of these w_i . This union is defined by a depth- k formula which we call θ . If there were any $(w', j) \models \theta$ where j is inequivalent, then we would have some (w, i) with $(w, i) \equiv_k (w', j)$, contrary to hypothesis. So θ is satisfied by exactly the required (w, i) .

Notation. If $w \in A^*$, then we write $\alpha(w)$ to denote the set of letters that occur in w . We will interpret $a(x, y)$ to be false whenever $y \leq x$.

3 The factorization sequence

We are going to prove Theorem 1, that our algebraic condition from [8] indeed holds over all alphabets. We only need to prove one direction.

► **Lemma 2** ($M_e\mathbf{DA}$ characterizes $FO^2[<, \mathbf{bet}]$). *Suppose finite monoid M satisfies the property $e \cdot M_e \cdot e \in \mathbf{DA}$ for all $e \in E(M)$. Then for all $m \in M$, $h^{-1}(m) \in FO^2[<, \mathbf{bet}]$.*

This will be proved by induction on the alphabet size. It is trivial for a one-letter alphabet, so assume $|A| > 1$ and that the theorem holds for all strictly smaller alphabets.

The bulk of the proof is combinatorics on words and finite model theory. We only use the algebra at the end.

For now we distinguish a letter $a \in A$, and restrict our attention to a -words w with the following three properties:

- $\alpha(w) = A$
- a is the first letter of w
- a is the *last* letter to appear in a *right-to-left* scan of w ; that is, $w = xay$ where $\alpha(y) = A \setminus \{a\}$.

We describe an algorithm for constructing a sequence of factorizations for any a -word. Each step of the algorithm is divided into two sub-steps, and we will refer to each of these sub-steps as a factorization scheme. The factors that occur in each scheme are formed by concatenating factors from the previous scheme. That is, at each step, we clump smaller factors into larger ones, so the number of factors decreases (non-strictly) at each step.

We begin by putting a linear ordering $<$ on the set of proper subalphabets of A that contain the letter a . This will be a topological sort of the subset partial order. That is, if B, C are two such subalphabets with $B \subsetneq C$, then $B < C$, but otherwise the ordering is arbitrary. For example, with $A = \{a, b, c, d\}$, we can take

$$\{a\} < \{a, b\} < \{a, c\} < \{a, b, c\} < \{a, d\} < \{a, b, d\} < \{a, c, d\},$$

as one of many possibilities. One way to think about our techniques is as a refinement of Thérien and Wilke's combinatorial characterization of **DA** [20] which only used the inclusion order over an alphabet.

Here is the algorithm, which is the new development over **DA**:

- Initially factor w as $au_1 \cdots au_k$, where each $\alpha(u_i)$ is properly contained in A .
- For each proper subalphabet B of A with $a \in B$, following the linear order
 - For each factor u such that $\alpha(u) = B$, combine all sequences of consecutive factors of this kind into a single factor. We say that B is now *collected*.
 - For each factor u such that $\alpha(u) = B$, combine each such factor with the factor immediately to its right. We say that B is now *capped*.

Here is an example. We begin with an a -word and its initial factorization:

$$adccdcc \cdot adc \cdot a \cdot a \cdot a \cdot addcccccdbcdc \cdot a \cdot ac \cdot abcbbd$$

We use the ordering in the example above. We start with $B = \{a\}$ and collect B :

$$adccdcc \cdot adc \cdot aaa \cdot addcccccdbcdc \cdot a \cdot ac \cdot abcbbd$$

then cap it:

$$adccdcc \cdot adc \cdot aaaaddcccccdbcdc \cdot aac \cdot abcbbd$$

We choose $B = \{a, b\}$. There is nothing to do here, because no factor contains just a and b . $B = \{a, c\}$ is already collected, because there is no pair of consecutive factors with this alphabet, so we cap it:

$$adccdcc \cdot adc \cdot aaaaddcccccdbcdc \cdot aacabcbbd$$

The next subalphabet in order that occurs as a factor is $\{a, c, d\}$. We collect:

$$adcccccadc \cdot aaaaddcccccdbcdc \cdot aacabcbbd$$

then cap:

$$adcccccaddadaaaaaddcccccdbcdc \cdot acabcbbd$$

Let us make a few general observations about this algorithm: Every proper subset of A containing a that occurs as the alphabet of a factor will eventually be capped, because the rightmost factor au_k of the initial factorization contains all the letters of A . Once B has been collected, there is no pair of consecutive factors with content B . Once B has been capped, there are no more factors with content B nor with strictly smaller content. Thus at the end of the process, every factor contains all the letters of A .

Note as well that immediately after a subalphabet B is collected to create a (possibly) larger factor u , both the factor immediately to the right of u and immediately to the left of u must contain a letter that is not in u .

4 Starts and jumps

We establish below several model-theoretic properties of the factorization schemes produced by the above algorithm.

► **Lemma 3.**

- (a) *There is a formula $start$ in $FO^2[<, \mathbf{bet}]$ such that for all a -words w , $(w, i) \models start(x)$ if and only if i is the first position in a factor of w .*
- (b) *Let $\phi_1(x)$ be a formula in $FO^2[<, \mathbf{bet}]$. Then there is a formula $next$ in $FO^2[<, \mathbf{bet}]$ with the following property: Let w be an a -word and let i be the first position in some factor of w that is not the rightmost factor. Then $(w, i) \models \phi_1(x)$ if and only if $(w, i_{succ}) \models next(x)$, where i_{succ} is the first position in the next factor of w . We also define the analogous property, with ‘rightmost’ replaced by ‘leftmost’, $next$ by previous, and i_{succ} by i_{pred} .*

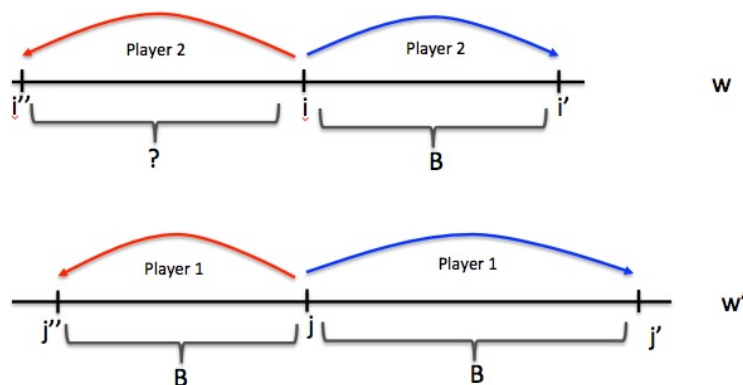
Proof. We prove these properties by induction on the construction of the sequence of factorization schemes. That is, we prove that they hold for the initial factorization scheme, and that they are preserved in each sub-step of the algorithm. For the induction, we will use Ehrenfeucht-Fraïssé games for the logic $FO^2[<, \mathbf{bet}]$ to argue for the existence of the formula $start = start_\tau$ (see Section 2).

We note that the claim in Item (b) implies the condition on games (possibly with different parameters). If for every formula ϕ_1 there is a corresponding ‘successor’ formula $next$, then there is some constant c such that $\text{qd}(next) \leq c + \text{qd}(\phi_1)$, where qd denotes quantifier depth. Suppose that Player 2 wins the $(k + c)$ -round game in $(w, i_{succ}), (v, i_{succ})$. Then $(w, i_{succ}) \equiv_{k+c} (v, j_{succ})$. Consider the formula ϕ_1 that defines the \equiv_k -class of (w, i) . Then $(w, i_{succ}) \models next$, so $(v, j_{succ}) \models next$. Thus $(v, j) \models \phi_1$, so $(w, i) \equiv_k (v, j)$, and Player 2 wins the k -round game in these words.

We begin with Item (a): For the initial factorization, we simply take $start(x)$ to be $a(x)$: the factor starts are exactly the positions that contain a . We now assume that τ is some factorization scheme in the sequence, and that for the preceding factorization scheme σ , the required formula, which we denote $start_\sigma$, exists.

To establish this formulation, let $(w, i), (w', j)$ be as described. Since, by the inductive hypothesis, the formula $start_\sigma$ for the preceding scheme σ exists, we can treat this as if it is an atomic formula, in describing our game strategy. Observe that i must also be the start of a factor of w according to the previous factorization scheme σ . We write this as $start_\sigma(i)$ rather than the more verbose $(w, i) \models start_\sigma(x)$. If j does not satisfy $start_\sigma(j)$, then by induction we are done, and can take the number k of rounds to be the quantifier depth of $start_\sigma$. Thus j is the start of a factor with respect to the scheme σ , not with respect to τ . This can happen in one of two ways, depending on whether the most recent sub-step collected a subalphabet B , or capped a subalphabet B .

In the first case, we will describe a winning strategy for Player 1 in a game that lasts just a few more rounds than the game for the previous scheme. Position j was the start of a factor in the prior scheme σ , and has been collected into a larger factor that begins at position to the left of j . First suppose that i is the start of a factor with content different from B . Then this factor must contain some $c \notin B$. Player 1 then wins as follows: He moves right in (w', j) , jumping to the start j' of the next factor (which must satisfy $start_\sigma(j')$). In so doing, all the letters he jumps belong to B . Player 2 must also jump to the right in (w, i) , and must also land on the start of a factor in the scheme σ ; otherwise, by induction, Player 1 will win the game in the next k rounds. But to do so, Player 2 will have to jump over a position containing c , so she cannot legally make this move. Thus i must be the start of



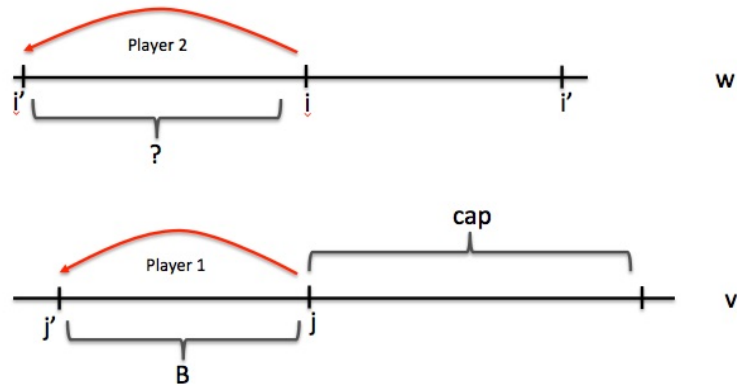
■ **Figure 2** Game-based proof of definability of factor starts. The figure shows the two words just after the step collecting the subalphabet B . We suppose i, j are factor starts for the preceding factorization scheme σ , and that i , but not j , is a factor start for the present scheme τ . This means that the factor with respect to σ beginning at j was joined to the previous factor as a result of the collection. If Player 1 moves to the start j' of the next factor of w' with respect to σ (blue arrow), then he jumps over precisely the letters of B . Thus for Player 2 to have a response, i must be the start of a factor with alphabet B . But this means that the factor with respect to σ in w that precedes i must contain a letter not in B . As a result, Player 2 cannot reply to a move by Player 1 to the start j'' of the factor with respect to σ that precedes j (red arrow).

a factor with content B . In this case, Player 1 moves left in (w', j) to j'' , the start of the previous factor with respect to σ . In doing so, he jumps over letters in B . Now Player 2 must also jump to the left in (w, i) to a position that was the start of a factor with respect to σ , but must jump over a letter not in B to do this, so Player 1 wins again. (See Figure 2.)

In the second case, where B was capped, j was the start of a factor that immediately followed a newly-collected factor with content B . Player 1 jumps left to j' , the start position of this factor, and in doing so jumps over a segment with content B . Thus Player 2 must jump to the start of a factor with respect to σ . For this to be a legal move, the segment she jumps must have content B . However, this is impossible, for any factor with this content in the scheme σ would have been capped by the following factor, so that i cannot be the start of a factor for τ . (Figure 3.)

Now for Item (b). Again, we use a game argument. We claim it will be enough to establish the following for sufficiently large values of k : Let $(w, i), (v, j)$ be marked words, where i, j are the starts of factors, and let $(w, i_{succ}), (v, i_{succ})$ be the same words, where the indices i_{succ}, j_{succ} mark the start of the successor factors. If Player 1 has a winning strategy in the k -round game in $(w, i), (v, j)$, then he has a winning strategy in the k' -round game in $(w, i_{succ}), (v, i_{succ})$ for some k' that depends only on k and the alphabet size, and not on v and w . Equivalently, if Player 2 wins in $(w, i_{succ}), (v, i_{succ})$ then she wins in $(w, i), (v, j)$. Of course, there is the analogous formulation for *previous*.

So we will suppose Player 1 has a winning strategy in the k -round game in $(w, i), (v, j)$, where k is at least as large as the quantifier depth of $start_\tau$. We will prove the existence of a strategy in $(w, i_{succ}), (v, j_{succ})$ for the k' -round game, where k' is larger than k . (By tracing through the various cases of the proof carefully, you can figure out how large k' needs to be.) What we will show in fact is that for each τ , Player 1 can force the starting configuration $(w, i_{succ}), (v, j_{succ})$ to the configuration $(w, i), (v, j)$, and from there apply his winning strategy in $(w, i), (v, j)$.



■ **Figure 3** This shows the case just after the step that caps the subalphabet B . Again suppose i, j are factor starts for the preceding factorization scheme σ , and that i , but not j , is a factor start for the present scheme τ . If Player 1 moves in v from j to j' , the start of the factor preceding j with respect to σ , then only letters in B are jumped. If Player 2 moves left from i to another factor start with respect to σ , she will have to jump over letters that are not in B , because all factors with alphabet B have been capped; thus Player 2 cannot respond to this move.

The base step is where τ is the initial factorization scheme. Here the factor starts are just the positions where the letter a occurs. Player 1 begins by jumping from i_{succ} to i . For Player 2 to respond correctly, she must jump from j_{succ} to j , because she is required to move left and land on a position containing a while jumping over a segment that does not contain the letter a .

So now we will suppose that τ is not the initial factorization scheme. We again denote the previous factorization scheme by σ . We assume that the property in Item (b) holds for σ . Thanks to what we proved above, we know that the property in Item (a) holds for both τ and σ . This means that we can treat $start_\tau$ and $start_\sigma$ essentially as atomic formulas.

If i_{succ} is also the successor of i (that is, the start of the next factor) with respect to the previous factorization scheme σ , and j_{succ} is the successor of j , then we have the desired result by induction. Thus we may suppose that one or both of the factor starts, either between i and i_{succ} or between j and j_{succ} , or both, were eliminated in the most recent sub-step of the algorithm.

Let us suppose first that the most recent sub-step was a collection step, collecting the subalphabet B . Player 1 jumps from i_{succ} left to i . The set of jumped letters is B . Player 2 must respond by jumping to some $j' < j_{succ}$ where j' satisfies $start_\tau$. If $j' < j$, then the set of jumped letters necessarily contains a letter not in B , so such a move is not legal. Thus $j' = j$. Player 1 now follows his winning strategy in $(w, i), (v, j)$. The identical strategy works for the predecessor version, because any factor following the sequence of collected factors must contain a letter not in B .

So suppose that the most recent sub-step was a capping step, and that the subalphabet B was capped. We may suppose that there is some i' with $i < i' < i_{succ}$ such that $start_\sigma(i')$, but not $start_\tau(i')$. Thus the interval from i to $i' - 1$ has content B and constitutes a factor that was collected during the prior sub-step, before being capped in the present one. Player 1 uses his strategy from the previous factorization scheme to force the configuration to $(w, i'), (v, j')$, where j' is the start of the factor preceding j in the scheme σ . Observe that we must have that j' does not satisfy $start_\tau$ because i' does not satisfy $start_\tau$. Thus $j < j' < j_{succ}$, so the interval from j to $j' - 1$ is also a factor with content B that was collected during the previous

substep. Player 1 now moves from i' left to i . Player 2 must respond with a move to $j'' \leq j$ such that $start_\tau(j'')$ holds. We cannot have $j'' < j$, for then the set of jumped letters would include a letter not in B . Thus $j'' = j$, and the game is now in the configuration $(w, i), (v, j)$.

The strategy for a capped step in the predecessor game uses the same idea: We may assume there is some i' with $i_{prec} < i' < i$ such that the interval from i_{prec} to $i' - 1$ has content B and constitutes a factor that was collected during the prior sub-step, before being capped in the present one. Thus in the previous scheme σ , i' was the successor position of i_{prec} . Player 1 uses his strategy from the previous scheme to force the game to the configuration $(w, i'), (v, j')$, where j' is the successor of j_{prec} in the scheme σ . We must have the set of jumped letters to be B in each case, so the intervals from i' to $i - 1$ and j' to $j - 1$ are the caps applied in the scheme τ , and thus i is the successor of i' , and j the successor of j' , in the scheme σ . Player 1 now uses his strategy for the scheme σ to force the game from the configuration $(w, i'), (v, j')$ to $(w, i), (v, j)$. ◀

5 Simulating factorization in logic

A factorization scheme σ gives a factorization $\sigma(w) = (w_1, \dots, w_k)$ of an a -word w . This in turn gives a word $\sigma_h(w) = m_1 \cdots m_k \in M^+$. We say that σ *admits simulations* if the following properties hold.

- For each sentence $\psi \in FO^2[<, Suc]$ over the alphabet M , there exists a sentence $\phi \in FO^2[<, bet]$ over the alphabet A with the following property. Let w be an a -word.

$$w \models \phi \quad \text{iff} \quad \sigma_h(w) \models \psi.$$

- For each formula $\psi(x) \in FO^2[<, Suc]$ with one free variable over the alphabet M , there exists a formula $\phi(x) \in FO^2[<, bet]$ with one free variable over the alphabet A with the following property. Let w be an a -word, $1 \leq i \leq k$ and let j_i be the position within w of the first letter of w_i in $\sigma(w)$. Then

$$(w, j_i) \models \phi(x) \quad \text{iff} \quad (\sigma_h(w), i) \models \psi(x).$$

► **Lemma 4 (Simulation).** *Each factorization scheme in our sequence admits simulations.*

It is useful to have abbreviations for commonly used subformulas of $FO^2[<, bet]$. If B is a subalphabet of A , we write $[B](x, y)$ to mean the conjunction of $\neg c(x, y)$ over all $c \notin B$; in other words, ‘every letter between x and y belongs to B ’. $[a](x, y)$ is always true if $y \leq x$ because $a(x, y)$ is false whenever $y \leq x$. We denote by $\llbracket B \rrbracket(x, y)$ the conjunction of $[B](x, y)$ together with the conjunction of $b(x, y)$ over all $b \in B$; in other words, B is exactly the set of letters between x and y .

Proof. The first claim in the Theorem follows easily from the second. So we will begin with the formula $\psi(x) \in FO^2[<, Suc]$ over M and show how to produce $\phi(x)$. We prove this by induction on the construction of formula ψ . So the base case is where $\psi(x)$ is an atomic formula $m(x)$, where $m \in M$. This means that for each factorization scheme σ , we have to produce a formula $\phi_{m,\sigma}(x)$ such that for an a -word w , $(w, i) \models \phi_{m,\sigma}(x)$ if and only if the factor starting at i maps to m under h .

We do this by induction on the sequence of factorization schemes. In the initial factorization, every factor is of the form au , where $a \notin \alpha(u)$. This factor maps to m if and only if $h(u) = m'$ for some $m' \in M$ satisfying $h(a) \cdot m' = m$. Since we suppose the main theorem holds for every alphabet strictly smaller than A , there is a sentence $\rho \in FO^2[<, bet]$ such that

$u \models \rho$ if and only if $h(u) = m'$ where $h(a) \cdot m' = m$. We now relativize ρ to obtain a formula ρ' with one free variable that is satisfied by (w, i) if and only if the factor of w starting at i has the form au , where $u \models \rho$. To do this, we do a standard relativization trick, working from the outermost quantifier of ρ inward. We can assume that all the quantifiers at the outermost level quantify the variable y . We replace each of these quantified formulas $\exists y \eta(y)$ by $\exists y (y > x \wedge \neg a(x, y) \wedge \eta(y))$. Similarly, as we work inward, we rewrite each occurrence of $\exists z' (z' > z \wedge \eta)$ and $\exists z' (z' < z \wedge \eta)$, where $\{z, z'\} = \{x, y\}$, by adding the clause $\neg a(z, z')$ or $\neg a(z', z)$. In essence, each time we jump left or right to a new position, we check that in so doing we did not jump over any occurrence of a , and thus remain inside the factor.

We now assume that τ is not the initial factorization scheme, and that the formula $\phi_{m, \sigma}(x)$ exists for the preceding factorization scheme σ . We first consider the case where τ was produced during a step that collected a subalphabet B . Observe that we can determine within a formula whether i is the start of a factor that was produced during this collection step, with the criterion

$$\exists y (x < y \wedge \text{start}_\tau(y) \wedge \llbracket B \rrbracket(x, y)).$$

(This includes the case where the collection is trivial because there is only one factor to collect.) If this condition does not hold, then we can test whether the factor maps to m with the formula produced during the preceding step. So we suppose that i is the first position of one of the new ‘collected’ factors. Since $B \subsetneq A$, there is a sentence ρ of $FO^2[<, \text{bet}]$ satisfied by exactly the words over this smaller alphabet that map to m . Once again, we must relativize ρ to make sure that whenever we introduce a new quantifier $\exists x (y > x \wedge \dots)$ or $\exists x (y < x \wedge \dots)$ we do not jump to a position outside the factor. To do this, we can replace $\exists x (y > x \wedge \dots)$ by

$$\exists x (y > x \wedge \llbracket B \rrbracket(x, y) \wedge \exists x (y < x \wedge \text{start}_\tau(x) \wedge \llbracket B \rrbracket(x, y))).$$

In other words, we did not jump over any letter not in $\{a\} \cup B$, and there is a factor start farther to the right that we can reach without jumping over any letter not in B . We replace $\exists x (y < x \wedge \dots)$ by

$$\exists x (y < x \wedge \llbracket B \rrbracket(y, x) \wedge \exists x (x \leq y \wedge \text{start}_\tau(x) \wedge \llbracket B \rrbracket(x, y))),$$

using essentially the same idea.

Now suppose that τ was produced during a step that capped the subalphabet B . Again, we can write a formula that says that i is the start of a new factor produced in this process: it is exactly the formula that said i was the start of a factor that collected B in the preceding scheme σ . So we only need to produce a formula that says the factor of w beginning at i maps to m under the assumption that this is one of the new ‘capped’ factors. Our factor has the form $u_1 u_2$, where u_2 is the cap and u_1 is the factor in which B was collected. We consider all pairs m_1, m_2 such that $m_1 \cdot m_2 = m$. We know that there are formulas $\rho_1(x)$ and $\rho_2(x)$ telling us that the factors in the preceding scheme σ map to m_1 and m_2 . We use the same formula $\rho_1(x)$, and take its conjunction with $\text{next}(x)$, the successor formula derived from $\rho_2(x)$ by means of Item (b) in Lemma 3. We are using the fact that the start of u_2 is the successor of the start of u_1 under the preceding scheme σ .

We are almost done (and we no longer need to induct on the sequence of factorization schemes) because $FO^2[<, \text{Suc}]$ formulas can be reduced to a few normal forms [5]. Let us first suppose that our formula ψ has the form $\exists x (\text{Suc}(x, y) \wedge \kappa(x))$. The inductive hypothesis

is that there is a formula μ simulating κ . Let *previous* be the predecessor formula whose existence is given by Item (b) of Lemma 3. We claim that *previous* simulates ψ . To see this, suppose w is an a -word, and j_i is the position where the i^{th} factor of w begins.

Suppose $(w, j_i) \models \textit{previous}$. Then $(w, j_{i+1}) \models \mu$.

So $(\sigma_h(w), i+1) \models \kappa$, which gives $(\sigma_h(w), i) \models \psi$.

This implication also runs in reverse, so we have shown that *previous* simulates ψ . Using the successor formula in place of the predecessor formula gives us the analogous result for ψ in the form $\exists x(\textit{Suc}(y, x) \wedge \kappa(x))$. \blacktriangleleft

6 Proof of the main lemma

Proof of Lemma 2. Again, we assume $|A| > 1$ and that the theorem holds for all strictly smaller alphabets. Let $m \in M$, where M satisfies the $\mathbf{M}_e\mathbf{DA}$ property. We need to show $h^{-1}(m)$ is defined by a sentence of $FO^2[<, \mathbf{bet}]$. As an overview, we will first, through a series of quite elementary steps, reduce this to the problem of showing that for each $a \in A$ and $s \in M$, the set of a -words mapping to s is defined by a sentence of $FO^2[<, \mathbf{bet}]$. We then use Lemma 4 on simulations, together with the identity $\mathbf{LDA} = \mathbf{DA} * \mathbf{D} [1]$ to find a defining sentence for the set of a -words that map to s .

First note that $h^{-1}(m) = \bigcup_{B \subseteq A} \{w \in h^{-1}(m) : \alpha(w) = B\}$.

It thus suffices to find, for each subalphabet B , a sentence ψ_B of $FO^2[<, \mathbf{bet}]$ defining the set of words $\{w \in h^{-1}(m) : \alpha(w) = B\}$. We then obtain a sentence for $h^{-1}(m)$ as

$$\bigvee_{B \subseteq A} (\psi_B \wedge \bigwedge_{b \in B} \exists x b(x) \wedge \bigwedge_{b \notin B} \neg \exists x b(x)).$$

Since we obtain the sentences ψ_B for proper subalphabets B of A by the induction hypothesis, we only need to find ψ_A .

For each w with $\alpha(w) = A$, let $\textit{last}(w)$ be the last letter of w to appear in a right-to-left scan of w . It will be enough to find, for each $a \in A$, a sentence ϕ_a of $FO^2[<, \mathbf{bet}]$ defining $\{w \in h^{-1}(m) : \textit{last}(w) = a\}$, since we then get ψ_A as

$$\exists y(a(y) \wedge \forall x(x > y \rightarrow \neg a(x))) \wedge \bigwedge_{b \neq a} \exists x(x > y \wedge b(x)) \wedge \phi_a.$$

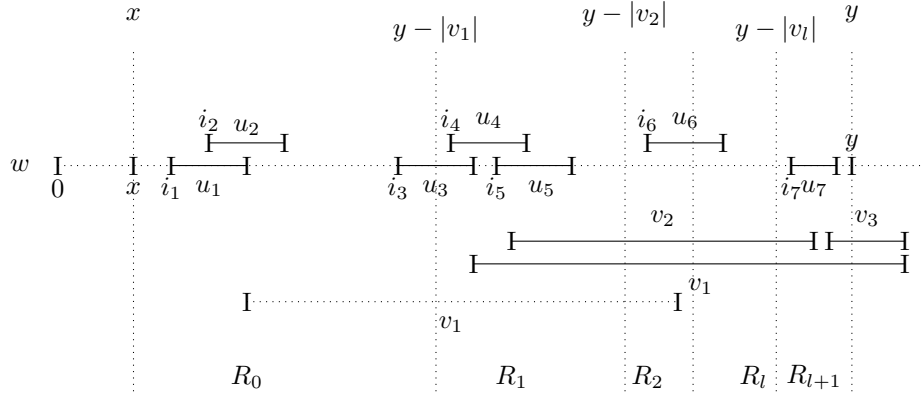
A word w with $\alpha(w) = A$ and $\textit{last}(w) = a$ has a unique factorization $w = uv$, where $\alpha(u) = A \setminus \{a\}$, and v is an a -word. We consider all factorizations $m = m_1 m_2$ in M . By the inductive hypothesis, there is a sentence μ of $FO^2[<, \mathbf{bet}]$ defining the set of all words over $A \setminus \{a\}$ that map to m_1 . Suppose that we are able to find a sentence ν defining the set of all a -words mapping to m_2 . We can then use a simple relativizing trick to obtain a sentence defining all concatenations uv such that $u \models \mu$ and $v \models \nu$. One simply modifies each quantified subformula $\exists x \zeta$ of μ and ν , starting from the outside, changing them to

$$\exists x(\neg \exists y(y \leq x \wedge a(y))) \text{ and } \exists x(\exists y(y \leq x \wedge a(y))).$$

The conjunction of the two modified sentences now says that μ holds in the factor preceding the first occurrence of a , and ν holds in the factor that begins at the first occurrence of a . Take the disjunction of these conjunctions over all factorizations $m_1 m_2$ of m to obtain ϕ_a .

It remains to show how to construct a sentence that defines the set of a -words that map to a given element s of M . Let $w \in A^*$ be an a -word. Let σ be the final factorization scheme in our sequence, so that

$$\sigma(w) = (w_1, \dots, w_k), \quad \sigma_h(w) = m_1 \cdots m_k \in M^+.$$



■ **Figure 4** Occurrence sequence for model w, x : (1) Region R_1 starts at position $y - |v_1|$ where v_1 is the longest negative requirement. This means any negative factor v_i which starts in region R_0 will finish before y . Similarly, any negative factor v_i other than v_1 starting in R_2 will end before y . On the other hand, any v_1 starting after R_0 will necessarily end after y . (2) Positive requirements start in order $u_1 < u_2 \dots < u_7$. Moreover, u_1, u_2, u_3 start in R_0 , words u_4, u_5 start in R_1 and u_6 starts in R_2 . Finally, u_7 starts in R_{l+1} .

In fact, each w_i can be mapped to the subalphabet

$$N = \{h(v) \in M : \alpha(v) = A, v \in aA^*\},$$

so we can restrict to this subalphabet N of M .

The map $n \mapsto n$ extends to a homomorphism from N^+ into the subsemigroup S of M generated by the elements of N . Since the generators of S are images of words v with $\alpha(v) = A$, we have $eSe \subseteq eM_e e$, which is in \mathbf{DA} for every idempotent $e \in E(S)$ by definition of $\mathbf{M}_e \mathbf{DA}$. Locality of \mathbf{DA} means that having all eSe in \mathbf{DA} , the semigroup S is in $\mathbf{DA}^* \mathbf{D}$. Thus the set of words over N multiplying to $s \in S$ is defined by a sentence ψ over N in $FO^2[<, \text{succ}]$ [20]. We can take the conjunction of this with a sentence that says every letter belongs to the alphabet N , and thus obtain a sentence ψ' over M , also in $FO^2[<, \text{Suc}]$, defining this same set of words. Thus by the Simulation Lemma 4, there is a sentence ϕ in $FO^2[<, \text{bet}]$ that defines the set of a -words that map to s . This completes the proof. \blacktriangleleft

7 A logic for intermediate occurrences of factors

As an extension of the techniques we developed, we add to two-variable logic ‘betweenness’ predicates $\langle u \rangle(x, y)$ for $u \in A^+$. If $u = a_1 \dots a_n$, then

$$\langle u \rangle(x, y) = \exists z_1 \dots \exists z_n (x < z_1 < \dots < z_n < y \wedge \text{Suc}(z_1, z_2) \wedge \dots \wedge \text{Suc}(z_{n-1}, z_n) \wedge a_1(z_1) \wedge \dots \wedge a_n(z_n)).$$

We call the logic $FO^2[<, \text{betfac}]$. Its increased expressiveness does not translate to computational difficulty, which we will show by translation to temporal logic LTL [6]. For convenience, for $u = a_1 u_2 \dots a_n$, we will abbreviate by u the LTL formula $a_1 \wedge X(a_2 \wedge \dots \wedge X a_n)$.

► **Theorem 5.** *Satisfiability of $FO^2[<, \text{betfac}]$ is Expspace-complete.*

Proof. In [8] we gave an *Expspace* lower bound for $FO^2[<, \text{bet}]$, so we only have to give an *Expspace* upper bound. We give an exponential translation from an $FO^2[<, \text{betfac}]$ sentence to temporal logic LTL , whose satisfiability is decidable in *Pspace* [16].

For a fixed betweenness predicate mentioning x and y in a $FO^2[<, \text{betfac}]$ sentence, consider all such predicates within the same scope, because they refer to the same x and y points. They may specify existence or non-existence requirements. Existence of a factor uvw implies the existence of a factor v and conversely for non-existence, we discard such implied requirements.

As an example of the interaction of these requirements, consider the positive requirements $a(x, y)$ and $b(x, y)$ and the negative requirement $\neg \text{cacbc}(x, y)$ on the word ccccacbc where $x = 1$ and $y = 9$ are the first and last positions. All three requirements are satisfied, because the factor cacbc is not present *strictly* between x and y . Order the negative requirements by length, without loss of generality we have $|v_1| > \dots > |v_l|$ for negative requirements $\neg v_1(x, y), \dots, \neg v_l(x, y)$. All these must be satisfied at the positions from $x + 1$ to $y - |v_1|$, all except $\neg v_1$ at positions from there upto $y - |v_2|$, and so on. We can express this by the formula Neg below:

$$(\neg v_1 \wedge \dots \wedge \neg v_l) \mathbf{U}(\mathbf{X}^{|v_1|-1} y \wedge (\neg v_2 \wedge \dots \wedge \neg v_l) \mathbf{U}_{|v_1|-|v_2|} \mathbf{X}^{|v_2|-1} y \wedge \dots ((\neg v_l) \mathbf{U}_{|v_{l-1}|-|v_l|} \mathbf{X}^{|v_l|-1} y) \dots),$$

where the *bounded untils* are defined by $p\mathbf{U}_i q = p \wedge \mathbf{X}(p\mathbf{U}_{i-1} q)$ and $p\mathbf{U}_0 q = q$. The subformulae $\mathbf{X}^{|v_1|-1} y, \mathbf{X}^{|v_2|-1} y, \dots, \mathbf{X}^{|v_{l-1}|-1} y$ in Neg are redundant since they follow from the last $\mathbf{X}^{|v_l|-1} y$ and the durations of the *bounded untils*. We will develop this idea below.

Neg is not quite an *LTTL* formula since y is a first-order variable. Abbreviate by N the formula $(\neg v_1 \wedge \dots \wedge \neg v_l)$ to the left of the first *until* in Neg . We can write Neg more properly as $Neg(Q(y)) = N\mathbf{U}(Q(y))$ where we will replace $Q(y)$ later by a temporal formula.

There are also the positive requirements to satisfy. We take a disjunction over the possible orderings of positions where they are satisfied for the first time, which we abbreviate specifying where in three intervals $(x, y - |v_1|]$, $(y - |v_1|, y - |v_l|]$, $(y - |v_l|, y)$ they are to be placed. (The first two intervals are left-open and right-closed.) It follows from the fact that we have no implied factors that if the starting point of a factor is before the starting point of another, its ending point also precedes the ending point of the other.

$$O = u_1(x, y - |v_1|] < \dots < u_k(x, y - |v_1|] < u_{k+1}(y - |v_1|, y - |v_l|] < \dots < u_{k+j}(y - |v_1|, y - |v_l|] < u_{k+j+1}(y - |v_l|, y - |u_{k+j+i}|] < \dots < u_{k+j+i}(y - |v_l|, y - |u_{k+j+i}|].$$

More precisely there are $l + 1$ intervals to consider, by dividing up the middle interval $(y - |v_1|, y - |v_l|]$ into $l - 1$ subintervals as was done in formula Neg above.

The formula

$$Pos_0 = \neg u_1 \mathbf{U}(u_1 \wedge (\neg u_2 \mathbf{U}(u_2 \dots \wedge (\neg u_k \mathbf{U}(u_k \wedge (\text{true} \mathbf{U} \mathbf{X}^{|u_k-1|} y) \dots))))$$

takes care of the first block of requirements. This has to be interleaved to the left of the first *until* in Neg . That is, $Neg(Q(y)) = N\mathbf{U}(Q(y))$ is replaced by

$$Pos'_0(Q(y)) = (\neg u_1 \wedge N) \mathbf{U}(u_1 \wedge N \wedge ((\neg u_2 \wedge N) \mathbf{U}(u_2 \wedge N \wedge \dots \wedge ((\neg u_k \wedge N) \mathbf{U}(u_k \wedge (N\mathbf{U}(Q(y)))))) \dots))).$$

Similarly the next j requirements have to be divided and interleaved with the *bounded untils* in the middle intervals in Neg , specified by formulae Pos_1, \dots, Pos_{l-1} in much the same manner, and the last i requirements specified by formula Pos_l , have to be interleaved with the last $|v_l| - 1$ *nexts* in Neg and updated to $Pos'_1(Q(y)), \dots, Pos'_{l-1}(Q(y)), Pos'_l(Q(y))$ to form:

$$Neg' = Pos'_0(Pos'_1(\dots (Pos'_{l-1}(Pos'_l(\mathbf{X}^{\min(|v_l|, |u_{k+j+i}|)-1} y)) \dots))).$$

The outcome of this interleaving procedure is that we have a formula having a *single* occurrence of the non-temporal variable y at the end. The size of this formula, for one ordering O , is polynomial in the size of the between requirements. The number of possible orderings O is exponential in the number of between requirements, $l + k + j + i$ above.

The technique of Etessami, Vardi and Wilke allows replacing the point y using its *type* [5], which produces an *LTL* formula. As argued by them, the complete *LTL* formula produced is exponential in terms of the sentence we started with. The exponentially many disjunctions produced by different orderings above compose with their procedure to give an exponential-sized formula. ◀

8 Characterization of $FO^2[<, \text{betfac}]$

The class of languages definable in the logic $FO^2[<, \text{betfac}]$ corresponds to a variety of finite semigroups rather than monoids. An operation which can be lifted to the level of semigroup and monoid classes is the semidirect product (which is not effective in general). We have obtained an *effective* algebraic characterization of $FO^2[<, \text{betfac}]$. Presenting the proof will require a detour into the algebraic theory of finite categories, so we will restrict ourselves here with the statement and the algebraic characterization, and reserve the proof of effectiveness for the full version of the paper.

► **Theorem 6** ($FO^2[<, \text{betfac}]$ characterizes $\mathbf{M}_e\mathbf{DA} * \mathbf{D}$). *Let $L \subseteq A^+$. L is definable in $FO^2[<, \text{betfac}]$ if and only if $S(L) \in \mathbf{M}_e\mathbf{DA} * \mathbf{D}$. Moreover, there is an effective procedure for determining if $S(L) \in \mathbf{M}_e\mathbf{DA} * \mathbf{D}$.*

Since $\mathbf{M}_e\mathbf{DA}$ contains $\Delta_3[<]$ in the quantifier alternation hierarchy [22], $\mathbf{M}_e\mathbf{DA} * \mathbf{D}$ contains $\Delta_3[<, \text{Suc}]$, which includes the language $BB_2 = (a(ab)^*b)^+$ which we showed in [8] was not in $\mathbf{M}_e\mathbf{DA}$. On the other hand it does not contain $BB_3 = (a(a(ab)^*b)^*b)^+$. Consider the language U_3 which is a sublanguage of $A^*c(a+b)^*cA^*$ such that between the marked c 's, the factor bb does not occur before the factor aa . This is in $\mathbf{M}_e\mathbf{DA} * \mathbf{D}$ since it is defined by the $\Pi_2[<, \text{Suc}]$ sentence

$$\begin{aligned} \forall x \forall y \forall z \forall z' (& c(x) \wedge c(y) \wedge x < z < z' < y \wedge \text{Suc}(z, z') \wedge b(z) \wedge b(z') \\ & \rightarrow \exists w \exists w' (x < w < w' < z \wedge \text{Suc}(w, w') \wedge a(w) \wedge a(w'))). \end{aligned}$$

The proof of the theorem, in both directions, depends on the characterization of $\mathbf{V} * \mathbf{D}$ in terms of \mathbf{V} [17]. This can be stated in several different ways, but all depend on some scheme for treating words of length k over A as individual letters. Here is a standard version. Let $k > 0$. Let A be a finite alphabet, and let $B = A^k$. We treat B as a finite alphabet itself – to distinguish the *word* $w \in A^*$ of length k from the same object considered as a *letter* of B , we write $\{w\}$ in the latter case. We will define, for a word $w \in A^+$ with $|w| \geq k - 1$, a new word $w' \in B^*$, where w' is simply the sequence of length- k factors of w . So, for example, with $A = \{a, b\}$ and $k = 3$, if $w = aa$, then $w' = 1 \in B$, while if $w = ababba$, then

$$w' = \{aba\}\{bab\}\{abb\}\{bba\}.$$

To make sure that the lengths match up, we supplement A with a new symbol $*$ and define B' as $(A \cup \{*\})^k$, and w'' as the sequence of length- k factors of $*^{k-1}w$. For example, with this new definition, if $k = 3$ and $w = ababba$, then

$$w'' = \{**a\}\{*ab\}\{aba\}\{bab\}\{abb\}\{bba\}.$$

► **Theorem 7** (characterization of $\mathbf{V} * \mathbf{D}$ [17]). *Let $h : A^+ \rightarrow S$ be a homomorphism onto a finite semigroup. $S \in \mathbf{V} * \mathbf{D}$ if and only if there exist: an integer $k > 1$, and a homomorphism $h' : B^* \rightarrow M \in \mathbf{V}$, where $B = A^k$, such that whenever $v, w \in A^+$ are words that have the same prefix of length $k - 1$, and the same suffix of length $k - 1$, and v', w' are the sequence of k -length factors of v, w respectively, with $h'(v') = h'(w')$, then $h(v) = h(w)$.*

In brief, you can determine $h(w)$ by looking at the prefix and suffix of w of length $k - 1$, and checking the value of w' under a homomorphism h' into an element of \mathbf{V} . Note that the statement is false if \mathbf{V} is the trivial variety (and only in this case), but we can correct by replacing \mathbf{D} in the statement by **LI**.

In the full version of the paper we will show:

► **Proposition 8** (Delay). *Let ϕ be a sentence of $FO^2[<, \mathbf{betfac}]$. Then there exist $k > 1$ and a sentence ϕ' of $FO^2[<, \mathbf{bet}]$ interpreted over $(A \cup \{*\})^k$, with this property: if $w \in A^+$ with $|w| \geq k - 1$, then $w \models \phi$ if and only if $w'' \models \phi'$.*

► **Proposition 9** (Expansion). *Let ϕ' be a sentence of $FO^2[<, \mathbf{bet}]$ interpreted over $(A \cup \{*\})^k$, where $k > 1$. Then there is a sentence ϕ of $FO^2[<, \mathbf{betfac}]$ with this property: if $w \in A^+$ with $|w| \geq k - 1$, then $w \models \phi$ if and only if $w'' \models \phi'$.*

Proof of Characterization Theorem 6. Let $L \subseteq A^+$, and suppose that L is definable by a sentence ϕ of $FO^2[<, \mathbf{betfac}]$. Let $k > 1$ and ϕ' in $FO^2[<, \mathbf{bet}]$ be as given by Proposition 8. Let $L' \subseteq ((A \cup \{*\})^k)^*$ be the language defined by ϕ' . We will show that $S(L) \in \mathbf{M}_e \mathbf{DA} * \mathbf{D}$.

Let $h : A^+ \rightarrow S(L)$ be the syntactic morphism of L . Let h' be the syntactic morphism of L' and let h'' be the restriction of h' to elements of $(A^k)^*$. Since ϕ' is a sentence of $FO^2[<, \mathbf{bet}]$, the syntactic monoid of L' , and hence the image of h'' , belongs to $\mathbf{M}_e \mathbf{DA}$. It is therefore enough, in view of Theorem 7, to suppose that $v, w \in A^+$ have the same prefix of length $k - 1$ and the same suffix of length $k - 1$, and that $h''(v') = h''(w')$, and then conclude that $h(v) = h(w)$. To show $h(v) = h(w)$ we must show that for any $x, y \in A^*$, $xvy \in L$ if and only if $xwy \in L$. Given the symmetric nature of the statement, it is enough to show $xvy \in L$ implies $xwy \in L$. So let $xvy \in L$. Then $xvy \models \phi$, so $(xvy)'' \models \phi'$. We take apart $(xvy)''$: Suppose $x = a_1 \cdots a_r, v = b_1 \cdots b_s, y = c_1 \cdots c_t$.

The leftmost $r + k - 1$ letters of $(xvy)''$ are

$$\{ *^{k-1} a_1 \} \{ *^{k-2} a_1 a_2 \} \cdots \{ a_r b_1 \cdots b_{k-1} \}.$$

The rightmost t letters of $(xvy)''$ are

$$\{ b_{s-k+2} \cdots b_s c_1 \} \{ b_{s-k+3} \cdots b_s c_1 c_2 \} \cdots \{ c_{t-k+1} \cdots c_t \}.$$

(The exact form of the last factor will be different if $t < k - 1$.) In between these two factors, we have the $s - k + 1$ letters of v' . Thus $h'((xvy)') = m_1 h''(v') m_2$, where m_1, m_2 depend only on x, y and the prefix and suffix of v of length at most $k - 1$. It follows that we likewise have $h'((xwy)') = m_1 h''(w') m_2$, with the same m_1, m_2 . Since $h''(v') = h''(w')$, we conclude $h'((xvy)') = h'((xwy)')$, so $(xwy)'' \models \phi'$. Thus $xwy \models \phi$, and so $xwy \in L$. This concludes the proof that $S(L) \in \mathbf{M}_e \mathbf{DA} * \mathbf{D}$.

Conversely, suppose $L \subseteq A^+$ and that $S(L) \in \mathbf{M}_e \mathbf{DA} * \mathbf{D}$. Let $h : A^+ \rightarrow S(L)$ be the syntactic morphism of L . Let $h' : (A^k)^* \rightarrow M \in \mathbf{M}_e \mathbf{DA}$ be the homomorphism given by Theorem 7. We extend h' to $((A \cup \{*\})^k)^*$ by defining $h'(b) = 1$ for any b that contains the new symbol $*$. Then for each $m \in M$, we have a sentence ϕ'_m of $FO^2[<, \mathbf{bet}]$ interpreted over $((A \cup \{*\})^k)^*$ defining $(h')^{-1}(m)$. Let ϕ_m be the sentence over $FO^2[<, \mathbf{betfac}]$ given

by Proposition 9. For each $x \in A^{k-1}$, let pref_x be a sentence defining the set of strings over A whose prefix of length $k-1$ is x , and similarly define suff_x . Observe that both of these sentences can be chosen to be in $FO^2[<, \text{betfac}]$. In fact, these properties are definable in $FO^2[<, \text{bet}]$ over A . It follows that the set of words in A^+ of length at least $k-1$ mapping to a given value s of $S(L)$ is given by a disjunction of finitely many sentences of the form

$$\text{pref}_x \wedge \text{suff}_y \wedge \phi'_m.$$

We thus get the complete preimage $h^{-1}(s)$ by taking the disjunction with a sentence that says the word lies in a particular finite set. So L itself is definable in $FO^2[<, \text{betfac}]$. ◀

References

- 1 Jorge Almeida. A syntactical proof of the locality of DA. *Int. J. Alg. Comput.*, 6:165–177, 1996.
- 2 Janusz Brzozowski. A generalization of finiteness. *Semigr. Forum*, 13:239–251, 1977.
- 3 Rina Cohen and Janusz Brzozowski. Dot-depth of star-free events. *J. Comput. Syst. Sci.*, 5(1):1–16, 1971.
- 4 Volker Diekert, Paul Gastin, and Manfred Kufleitner. First-order logic over finite words. *Int. J. Found. Comp. Sci.*, 19:513–548, 2008.
- 5 Kousha Etessami, Moshe Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.
- 6 Johan Anthony Willem Kamp. Tense logic and the theory of linear order. UCLA, 1968. PhD thesis.
- 7 Robert Knast. A semigroup characterization of dot-depth one languages. *Inf. Theor. Appl.*, 17(4):321–330, 1983.
- 8 Andreas Krebs, Kamal Lodaya, Paritosh Pandya, and Howard Straubing. Two-variable logic with a between relation. In Martin Grohe, Erik Koskinen, and Natarajan Shankar, editors, *Proc. 31st LICS, New York*, pages 106–115. ACM/IEEE, 2016.
- 9 Robert McNaughton and Seymour Papert. *Counter-free automata*. MIT Press, 1971.
- 10 Jean-Éric Pin. *Varieties of formal languages*. Plenum, 1986.
- 11 Thomas Place and Luc Segoufin. Decidable characterization of $fo^2(<, +1)$ and locality of DA. Preprint, ENS Cachan, 2014.
- 12 Thomas Place and Marc Zeitoun. Going higher in the first-order quantifier alternation hierarchy on words. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Proc. 41st Icalp, Part 2, Copenhagen*, volume 8573 of *LNCS*, pages 342–353, 2014.
- 13 Thomas Place and Marc Zeitoun. Separation and the successor relation. In Ernst W. Mayr and Nicolas Ollinger, editors, *Proc. 32nd Stacs, Garching*, volume 30 of *Lipics*, pages 662–675, 2015.
- 14 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Contr.*, 8:190–194, 1965.
- 15 Marcel-Paul Schützenberger. Sur le produit de concaténation non ambigu. *Semigr. Forum*, 13:47–75, 1976.
- 16 A. Prasad Sistla and Edmund Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.
- 17 Howard Straubing. Finite semigroup varieties of the form V^*D . *J. Pure Appl. Alg.*, 36:53–94, 1985.
- 18 Howard Straubing. *Finite automata, formal languages, and circuit complexity*. Birkhäuser, 1994.

- 19 Pascal Tesson and Denis Thérien. Logic meets algebra: the case of regular languages. *Log. Meth. Comp. Sci.*, 3(1:4):1–37, 2007.
- 20 Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In Jeffrey Vitter, editor, *Proc. 30th STOC, Dallas*, pages 234–240. ACM, 1998.
- 21 Bret Tilson. Categories as algebra. *J. Pure Appl. Alg.*, 48:83–198, 1987.
- 22 Pascal Weil. Some results on the dot-depth hierarchy. *Semigr. Forum*, 46:352–370, 1993.

Basic Operational Preorders for Algebraic Effects in General, and for Combined Probability and Nondeterminism in Particular

Aliaume Lopez

École Normale Supérieure Paris-Saclay
Université Paris-Saclay, France
aliaume.lopez@ens-paris-saclay.fr

Alex Simpson

Faculty of Mathematics and Physics
University of Ljubljana, Slovenia
Alex.Simpson@fmf.uni-lj.si

Abstract

The “generic operational metatheory” of Johann, Simpson and Voigtländer (LiCS 2010) defines contextual equivalence, in the presence of algebraic effects, in terms of a *basic operational preorder* on ground-type effect trees. We propose three general approaches to specifying such preorders: (i) operational (ii) denotational, and (iii) axiomatic; coinciding with the three major styles of program semantics. We illustrate these via a nontrivial case study: the combination of probabilistic choice with nondeterminism, for which we show that natural instantiations of the three specification methods (operational in terms of Markov decision processes, denotational using a powerdomain, and axiomatic) all determine the same canonical preorder. We do this in the case of both angelic and demonic nondeterminism.

2012 ACM Subject Classification Theory of computation → Operational semantics, Theory of computation → Denotational semantics, Theory of computation → Axiomatic semantics

Keywords and phrases contextual equivalence, algebraic effects, operational semantics, domain theory, nondeterminism, probabilistic choice, Markov decision process

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.29

Acknowledgements We thank Gordon Plotkin, Matija Pretnar and Niels Voorneveld for helpful discussions.

1 Introduction

Contextual equivalence, in the style of Morris, is a powerful and general method for defining program equivalence, applicable to many programming languages. Two programs are said to be contextually equivalent if they ‘behave’ equivalently when embedded in any suitable context that leads to ‘observable’ behaviour. More generally,¹ one can define *contextual preorder* in the same manner. Let P_1 and P_2 be comparable programs (for example, in a typed language, P_1 and P_2 would have the same type in order to be comparable). Suppose further that we have some basic preorder \preceq , defined on ‘observable’ computations, according to appropriate behavioural considerations. Then the contextual preorder is defined by

$$P_1 \sqsubseteq_{\text{ctxt}} P_2 \iff \text{for all observation contexts } C[-], C[P_1] \preceq C[P_2] . \quad (1)$$

¹ It is more general, since every equivalence relation is a preorder.



This method of definition has important consequences. For example, the relation $\sqsubseteq_{\text{ctxt}}$ is guaranteed to be a precongruence with respect to the constructors of the programming language. However, the quantification over contexts makes the definition awkward to work with directly. So various more manageable techniques for reasoning about contextual preorder relations have been developed, including: (bi)simulations and their refinements (applicative/environmental bisimulations, bisimulations up-to), denotational interpretation in domains, game semantics, program logics, and logical relations. These techniques are all reasonably general, in the sense that they adapt to different styles of programming languages, and combinations of programming features. Nonetheless, they are usually studied on a language-by-language basis.

One direction for the systematisation of a range of programming features has been provided by Plotkin and Power through their work on *algebraic effects* [13, 14]. Broadly speaking, effects are interactions between a program and its environment (including the machine state), and include features such as error raising, global/local state, input/output, nondeterminism and probabilistic choice. Plotkin and Power realised that the majority of effects (including all the aforementioned ones) are *algebraic*, in the sense that the operations that trigger them satisfy a certain natural behavioural constraint.²

The algebraic effects in a programming language can be supplied via an algebraic signature Σ of effect-triggering operations, and the operational semantics of the language can then be defined parametrically in Σ . This is achieved by effectively splitting the semantics of the language into two steps. In the first step, operational rules specify how any program P evaluates to an associated *effect tree* $|P|$, which documents all the effects that might potentially occur during execution. In an effect tree, the effects themselves are uninterpreted, in the sense that no specific execution behaviour is imposed upon them. As the second step, an interpretation is given to effect trees, by one means or another, from which a semantics for the whole language is extrapolated. This methodology was first followed in [13], where the operational reduction to effect trees (there called *infinitary effect values*) is used as a method for proving the computational adequacy of denotational semantics. In [6], effect trees (there called *computation trees*) are used to give a uniform definition of contextual preorder, and to characterise it as a logical relation. Effect trees also allow a general definition of applicative (bi)similarity for effects [17] (see [1] for a related approach not based on trees).

In this paper, as in [6], our aim is to exploit the notion of effect tree for the purpose of giving a unified theory of contextual preorders for programming languages with algebraic effects. In [6], this was carried out in the context of a specific polymorphically-typed call-by-name functional language with general recursion, to which algebraic effects were added. In this paper, we build on the technical work of [6], but an important departure is that we detach the development from any fixed choice of background programming language. This is based on the following general considerations. In order to define contextual preorder via (1) above, one needs to specify what constitutes an observation context, and also the basic behavioural relation \preceq on the computations such contexts induce. In the case of a language with algebraic effects, we can observe two things about a computation. Firstly, we can observe any discrete return value. In any sufficiently expressive language, discrete values should be convertible to natural numbers. So it is a not unreasonable restriction to restrict observation contexts to *ground contexts* whose return values (if any) are natural numbers. Secondly, we can also potentially observe aspects of effectful behaviour of such computations,

² In operational terms, the constraint is that the behaviour of the operation does not depend on the content of the continuation at the time the operation is triggered.

with exactly what is observable very much depending on the effects in question. One general approach to taking such effectful behaviour into account is to specify a *basic operational preorder* \preceq on the set of effect trees with natural-number-labelled leaves, which implements a desired behavioural preorder on effectful computations with return values in \mathbb{N} . We are thus led to the following general formulation of contextual preorder. Given a chosen basic operational preorder \preceq , we define the induced contextual preorder on programs by:

$$P_1 \sqsubseteq_{\text{ctxt}} P_2 \iff \text{for all ground contexts } C[-], |C[P_1]| \preceq |C[P_2]| . \quad (2)$$

In [6], this general approach was developed in detail for a polymorphically typed call-by-name functional language with algebraic effects. The main result was that the resulting contextual preorder, defined by (2), is well behaved if the basic operational preorder satisfies two technical properties, *admissibility* and *compositionality*. In particular, it follows from these conditions that the contextual preorder is characterisable as a logical relation (and hence amenable to an important proof technique), and also that, on ground type programs P_1, P_2 , the contextual and basic operational preorders coincide (i.e., $P_1 \sqsubseteq_{\text{ctxt}} P_2$ if and only if $|P_1| \preceq |P_2|$). Recently, we have carried out a similar programme for a call-by-value language, similar to the language in [13], and obtained analogous results.³ It seems likely that similar results hold for other language variants.

The notion of admissible and compositional basic operational preorder thus provides a uniform and well-behaved definition of contextual preorder, for different languages with algebraic effects. Furthermore, as is argued in [6, §V], it can also be given an intrinsic, more conceptually motivated justification in terms of an explicit notion of *observation*. Our general position is that the notion of admissible and compositional basic operational preorder is a fundamental one. *For any given combination of algebraic effects, one need only define a corresponding admissible and compositional basic operational preorder.* Once this has been done, one obtains, via (2), a definition of contextual preorder that can be applied to many programming languages containing those effects, and which will enjoy good properties.

In this paper, we describe three different approaches to defining basic operational preorders. The first is an *operational* approach. One explicitly models the execution of the effects in question, and uses this model to determine the preorder. This is the approach that was followed in [6]. Under this approach, admissibility and compositionality do not hold automatically, and so need to be explicitly verified. The second is a *denotational* approach. One builds a suitable domain-based model of the relevant effect operations. This induces a basic operational preorder on effect trees that is automatically admissible and compositional. The third is *axiomatic*. One finds a set of (possibly infinitary) Horn-clause axioms asserting desired properties of the intended preorder. The basic operational preorder is then taken to be the smallest admissible preorder satisfying the axioms. In addition to being admissible by definition, the resulting preorder is automatically compositional.

It will not have escaped the readers attention that our three approaches to defining preorders parallel the three main styles of program semantics: *operational*, *denotational* and *axiomatic*. Nonetheless, irrespective of how they are defined, we view basic operational preorders themselves as a part of operational semantics, for their purpose is to define the operational notion of contextual preorder.

The general identification of these three approaches is the first main contribution of the paper. Our second contribution is more technical. We illustrate the three approaches with a nontrivial case study: the combination of (finitary) nondeterminism with probabilistic choice,

³ Unfortunately, there is no space to include these results, which were obtained while the first author was on an internship in Ljubljana in 2017, in this paper.

which is a combination of effects that enjoys a certain notoriety for some of the technical complications it incurs [11, 12, 21, 20, 2, 3, 7]. On the operational side, we consider effect trees as Markov decision processes (MDPs), and we define a basic operational preorder based on the comparison of values of MDPs. On the denotational side, we make use of recently developed domain-theoretic models of combined nondeterministic and probabilistic choice [20, 3, 7]. On the axiomatic side, we give a simple axiomatisation, similar to axiomatisations in [12, 7]. Our main result is that the operationally, denotationally and axiomatically-defined basic operational preorders all coincide with each other. In fact, we give this result in two different versions. The first is for an *angelic* interpretation of nondeterminism, in which nondeterministic choices are resolved by a cooperative scheduler. The second is for *demonic* nondeterminism, where an antagonistic scheduler is assumed. In each case, our coincidence theorem suggests the canonicity of the preorder we obtain for the form of nondeterminism in question, with each of the three methods of definition providing a distinct perspective on it.

In Sections 2 and 3, we review the definition of effect trees and basic operational preorders, largely following [6]. Our main contribution starts in Sections 4, 5 and 6, which discuss the operational, denotational and axiomatic approaches to defining basic operational preorders. The discussion is illustrated using the example of combined nondeterminism and probabilistic choice. The main coincidence theorem, for this example, is then proved in Section 7. Finally, in Section 8, we briefly discuss related and further work.

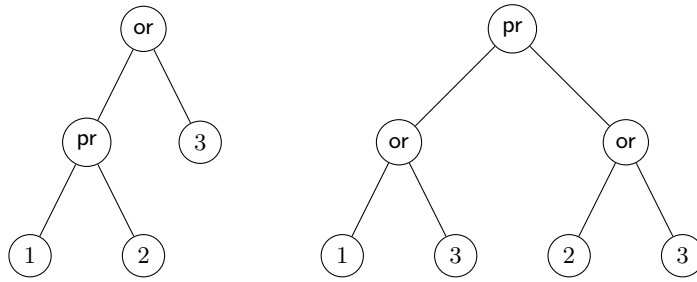
2 Effect trees

The general scenario this paper addresses is that of a programming language whose programs may perform effects as they compute. In this paper, we assume that the available effects are specified by an *effect signature*: a set Σ of operation symbols, each with an associated finite arity. We call the operations in Σ *effect operations*. This setting is explicitly that of [13]. More general effect signatures appear in the literature, e.g., allowing parameterised operations and infinite arities [6, 19]. The technical development in this paper can be generalised to such more general signatures. Since, however, the main running example considered in this paper has only binary operations, we restrict ourselves to finite arity operations for the sake of presentational convenience.

► **Example 1** (Signature for combined probabilistic and non-deterministic choice). Consider a programming language that can perform two effects: probabilistic and nondeterministic choice. An appropriate signature for such a language is $\Sigma_{\text{pr/nd}} = \{(\text{pr}, 2), (\text{or}, 2)\}$ containing two binary operations: nondeterministic choice *or*, and fair probabilistic choice *pr*. (As is well known, in programming languages with general recursion, all computable discrete probability distributions can be simulated using fair probabilistic choice.)

During the execution of a program with effects, three different situations can arise. Firstly, the computation process may trigger an effect, represented by some $o \in \Sigma$. The execution will then continue along one of the n possible continuation processes given as arguments to the operation o . Secondly, the execution may terminate, in which case it may produce a resulting value. Thirdly, the execution may continue forever without terminating and without invoking any effects. We call this last situation *silent nontermination* to distinguish it from *noisy nontermination*, which occurs when the computation process computes for ever while performing an infinite sequence of effects along the way.

The global behaviour of such a program is captured by the notion of an *effect tree*: a finitely branching tree, whose internal nodes represent effect operations, and whose leaves represent either termination with a result, or silent nontermination. The branches of the tree



■ **Figure 1** Two effect trees.

represent potential execution sequences of the program. Trees are allowed to be infinitely deep, with their infinite branches representing noisy nontermination. Such trees were introduced as *infinitary effect values* in [13], and used extensively in [6], where they are called *computation trees*. Two example trees, for computations that return natural number values, are drawn in Figure 1 below. The left-hand tree $\text{or}(\text{pr}(1, 2), 3)$ represents a program that first makes a nondeterministic choice and then a potential probabilistic choice, with the choices determining the resulting number. In the second tree $\text{pr}(\text{or}(1, 3), \text{or}(2, 3))$, the probabilistic choice is made first, followed by the relevant nondeterministic choice.

► **Definition 2.** The set $\text{Trees}(X)$ of *effect trees* with values from the set X is coinductively defined so that every tree has one of the following forms.

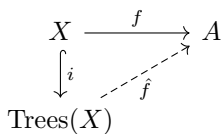
- The root of the tree is labelled with an operation $o \in \Sigma$, and the tree has the form $o(t_1, \dots, t_n)$ where n is the arity of o and $t_1, \dots, t_n \in \text{Trees}(X)$; or
- the tree is a leaf labelled with a value $x \in X$; or
- the tree is a leaf labelled with \perp .

As this is a coinductive definition, $\text{Trees}(X)$ contains trees of both finite and infinite depth.

We define a partial order on $\text{Trees}(X)$ by $t_1 \sqsubseteq t_2$ if and only if t_2 can be obtained from t_1 by replacing (possibly infinitely many) \perp -leaves appearing in t_1 with arbitrary replacement trees (rooted where the leaves were located). With this ordering, $\text{Trees}(X)$ is an ω -complete partial order (ω CPO) with least element \perp . Furthermore, by considering it as a tree constructor, every operation $o \in \Sigma$ defines a continuous (i.e., ω -continuous) function $o : \text{Trees}(X)^n \rightarrow \text{Trees}(X)$, where n is the arity of o . (For notational convenience, we use o for both operation symbol and function. The ambiguity can be resolved from the context.)

The properties described above state that $\text{Trees}(X)$ is a continuous Σ -algebra. In general, a *continuous Σ -algebra* is a pointed (i.e., with least element) ω CPO A with associated continuous functions $o_A : A^n \rightarrow A$ for every $o \in \Sigma$ of arity n . As morphisms between continuous Σ -algebras A and B , we consider functions $h : A \rightarrow B$ that are strict (i.e., preserve least element) continuous homomorphisms with respect to the Σ -algebra structure. We refer to such functions $h : A \rightarrow B$ as *continuous homomorphisms*, leaving the strictness property implicit. We write $\mathbf{ContAlg}_\Sigma$ for the category of continuous Σ -algebras and continuous homomorphisms. The characterisation of $\text{Trees}(X)$ below is standard.

► **Proposition 3.** $\text{Trees}(X)$ is the free continuous Σ -algebra over the set X .



29:6 Basic Operational Preorders for Probability and Nondeterminism

That is, for every function $f : X \rightarrow A$, where A is a continuous Σ -algebra, there exists a unique continuous homomorphism

$$\hat{f} : \text{Trees}(X) \rightarrow A$$

such that $f = \hat{f} \circ i$, where $i : X \rightarrow \text{Trees}(X)$ is the function mapping every $x \in X$ to the leaf-tree labelled x .

We use the above proposition to define a substitution operation on trees. For any tree $t \in \text{Trees}(X)$, every function $f : X \rightarrow \text{Trees}(Y)$ determines a tree $t[f]$ in $\text{Trees}(Y)$ defined by substitution, viz: $t[f] := \hat{f}(t)$.

3 Basic operational preorders

As discussed in Section 1, our interest in effect trees is that they provide a uniform template for defining *contextual preorders* for programming languages with algebraic effect operations specified by signature Σ . As in [6], the crucial data is provided by a preorder \preceq on $\text{Trees}(\mathbb{N})$, called the *basic operational preorder*. In order for the resulting contextual preorder to be well behaved, we ask for the basic operational preorder satisfy two properties: *admissibility* and *compositionality*. In this section, we review the definitions of these and related notions.

► **Definition 4** (Admissibility). A binary relation R on $\text{Trees}(X)$ is *admissible* if, for every ascending chain $(t_i)_{i \geq 0}$ and $(t'_i)_{i \geq 0}$, we have:

$$(t_i R t'_i \text{ for all } i) \implies \left(\bigsqcup_{i \geq 0} t_i \right) R \left(\bigsqcup_{i \geq 0} t'_i \right) .$$

► **Definition 5** (Compatibility). A binary relation R on $\text{Trees}(X)$ is *compatible* if, for every $o \in \Sigma$ of arity n , and for all trees t_1, \dots, t_n and t'_1, \dots, t'_n , we have:

$$(t_i R t'_i \text{ for all } i = 1, \dots, n) \implies o(t_1, \dots, t_n) R o(t'_1, \dots, t'_n) .$$

If a compatible relation is a preorder then it is called a *precongruence*. If it is an equivalence relation it is called a *congruence*.

The next two definitions make use of the substitution operation on trees defined at the end of Section 2.

► **Definition 6** (Substitutivity). A binary relation R on $\text{Trees}(X)$ is *substitutive* if, for all trees t, t' and $\{t_x\}_{x \in X}$, we have:

$$t R t' \implies t[x \mapsto t_x] R t'[x \mapsto t_x] .$$

► **Definition 7** (Compositionality). A binary relation R on $\text{Trees}(X)$ is *compositional* if, for all trees t, t' , $\{t_x\}_{x \in X}$, and $\{t'_x\}_{x \in X}$, we have:

$$(t R t' \text{ and } t_x R t'_x \text{ for all } x \in X) \implies t[x \mapsto t_x] R t'[x \mapsto t'_x] .$$

► **Proposition 8.** Let \preceq be a preorder on $\text{Trees}(\mathbb{N})$.

1. If \preceq is compositional then it is a substitutive precongruence.
2. If \preceq is an admissible substitutive precongruence then it is compositional.

Proof. We prove statement 2. Suppose \preceq is admissible, substitutive and compatible. Suppose also that $t \preceq t'$ and $t_n \preceq t'_n$, for all $n \in \mathbb{N}$. By substitutivity, we have $t[n \mapsto t_n] \preceq t'[n \mapsto t_n]$. We would like to use compatibility to derive that also $t'[n \mapsto t_n] \preceq t'[n \mapsto t'_n]$, however this is only possible if t' is finite. The solution is to use finite approximations (s'_i) of t' satisfying $\bigsqcup_i s'_i = t'$. For each finite tree s'_i we have that $s'_i[n \mapsto t_n] \preceq s'_i[n \mapsto t'_n]$, by compatibility. Hence, by admissibility, $t'[n \mapsto t_n] \preceq t'[n \mapsto t'_n]$, whence $t[n \mapsto t_n] \preceq t'[n \mapsto t'_n]$ by transitivity. \blacktriangleleft

4 Operationally-defined preorders

In this section, we consider our first approach to defining an admissible and compositional basic operational preorder \preceq on $\text{Trees}(\mathbb{N})$. We call this method *operational*. Its characteristic is that the preorder \preceq is directly defined using a mathematical model of the way that an effect tree in $\text{Trees}(\mathbb{N})$ will be executed. There is not much to say in general about this approach, since such execution models vary enormously from one effect to another. The main point to emphasise is that there is no general reason for admissibility and compositionality to hold for such operationally defined preorders. Accordingly, these properties need to be established on a case-by-case basis.

The operational approach to defining basic preorders is illustrated for several examples of effects in [6]. The main goal of the section is to demonstrate the approach using a different example, the signature $\Sigma_{\text{pr/nd}} = \{\text{pr}, \text{or}\}$ from Example 1, which is of interest because of the interplay between probabilistic and nondeterministic effects. In this case, trees in $\text{Trees}(\mathbb{N})$ have both probabilistic and nondeterministic branching nodes, as in Figure 1.

It is natural to consider such trees as (countable state) Markov decision processes, with the leaves representing nodes which either carry an observable value from \mathbb{N} , or which represent nontermination \perp . Nondeterministic choices may be thought of as being resolved by an external agent, the scheduler. We model the actions of the scheduler by a function $s : \{l, r\}^* \rightarrow \{l, r\}$. The idea is that a word $w \in \{l, r\}^*$ represents a finite path of left/right choices from the root of a tree $t \in \text{Trees}(\mathbb{N})$. If the computation reaches a nondeterministic choice at the node indexed by w then it takes the left/right branch according to the value of $s(w)$. This way of representing choices has some redundancy (in every tree that is not a complete infinite binary tree, there will be words w that do not index nodes in the tree; if $s(\varepsilon) = l$ then the value of s on words beginning with r is immaterial; the value of $s(w)$ on words w that index probabilistic nodes in t is irrelevant, etc.), but it is simple and convenient for future purposes. For any given $t \in \text{Trees}(\mathbb{N})$, such a function $s : \{l, r\}^* \rightarrow \{l, r\}$ can be thought of as a (deterministic) *strategy* for the scheduler, in which the choice of direction at a nondeterministic node can respond to the outcomes of probabilistic nodes higher up the tree.

A strategy s and a tree t in combination determine a subtree $t \upharpoonright s$, defined by removing, at every nondeterministic node in t with index w , the child tree that is not selected by $s(w)$. So $t \upharpoonright s$ is a tree that has binary branching at probabilistic nodes, and unary branching at nondeterministic nodes. It is thus, in effect, a purely probabilistic tree, with leaves in $\mathbb{N} \cup \{\perp\}$, and so may be viewed as a Markov chain, in which the branching nodes are fair binary choices, determining a subprobability distribution over \mathbb{N} . Specifically, each $n \in \mathbb{N}$ is assigned the probability that a run of the Markov chain will end at a leaf labelled with n . This is a subprobability distribution in general because there can be a positive probability of nontermination (either at a \perp leaf, or along an infinite branch).

The *angelic* interpretation of nondeterminism takes into account the possibility of a nondeterministic computation achieving a specified goal, given a cooperative scheduler. The *demonic* interpretation, models the certainty with which a goal can be achieved, however

adversarial the scheduler. This suggests the two basic operational preorders below. In each case, we consider functions $h: \mathbb{N} \rightarrow [0, \infty]$ assigning desirability weightings to possible results of a run of the computation. We then define $t \preceq t'$ if, for any h , the ‘expected’ desirability weighting of t' exceeds that of t . Here, ‘expected’ is in inverted commas, because we have to take into account the actions of the scheduler, so this is not just a probabilistic expectation. In the case of angelic nondeterminism, the scheduler will help us, whereas, under demonic nondeterminism, it will impede us. Technically this is taken account of by considering suprema of probabilistic expectations in the angelic case, and infima in the demonic case.

$$t \preceq_{\text{pr/ang}}^{\text{op}} t' \Leftrightarrow \forall h: \mathbb{N} \rightarrow [0, \infty] \quad \sup_s \mathbf{E}_{t|s}(h) \leq \sup_s \mathbf{E}_{t'|s}(h)$$

$$t \preceq_{\text{pr/dem}}^{\text{op}} t' \Leftrightarrow \forall h: \mathbb{N} \rightarrow [0, \infty] \quad \inf_s \mathbf{E}_{t|s}(h) \leq \inf_s \mathbf{E}_{t'|s}(h)$$

Here $\mathbf{E}_{t|s}(h)$ means the expectation of the function h under the subprobability distribution on \mathbb{N} induced by the Markov chain $t|s$. In Markov-decision-process terminology, each preorder says that the *value* of the MDP t , for any weighting h , is below the value of t' for h . In the angelic case the value maximises the expectation of h , in the demonic case it minimises it.

► **Proposition 9.** *The preorders $\preceq_{\text{pr/ang}}^{\text{op}}$ and $\preceq_{\text{pr/dem}}^{\text{op}}$ are admissible and compositional.*

We outline the proof of this proposition in the case of $\preceq_{\text{pr/dem}}^{\text{op}}$. The proof for $\preceq_{\text{pr/ang}}^{\text{op}}$ is easier, largely because the analogue of the lemma below is trivial in the case of angelic nondeterminism.

► **Lemma 10.** *Consider $\text{Trees}(\mathbb{N})$ and $[0, +\infty]$ as ω CPOs. Then, for any $h: \mathbb{N} \rightarrow [0, \infty]$, the value-finding function F_h is continuous:*

$$F_h: t \mapsto \inf_s \mathbf{E}_{t|s}(h) : \text{Trees}(\mathbb{N}) \rightarrow [0, +\infty]$$

Proof. The set $S = \{l, r\}^{\{l, r\}^*}$ of strategies is a countably-based compact Hausdorff space under the product topology. (It is Cantor space.) It is easy to see that the function

$$G_h: (s, t) \mapsto \mathbf{E}_{t|s}(h) : S \times \text{Trees}(\mathbb{N}) \rightarrow [0, +\infty]$$

is continuous. Essentially, it follows that F_h is continuous because it is defined from G_h by taking an infimum over a compact set. This can be made precise using, e.g., the general machinery in Section 7.3 of [16]. For completeness, we give a self-contained argument.

Suppose (t_i) is an ascending chain of trees. Because S is compact, there is $s_i \in S$ with $\inf_s G_h(s, t_i) = G_h(s_i, t_i)$, and we can then extract a convergent subsequence (s_{a_i}) of (s_i) such that $s_{a_i} \rightarrow s_\infty$ in S . Then:

$$\sup_i \inf_s G_h(s, t_i) = \sup_i G_h(s_i, t_i) = \sup_i G_h(s_{a_i}, t_{a_i}) = G_h(s_\infty, \bigsqcup_i t_i) \geq \inf_s G_h(s, \bigsqcup_i t_i),$$

where the second equality holds because $G_h(s_i, t_i)$ is an ascending sequence, and the third by the continuity of G_h . We have shown that $\sup_i \inf_s G_h(s, t_i) \geq \inf_s G_h(s, \bigsqcup_i t_i)$, i.e., $\sup_i F_h(t_i) \geq F_h(\bigsqcup_i t_i)$. Therefore F_h is continuous (since it is obviously monotone). ◀

The admissibility of $\preceq_{\text{pr/dem}}^{\text{op}}$ follows easily from the lemma. Suppose $t_i \preceq_{\text{pr/dem}}^{\text{op}} t'_i$, for ascending chains (t_i) and (t'_i) . Then $F_h(t_i) \leq F_h(t'_i)$, for all i and h . By the lemma, $F_h(\bigsqcup_i t_i) \leq F_h(\bigsqcup_i t'_i)$, for all h . So indeed $\bigsqcup_i t_i \preceq_{\text{pr/dem}}^{\text{op}} \bigsqcup_i t'_i$.

For compositionality, by Proposition 8, it suffices to show that $\preceq_{\text{pr/dem}}^{\text{op}}$ is a substitutive precongruence. The compatibility properties of a precongruence are easily shown. Substitutivity follows from the lemma below.

► **Lemma 11.** *Suppose t and $\{t_n\}_{n \in \mathbb{N}}$ are trees in $\text{Trees}(\mathbb{N})$ then, for any weighting h ,*

$$\inf_s \mathbf{E}_{t[n \rightarrow t_n] \uparrow s}(h) = \inf_s \mathbf{E}_{t \uparrow s}(\hat{h}) \quad \text{where} \quad \hat{h}(n) = \inf_s \mathbf{E}_{t_n \uparrow s}(h) .$$

This lemma is proved first for finite trees, by induction on their height. It is then extended to infinite trees by expressing them as suprema of finite trees, and applying Lemma 10.

We end this section by observing that a natural attempt to simplify the definitions of $\preceq_{\text{pr}/\text{ang}}^{\text{op}}$ and $\preceq_{\text{pr}/\text{dem}}^{\text{op}}$ does not work. Instead of considering arbitrary weightings $h: \mathbb{N} \rightarrow [0, \infty]$, one might restrict to functions $h: \mathbb{N} \rightarrow \{0, 1\}$, which can be viewed as specifying goal subsets $H \subseteq \mathbb{N}$. Proceeding analogously to above, we compare suprema of probabilities of landing in H in the angelic case, and infima in the demonic case. For both the angelic and demonic versions, the desired compositionality property fails.

► **Proposition 12.** *Neither of the formulas below defines a compositional relation $t \preceq t'$.*

$$\forall H \subseteq \mathbb{N} \quad \sup_s \mathbf{P}_{t \uparrow s}(H) \leq \sup_s \mathbf{P}_{t' \uparrow s}(H)$$

$$\forall H \subseteq \mathbb{N} \quad \inf_s \mathbf{P}_{t \uparrow s}(H) \leq \inf_s \mathbf{P}_{t' \uparrow s}(H)$$

Proof. We use the two trees in Figure 1, representing the expressions $A = 3 \text{ or } (1 \text{ pr } 2)$ and $B = (3 \text{ or } 1) \text{ pr } (3 \text{ or } 2)$. It is easily checked that, for every subset $H \subseteq \{1, 2, 3\}$, it holds that $\sup_s \mathbf{P}_{A \uparrow s}(H) = \sup_s \mathbf{P}_{B \uparrow s}(H)$ and $\inf_s \mathbf{P}_{A \uparrow s}(H) = \inf_s \mathbf{P}_{B \uparrow s}(H)$. Thus A is equivalent to B under both preorders.

However, one can build a family $\{t_1, t_2, t_3\}$ such that $A[i \mapsto t_i] = t_3 \text{ or } (t_1 \text{ pr } t_2) = C$ is not equivalent to $B[i \mapsto t_i] = (t_3 \text{ or } t_1) \text{ pr } (t_3 \text{ or } t_2) = D$, which contradicts substitutivity. Let $t_1 = 0 \text{ pr } (0 \text{ pr } (0 \text{ pr } (0 \text{ pr } 1)))$, $t_2 = 1$ and $t_3 = 0 \text{ pr } (0 \text{ pr } (0 \text{ pr } 1))$. The distinguishing factor will be the probability associated with the subset $\{1\}$.

A simple calculation shows that $\sup_s \mathbf{P}_{C \uparrow s}(\{1\}) = 9/16 \neq 5/8 = \sup_s \mathbf{P}_{D \uparrow s}(\{1\})$. Similarly $\inf_s \mathbf{P}_{C \uparrow s}(\{1\}) = 1/4 \neq 3/16 = \inf_s \mathbf{P}_{D \uparrow s}(\{1\})$. This contradicts the substitutivity and hence also the compositionality of both preorders. ◀

The necessity of using quantitative properties to obtain a compositional preorder is consistent with a general need for quantitative concepts that can be found in the literature on probabilistic computation. For example, in [8, 9], quantitative logics are required to obtain compositional reasoning methods. Similarly, in [10], quantitative observations are needed to distinguish non-bisimilar processes combining probabilistic and nondeterministic choice.

5 Denotationally-defined preorders

Our second approach to defining an admissible and compositional basic denotational preorder \preceq on $\text{Trees}(\mathbb{N})$ is to make use of established constructions from domain theory. Under this approach, admissibility and compositionality of the defined preorder \preceq hold for general reasons. Since this approach essentially amounts to giving a denotational semantics to effect trees, we call it the *denotational* method of defining a basic operational preorder.

In order to define a basic operational preorder using the denotational method, one needs to merely provide a continuous Σ -algebra D (see Section 2), together with a function $j: \mathbb{N} \rightarrow D$. Define $[\cdot]: \text{Trees}(\mathbb{N}) \rightarrow D$ to be the unique continuous homomorphism that makes the diagram below commute.

$$\begin{array}{ccc} \mathbb{N} & \xrightarrow{j} & D \\ \downarrow i & \nearrow [\cdot] & \\ \text{Trees}(\mathbb{N}) & & \end{array}$$

29:10 Basic Operational Preorders for Probability and Nondeterminism

The map $\llbracket \cdot \rrbracket : \text{Trees}(\mathbb{N}) \rightarrow D$ is used to induce the basic operational preorder \preceq_D from the partial order relation on the ω CPO D .

$$t \preceq_D t' \Leftrightarrow \llbracket t \rrbracket \sqsubseteq \llbracket t' \rrbracket .$$

► **Proposition 13.** *The relation \preceq_D is admissible precongrence.*

The proof is immediate: admissibility follows from the continuity of $\llbracket \cdot \rrbracket$, and compatibility because $\llbracket \cdot \rrbracket$ is a homomorphism.

In order to obtain substitutivity, hence compositionality, a further property is required.

► **Definition 14 (Factorisation property).** The map $j : \mathbb{N} \rightarrow D$ is said to have the *factorisation property* if, for every function $f : \mathbb{N} \rightarrow D$, there exists a continuous homomorphism $h_f : D \rightarrow D$ such that $f = h_f \circ j$.

$$\mathbb{N} \begin{array}{c} \xrightarrow{j} D \xrightarrow{h_f} D \\ \searrow f \nearrow \end{array}$$

► **Proposition 15.** *If $j : \mathbb{N} \rightarrow D$ has the factorisation property then the relation \preceq_D is substitutive, hence it is an admissible compositional precongrence.*

Proof. Suppose $\sigma : \mathbb{N} \rightarrow \text{Trees}(\mathbb{N})$ is any substitution. Let $\hat{\sigma} : \text{Trees}(\mathbb{N}) \rightarrow \text{Trees}(\mathbb{N})$ be the continuous homomorphism such that $\hat{\sigma} \circ i = \sigma$. Consider the map $g := \llbracket \cdot \rrbracket \circ \hat{\sigma} \circ i : \mathbb{N} \rightarrow D$. By the factorisation property, there exists $h_g : D \rightarrow D$ such that $g = h_g \circ j$. Expanding this, and using the definition of $\llbracket \cdot \rrbracket$, we have:

$$\llbracket \cdot \rrbracket \circ \hat{\sigma} \circ i = h_g \circ j = h_g \circ \llbracket \cdot \rrbracket \circ i .$$

It then follows from the uniqueness property of Proposition 3 that

$$\llbracket \cdot \rrbracket \circ \hat{\sigma} = h_g \circ \llbracket \cdot \rrbracket , \tag{3}$$

because both maps are continuous homomorphisms.

Now, for substitutivity, suppose that $t \preceq_D t'$, i.e., $\llbracket t \rrbracket \leq \llbracket t' \rrbracket$. Then $h_g(\llbracket t \rrbracket) \leq h_g(\llbracket t' \rrbracket)$ by monotonicity. That is $\llbracket \hat{\sigma}(t) \rrbracket \leq \llbracket \hat{\sigma}(t') \rrbracket$, by (3). This says that $\llbracket t[\sigma] \rrbracket \leq \llbracket t'[\sigma] \rrbracket$. That is $t[\sigma] \preceq_D t'[\sigma]$, as required. ◀

In practice, it is usually not necessary to prove the factorisation property directly. Instead it holds as a consequence of the continuous algebra D and map $j : \mathbb{N} \rightarrow D$ being derived from a suitable monad. The next result establishes general conditions under which this holds.

► **Proposition 16.** *Let \mathbf{S} be a category with a faithful functor $U : \mathbf{S} \rightarrow \mathbf{Set}$. Suppose also that \mathbf{S} has an object N such that $UN = \mathbb{N}$, and every hom set $\mathbf{S}(N, X)$ is mapped bijectively by U to $\mathbf{Set}(\mathbb{N}, UX)$. Suppose also that (T, η, μ) is a monad on \mathbf{S} with the following properties: there is a continuous Σ -algebra structure on UTN ; and, for every map $f : N \rightarrow TN$, the induced function Uf^* , where $f^* : TN \rightarrow TN$ is the Kleisli lifting, is a continuous homomorphism. Then defining D to be the continuous Σ -algebra on UTN , and j to be $U\eta : \mathbb{N} \rightarrow UTN$, it follows that j has the factorisation property.*

We omit the easy proof. Although the statement of the proposition is verbose, the result is relatively easy to apply in practice, as the examples we consider next will illustrate.

In the remainder of the section, we return to our main example, and again define basic operational preorders for the combination of probabilistic choice and nondeterminism (both angelic and demonic), but this time we use the denotational method. Accordingly, we call the defined preorders $\preceq_{\text{pr/ang}}^{\text{den}}$ and $\preceq_{\text{pr/dem}}^{\text{den}}$

We use the powerdomains combining probabilistic choice and nondeterminism defined in [7, §3.4], although our setting is simpler because we only need to apply them to sets. The basic idea of these constructions is that a computation with probabilistic and nondeterministic choice is modelled as a set of subprobability distributions, where the set collects the possible nondeterministic outcomes, each of which is probabilistic in nature. As is standard, the sets relevant to angelic nondeterminism are the closed sets in the Scott topology, whereas those relevant to demonic nondeterminism are the compact upper-closed sets, see [18]. Due to the combination with probabilistic choice, sets are further required to be convex; see, for example, the discussion in [7].

Let $\mathcal{V}_{\leq 1} X$ be the ω CPO of (discrete) subprobability distributions on a set X . We write $\mathcal{HV}_{\leq 1} X$ for the ω CPO of nonempty Scott-closed convex subsets of $\mathcal{V}_{\leq 1} X$ ordered by subset inclusion \subseteq . We write $\mathcal{SV}_{\leq 1} X$ for the ω CPO of nonempty Scott-compact convex upper-closed subsets of $\mathcal{V}_{\leq 1} X$ ordered by reverse inclusion \supseteq . The ω CPOs $\mathcal{HV}_{\leq 1} X$ and $\mathcal{SV}_{\leq 1} X$ are both continuous algebras for $\Sigma_{\text{pr/nd}}$. In both cases, the operations are defined by:

$$\text{or}(A, B) = \text{conv}(A \cup B) \quad \text{pr}(A, B) = \left\{ \frac{1}{2}a + \frac{1}{2}b \mid a \in A, b \in B \right\} ,$$

where conv is the convex closure operation. We remark that these straightforward uniform definitions are possible because of the simple structure of the domains $\mathcal{HV}_{\leq 1} X$ and $\mathcal{SV}_{\leq 1} X$, over a set X . For the more general lower and upper ‘Kegelspitze’ considered in [7], additional order-theoretic and topological closure operations need to be applied.

To apply the above in the angelic case, we use the fact that $\mathcal{HV}_{\leq 1} X$ is the free Kegelspitze join semilattice over a set X [7, Corollary 3.15] (where the result is proved more generally for domains). It follows that $\mathcal{HV}_{\leq 1}$ is a monad on **Set** itself satisfying the conditions of Proposition 16. Thus defining $D_{\text{pr/ang}} = \mathcal{HV}_{\leq 1} \mathbb{N}$, and $j(n) = \downarrow \delta(n)$ (where $\delta(n)$ is the Dirac probability distribution that assigns probability 1 to n and 0 to all other numbers, and $\downarrow x$ is the down-closure $\{y \mid y \leq x\}$), the induced $\llbracket \cdot \rrbracket_{\text{pr/ang}} : \text{Trees}(\mathbb{N}) \rightarrow D_{\text{pr/ang}}$ defines an admissible and compositional preorder

$$t \preceq_{\text{pr/ang}}^{\text{den}} t' \Leftrightarrow \llbracket t \rrbracket_{\text{pr/ang}} \leq \llbracket t' \rrbracket_{\text{pr/ang}} .$$

Similarly, in the demonic case, we use [7, Corollary 3.16], which characterises $\mathcal{SV}_{\leq 1} X$ as the free Kegelspitze meet semilattice over X . Again $\mathcal{SV}_{\leq 1}$ is a monad on **Set** to which Proposition 16 applies. In this case, we define $D_{\text{pr/dem}} = \mathcal{SV}_{\leq 1} \mathbb{N}$, and $j(n) = \{\delta(n)\}$. Then the induced $\llbracket \cdot \rrbracket_{\text{pr/dem}} : \text{Trees}(\mathbb{N}) \rightarrow D_{\text{pr/dem}}$ defines an admissible and compositional preorder

$$t \preceq_{\text{pr/dem}}^{\text{den}} t' \Leftrightarrow \llbracket t \rrbracket_{\text{pr/dem}} \leq \llbracket t' \rrbracket_{\text{pr/dem}} .$$

6 Axiomatically-defined preorders

In this section, we look at the definition of basic operational preorders by axiomatising properties of the operations in the effect signature Σ . Since we are defining a preorder, it is appropriate for the axiomatisation to involve inequalities specifying desired properties of the operational preorder. As the technical framework for this, we allow axiomatisations involving infinitary Horn-clauses of inequalities between infinitary terms. This provides a flexible general setting for axiomatising admissible and compositional preorders on $\text{Trees}(\mathbb{N})$.

Let Vars be a set of countably many distinct variables. By an *expression*, we mean a tree $e \in \text{Trees}(\text{Vars})$. The use of trees incorporates infinitary non-well-founded terms alongside

29:12 Basic Operational Preorders for Probability and Nondeterminism

the usual finite algebraic terms. By an *inequality* we mean a statement $e_1 \leq e_2$, where e_1, e_2 are expressions. By an (*infinitary*) *Horn clause* we mean an implication of the form:

$$\left(\bigwedge_{i \in I} e_i \leq e'_i \right) \implies e \leq e' , \quad (4)$$

An *effect theory* T is a set of Horn clauses.

A precongruence \preceq on $\text{Trees}(X)$ is said to *satisfy* a Horn clause (4) if, for every environment $\rho: \text{Vars} \rightarrow \text{Trees}(X)$, the implication below holds (recall the notation for tree substitution from Section 2).

$$\left(\bigwedge_{i \in I} e_i[\rho] \preceq e'_i[\rho] \right) \implies e[\rho] \preceq e'[\rho] .$$

We say that a precongruence \preceq is a *model* of a Horn clause theory T if it satisfies every Horn clause in T . We consider models as subsets of $\text{Trees}(X) \times \text{Trees}(X)$, partially ordered by inclusion. Note that models are precongruences by assumption.

► **Proposition 17.** *Every Horn clause theory T has a smallest admissible model*

$$\preceq_T \subseteq \text{Trees}(X) \times \text{Trees}(X) ,$$

for any set X . The model \preceq_T is substitutive. In the case that $X = \mathbb{N}$, the smallest admissible model is thus an admissible compositional preorder.

Proof. It is easily seen that the intersection of any set of admissible models is itself an admissible model. Thus the intersection of the set of all admissible models is the required smallest admissible model \preceq_T . For substitutivity, define

$$t \preceq_S t' \iff \forall \sigma : X \rightarrow \text{Trees}(X). t[\sigma] \preceq_T t'[\sigma] . \quad (5)$$

Using the substitution $\sigma(x) = x$, we see that $\preceq_S \subseteq \preceq_T$. Conversely, it is easily shown that the relation \preceq_S is itself an admissible model of T . Thus $\preceq_T \subseteq \preceq_S$. Since \preceq_T and \preceq_S coincide, (5) asserts the substitutivity of \preceq_T . The statement about compositionality now follows from Proposition 8. ◀

Given the proposition, we can use any effect theory to define an admissible and compositional basic operational preorder, namely the smallest admissible model over \mathbb{N} . We now apply this method to our running example of combined nondeterminism and probabilistic choice. The axioms are given in Figure 2.

The axioms include a special axiom for \perp , which is legitimate since \perp is a tree, hence an expression. The axioms for probability include three standard equalities (each of which is given officially as two inequalities), and one Horn approximation axiom, *Appr*, which is separated out for the sake of Proposition 19 below. The axioms for nondeterminism are split into a neutral list, followed by further axioms for angelic and demonic nondeterminism respectively. Finally, there is a distributivity axiom that relates probabilistic and nondeterministic choice. Our two effect theories of interest are:

$$T_{\text{pr/ang}} = \text{Bot}, \text{Prob}, \text{Appr}, \text{Nondet}, \text{Ang}, \text{Dist}$$

$$T_{\text{pr/dem}} = \text{Bot}, \text{Prob}, \text{Appr}, \text{Nondet}, \text{Dem}, \text{Dist} .$$

Bot:	$\perp \leq x$
Prob:	$x \text{ pr } x = x, x \text{ pr } y = y \text{ pr } x, (x \text{ pr } y) \text{ pr } (z \text{ pr } w) = (x \text{ pr } z) \text{ pr } (y \text{ pr } w)$
Appr:	$x \text{ pr } y \leq y \implies x \leq y$
Nondet:	$x \text{ or } x = x, x \text{ or } y = y \text{ or } x, x \text{ or } (y \text{ or } z) = (x \text{ or } y) \text{ or } z$
Ang:	$x \text{ or } y \geq x$
Dem:	$x \text{ or } y \leq x$
Dist:	$x \text{ pr } (y \text{ or } z) = (x \text{ pr } y) \text{ or } (x \text{ pr } z)$

■ **Figure 2** Horn theory for mixed probability and non determinism.

We then define $\preceq_{\text{pr/ang}}^{\text{ax}}$ as the smallest admissible model of $T_{\text{pr/ang}}$ over \mathbb{N} , and $\preceq_{\text{pr/dem}}^{\text{ax}}$ as the smallest admissible model of $T_{\text{pr/dem}}$. By Proposition 17, both these basic operational preorders are admissible and compositional.

To end the section, we observe that the Horn-clause axiom in Figure 2 can be replaced with an equational axiom, albeit one involving an infinitary expression.

► **Definition 18.** Let t be a tree. For each $n \in \mathbb{N} \cup \{\infty\}$, we define a tree $(1-2^{-n})t$ inductively by $(1-2^{-0})t = \perp$ and $(1-2^{-(n+1)})t = t \text{ pr } (1-2^{-n})t$. The tree $(1-2^{-\infty})t$ is defined as $\bigsqcup_n (1-2^{-n})t$.

► **Proposition 19.** For any effect theory containing the Bot and Prob axioms, an admissible model satisfies the Appr axiom if and only if it satisfies the equation $(1-2^{-\infty})x = x$.

Proof. We first derive $(1-2^{-\infty})x = x$, from the axioms with Appr. It is clear that $(1-2^{-n})x \leq x$ for every $n < \infty$, and therefore $(1-2^{-\infty})x \leq x$ by admissibility. We also have $x \text{ pr } (1-2^{-n})x \leq (1-2^{-(n+1)})x$, and so, again by admissibility, $x \text{ pr } (1-2^{-\infty})x \leq (1-2^{-\infty})x$. Whence, by the Horn axiom, $x \leq (1-2^{-\infty})x$. We have thus derived $(1-2^{-\infty})x = x$.

For the converse, we assume $(1-2^{-\infty})x = x$ and derive Appr. Suppose $x \text{ pr } y \leq y$. Then also $x \text{ pr } (x \text{ pr } y) \leq y$, and $x \text{ pr } (x \text{ pr } (x \text{ pr } y)) \leq y$, etc. So also $x \text{ pr } \perp \leq y$, and $x \text{ pr } (x \text{ pr } \perp) \leq y$, and $x \text{ pr } (x \text{ pr } (x \text{ pr } \perp)) \leq y$, etc. That is, $(1-2^{-n})x \leq y$, for every $n < \infty$. By admissibility, $(1-2^{-\infty})x \leq y$. Whence, by the assumed axiom, $x \leq y$ as required. ◀

7 The coincidence theorem

Our main theorem is that our operational, denotational and axiomatic preorders for combined probability and nondeterminism all coincide, in both the angelic and demonic cases.

► **Theorem 20 (Coincidence theorem).**

1. The three preorders $\preceq_{\text{pr/ang}}^{\text{op}}$, $\preceq_{\text{pr/ang}}^{\text{den}}$ and $\preceq_{\text{pr/ang}}^{\text{ax}}$, for mixed probability and angelic nondeterminism, coincide.
2. Similarly, the preorders $\preceq_{\text{pr/dem}}^{\text{op}}$, $\preceq_{\text{pr/dem}}^{\text{den}}$ and $\preceq_{\text{pr/dem}}^{\text{ax}}$, for mixed probability and demonic nondeterminism, coincide.

We outline the proof of the theorem in the demonic case, which we split into three lemmas. The proof for the angelic case is similar.

► **Lemma 21.** $\preceq_{\text{pr/dem}}^{\text{ax}} \subseteq \preceq_{\text{pr/dem}}^{\text{op}}$.

Proof. It is easily checked that $\preceq_{\text{pr/dem}}^{\text{op}}$ satisfies the axioms of $T_{\text{pr/dem}}$. Since $\preceq_{\text{pr/dem}}^{\text{op}}$ is admissible and \preceq^{ax} is the smallest admissible model, $\preceq_{\text{pr/dem}}^{\text{ax}} \subseteq \preceq_{\text{pr/dem}}^{\text{op}}$. ◀

We remark on the following aspect of the above result. The distributivity axiom Dist of Figure 2 is sometimes discussed as expressing that nondeterministic choices are resolved before probabilistic ones; see, e.g., [12, 7]. Such statements need careful interpretation. The definition of $\preceq_{\text{pr/dem}}^{\text{op}}$, which is based on implementing nondeterministic schedulers as strategies for MDPs, explicitly allows the scheduler's choices to take account of the outcomes of probabilistic choices that precede it. Nevertheless, the distributivity axiom is sound.

► **Lemma 22.** $\preceq_{\text{pr/dem}}^{\text{op}} = \preceq_{\text{pr/dem}}^{\text{den}}$.

Proof. We make use of the functional representation of $\mathcal{SV}_{\leq 1} \mathbb{N}$ from [7] (see also [3]). For any topological space X , we write $\mathcal{L}(X)$ for the space of all *lower semicontinuous* functions from X to $[0, \infty]$ (i.e., functions that are continuous with respect to the Scott topology on $[0, \infty]$), and we endow $\mathcal{L}(X)$ itself with the Scott topology. The space $D' = \mathcal{L}(\mathcal{L}(\mathbb{N}))$ carries a continuous $\Sigma_{\text{pr/nd}}$ -algebra structure

$$(F \text{ or } G)(f) = \min(F(f), G(f)) \quad (F \text{ pr } G)(f) = \frac{1}{2}F(f) + \frac{1}{2}G(f) .$$

(There is another $\Sigma_{\text{pr/nd}}$ -algebra structure, relevant to angelic nondeterminism, in which \min is replaced with \max .) Define $j' : \mathbb{N} \rightarrow D'$ by $j'(n)(f) = f(n)$. This induces $\llbracket \cdot \rrbracket'_{\text{pr/dem}} : \text{Trees}(\mathbb{N}) \rightarrow D'$ satisfying $\llbracket \cdot \rrbracket'_{\text{pr/dem}} \circ i = j'$, as in Section 5. We show that $\llbracket t \rrbracket'_{\text{pr/dem}}(h) = F_h(t)$, where F_h is defined as in Lemma 10. For this, the function $t \mapsto (h \mapsto F_h(t))$ is easily shown to be a $\Sigma_{\text{pr/nd}}$ -algebra homomorphism satisfying $F_h(i(n)) = j'(n)$. Moreover, it is continuous by Lemma 10. Thus it indeed coincides with $\llbracket \cdot \rrbracket'_{\text{pr/dem}}$. By the definition of F_h , it follows that that $t \preceq_{\text{pr/dem}}^{\text{op}} t'$ if and only if $\llbracket t \rrbracket'_{\text{pr/dem}} \leq \llbracket t' \rrbracket'_{\text{pr/dem}}$.

Corollary 4.7 of [7] provides a functional representation of $\mathcal{SV}_{\leq 1} X$ inside $\mathcal{L}(\mathcal{L}(X))$. In the case $X = \mathbb{N}$, consider the function

$$\Lambda : A \mapsto \left(f \mapsto \inf_{p \in A} \mathbf{E}_p f \right) : \mathcal{SV}_{\leq 1} \mathbb{N} \rightarrow D' .$$

It is shown in [7] that Λ is a continuous $\Sigma_{\text{pr/nd}}$ -algebra homomorphism, and also an order embedding (i.e., $\Lambda(A) \leq \Lambda(B)$ implies $A \supseteq B$). By the uniqueness property of Proposition 3, it thus holds that $\Lambda \circ \llbracket \cdot \rrbracket_{\text{pr/dem}} = \llbracket \cdot \rrbracket'_{\text{pr/dem}}$. We therefore have

$$t \preceq_{\text{pr/dem}}^{\text{op}} t' \Leftrightarrow \llbracket t \rrbracket'_{\text{pr/dem}} \leq \llbracket t' \rrbracket'_{\text{pr/dem}} \Leftrightarrow \llbracket t \rrbracket_{\text{pr/dem}} \leq \llbracket t' \rrbracket_{\text{pr/dem}} \Leftrightarrow t \preceq_{\text{pr/dem}}^{\text{den}} t' ,$$

where the middle equivalence holds because Λ is an order embedding. ◀

► **Lemma 23.** $\preceq_{\text{pr/dem}}^{\text{den}} \subseteq \preceq_{\text{pr/dem}}^{\text{ax}}$.

Proof. The proof proceeds in three steps.

1. Prove that both preorders coincide on *probability trees* (i.e., trees without or nodes).
2. Prove the inclusion of preorders for trees with a *finite* number of or nodes.
3. Use finite approximations and admissibility to conclude the general case.

We omit discussion of the first step, which is comparatively straightforward, cf. [4].

For step 2, suppose $t \preceq_{\text{pr/dem}}^{\text{den}} t'$ where t, t' are trees with finitely many or nodes. For each of t, t' , we use the distributivity axiom to rewrite the tree as an or-combination of finitely many (possibly infinite) probability trees. We then establish the following.

- (a) If for every probability tree t'_i in t' there exists a corresponding tree t_i in t such that $t_i \preceq_{\text{pr/dem}}^{\text{den}} t'_i$, then we have that $t \preceq_{\text{pr/dem}}^{\text{ax}} t'$, using the Dem axiom, and step 1 above.

- (b) The tree t is equivalent in both preorders to t or k , where $k = \lambda_1 t_1 + \dots + \lambda_n t_n$ is any tree representing a convex combination of the probability trees of t . The tree k is defined using infinite combinations of pr nodes to assign the correct weight to each t_i .
- (c) Making direct use of the definition of $\mathcal{SV}_{\leq 1} \mathbb{N}$, it follows from $t \preceq_{\text{pr/dem}}^{\text{den}} t'$ that, for every probability tree t'_i of t' , there is a convex combination $k_i := \lambda_1 t_1 + \dots + \lambda_n t_n$ of probability trees of t , such that $k_i \preceq_{\text{pr/dem}}^{\text{den}} t'_i$.

To complete the argument for step 2, the tree t' has the form t'_1 or \dots or t'_m . By (c), there exist corresponding k_1, \dots, k_m . By (b), t is equivalent to t or k_1 or \dots or k_m . It now follows from (a) that $t \preceq_{\text{pr/dem}}^{\text{ax}} t'$, by the property of the k_j given by (c).

For step 3, suppose $t \preceq_{\text{pr/dem}}^{\text{den}} t'$, where t, t' are arbitrary. Take approximating sequences $t = \bigsqcup_i t_i$ and $t' = \bigsqcup_i t'_i$, where both ascending sequences are composed of finite trees.

We use Definition 18 to further restrict the approximations of t . Using the finiteness of t_i , we have $\llbracket (1-2^{-n})t_i \rrbracket_{\text{pr/dem}} \ll \llbracket t_i \rrbracket_{\text{pr/dem}}$ in the way-below relation on $\mathcal{SV}_{\leq 1} \mathbb{N}$, via the explicit characterisation of this relation in [7]. Also, $((1-2^{-i})t_i)$ is an ascending sequence of finite trees with $\bigsqcup_i (1-2^{-i})t_i = (1-2^{-\infty})t$

For every i , we have $\llbracket (1-2^{-i})t_i \rrbracket_{\text{pr/dem}} \ll \llbracket t_i \rrbracket_{\text{pr/dem}} \leq \llbracket t \rrbracket_{\text{pr/dem}} \leq \llbracket t' \rrbracket_{\text{pr/dem}}$. That is $\llbracket (1-2^{-i})t_i \rrbracket_{\text{pr/dem}} \ll \llbracket t' \rrbracket_{\text{pr/dem}}$. Since $\llbracket t' \rrbracket_{\text{pr/dem}} = \bigsqcup \llbracket t'_i \rrbracket_{\text{pr/dem}}$, it follows from the way-below property that, for every i , $\llbracket (1-2^{-i})t_i \rrbracket_{\text{pr/dem}} \leq \llbracket t'_{j_i} \rrbracket_{\text{pr/dem}}$ for some j_i , where the sequence (j_i) can be assumed strictly ascending. So, by step 2 above, $(1-2^{-i})t_i \preceq_{\text{pr/dem}}^{\text{ax}} t'_{j_i}$, for every i . Whence by admissibility, $\bigsqcup_i (1-2^{-i})t_i \preceq_{\text{pr/dem}}^{\text{ax}} \bigsqcup_i t'_{j_i}$; i.e., $(1-2^{-\infty})t \preceq_{\text{pr/dem}}^{\text{ax}} t'$. Thus $t \preceq_{\text{pr/dem}}^{\text{ax}} t'$, by Proposition 19. \blacktriangleleft

8 Related and future work

The results in this paper concern three methods of defining *basic operational preorders* on effect trees, which we claim to be a useful abstraction for defining contextual preorder for programming languages with algebraic effects. This has been verified for simple call-by-name [6] and call-by-value³ languages, but needs further substantiation.

The axiomatic approach to defining basic operational preorders in Section 6 is close in spirit to the algebraic axiomatisation of effects of Plotkin and Power [14], but with a different focus. In [14], (in)equational axiomatisations are required in order to determine a free-algebra monad modelling denotational equality of programs. Such axiomatisations have also been used to combine effects [5], and to induce a logic of effects [15]; but they have not hitherto been explicated as a method for defining contextual preorder/equivalence. In this paper, we have used infinitary Horn clause axioms between infinitary terms for this purpose, with the notion of admissible model playing an important role.

The main coincidence theorem in Section 7 has some precursors in the literature. The characterisations of $\mathcal{HV}_{\leq 1} D$ and $\mathcal{SV}_{\leq 1} D$ as free Kegelspitze in [7] can be viewed as completeness theorems for inequational axiomatisations with respect to *domains* D . In the special case $D = \mathbb{N}$, this is implied by our results, for it can be derived from Lemma 23 that the partial-order quotients of $\text{Trees}(\mathbb{N})$ by $\preceq_{\text{pr/ang}}^{\text{ax}}$ and $\preceq_{\text{pr/dem}}^{\text{ax}}$ are isomorphic to $\mathcal{HV}_{\leq 1} \mathbb{N}$ and $\mathcal{SV}_{\leq 1} \mathbb{N}$. Another related completeness result is given in [12], where inequational axioms for a simple process algebra with nondeterministic and probabilistic choice are proved complete with respect to a domain-theoretic semantics. Translated into our setting, this process algebra corresponds to *regular trees* in a signature that combines the operations or , pr with an additional prefix operation and zero constant. In [12], the semantics uses the convex powerdomain, rather than the upper \mathcal{S} and lower \mathcal{H} that we consider. In the present paper, we have not considered convex powerdomains and the associated *neutral* (as opposed to angelic or demonic) nondeterminism. However, it would be a natural extension to do so.

The main limitation we see of the present paper is the restriction throughout to *admissible* basic operational preorders. The admissibility condition plays a fundamental role in almost everything we do. It is, however, violated by some natural operational preorders; for example, for countable demonic nondeterminism. It is an open question how to incorporate such more general preorders into our theory.

References

- 1 U. Dal Lago, F. Gavazzo, and P. Blain Levy. Effectful Applicative Bisimilarity: Monads, Relators, and Howe’s Method. In *Proceedings of 32nd Annual Symposium on Logic in Computer Science*, 2017.
- 2 Jean Goubault-Larrecq. Full abstraction for non-deterministic and probabilistic extensions of PCF I: the angelic cases. *Journal of Logic and Algebraic Methods in Programming*, 84:155–184, 2015.
- 3 Jean Goubault-Larrecq. Isomorphism theorems between models of mixed choice. *Mathematical Structures in Computer Science*, 27(6):1032–1067, 2017.
- 4 Reinhold Heckmann. Probabilistic domains. In *Proceedings of CAAP ’94*, number 787 in Lecture Notes in Computer Science. Springer, 1994.
- 5 Martin Hyland, Gordon Plotkin, and John Power. Combining effects: Sum and tensor. *Theoretical Computer Science*, 357(1):70–99, 2006.
- 6 Patricia Johann, Alex Simpson, and Janis Voigtländer. A generic operational metatheory for algebraic effects. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 209–218, July 2010.
- 7 Klaus Keimel and Gordon D. Plotkin. Mixed powerdomains for probability and non-determinism. *Logical Methods in Computer Science*, 13(1), 2017.
- 8 Dexter Kozen. A probabilistic PDL. *Journal of Computer and System Sciences*, 30(2):162–178, 1985.
- 9 Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005.
- 10 Matteo Mio. Upper-expectation bisimilarity and lukasiewicz μ -calculus. In *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014*, pages 335–350, 2014.
- 11 Michael Mislove. Models supporting nondeterminism and probabilistic choice. In José Rolim, editor, *Parallel and Distributed Processing: 15 IPDPS 2000 Workshops Cancun, Mexico, May 1–5, 2000 Proceedings*, pages 993–1000. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.
- 12 Michael Mislove, Joël Ouaknine, and James Worrell. Axioms for probability and non-determinism. *Electronic Notes in Theoretical Computer Science*, 96:7–28, 2004.
- 13 Gordon Plotkin and John Power. Adequacy for algebraic effects. In *International Conference on Foundations of Software Science and Computation Structures*, pages 1–24. Springer, 2001.
- 14 Gordon Plotkin and John Power. Notions of computation determine monads. In *International Conference on Foundations of Software Science and Computation Structures*, pages 342–356. Springer, 2002.
- 15 Gordon Plotkin and Matija Pretnar. A logic of algebraic effects. In *Proceedings of the 23rd Annual Symposium on Logic in Computer Science*, 2008.
- 16 Andrea Schalk. *Algebras for generalized power constructions*. PhD thesis, TH Darmstadt, 1993.

- 17 Alex Simpson and Niels Voorneveld. Behavioural equivalence via modalities for algebraic effects. In *Proceedings of 27th European Symposium on Programming (ESOP)*. Springer, 2018.
- 18 Michael B. Smyth. Power domains and predicate transformers: A topological view. In *Proceedings of International Colloquium on Automata, Languages, and Programming ICALP 1983*, volume 154 of *Lecture Notes in Computer Science*, pages 662–675. Springer, 1983.
- 19 Sam Staton. Instances of computational effects. In *Proceedings of the 28th Annual Symposium on Logic in Computer Science*, 2013.
- 20 Regina Tix, Klaus Keimel, and Gordon Plotkin. Semantic domains for combining probability and non-determinism. *Electronic Notes in Theoretical Computer Science*, 222:3–99, 2009.
- 21 Daniele Varacca and Glynn Winskel. Distributing probability over nondeterminism. *Mathematical Structures in Computer Science*, 16(1), 2006.

Canonical Models and the Complexity of Modal Team Logic

Martin Lück

Institut für Theoretische Informatik, Leibniz Universität Hannover
Appelstraße 4, 30167 Hannover, Germany
lueck@thi.uni-hannover.de

Abstract

We study modal team logic MTL, the team-semantical extension of classical modal logic closed under Boolean negation. Its fragments, such as modal dependence, independence, and inclusion logic, are well-understood. However, due to the unrestricted Boolean negation, the satisfiability problem of full MTL has been notoriously resistant to a complexity theoretical classification.

In our approach, we adapt the notion of canonical models for team semantics. By construction of such a model, we reduce the satisfiability problem of MTL to simple model checking. Afterwards, we show that this method is optimal in the sense that MTL-formulas can efficiently enforce canonicity.

Furthermore, to capture these results in terms of computational complexity, we introduce a non-elementary complexity class, TOWER(poly), and prove that the satisfiability and validity problem of MTL are complete for it. We also show that the fragments of MTL with bounded modal depth are complete for the levels of the elementary hierarchy (with polynomially many alternations).

2012 ACM Subject Classification Theory of computation → Complexity theory and logic, Theory of computation → Logic

Keywords and phrases team semantics, modal logic, complexity, satisfiability

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.30

Related Version A full version of the paper is available at [30], <https://arxiv.org/abs/1709.05253>.

Acknowledgements The author wishes to thank Heribert Vollmer, Irena Schindler and Arne Meier, as well as the anonymous referees, for numerous helpful comments and hints.

1 Introduction

It is well-known that non-linear quantifier dependencies, such as w depending only on z in the sentence $\forall x \exists y \forall z \exists w \varphi$, cannot be expressed in first-order logic. To overcome this restriction, logics of incomplete information such as *independence-friendly logic* [19] have been studied. Later, Hodges [20] introduced *team semantics* to provide these logics with a compositional interpretation. The fundamental idea is to not consider only plain assignments to free variables, but instead whole sets of assignments, called *teams*.

In this vein, Väänänen [38] expressed non-linear quantifier dependencies by the *dependence atom* $=(x_1, \dots, x_n, y)$, which intuitively states that the values of y in the team must depend only on those of x_1, \dots, x_n . Logics with numerous other non-classical atoms such as *independence* \perp [9], *inclusion* \subseteq and *exclusion* $|$ [7] have been studied since, and have found manifold application in scientific areas such as statistics, database theory, physics, cryptography and social choice theory (see also Abramsky et al. [1]).



© Martin Lück;

licensed under Creative Commons License CC-BY

27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 30; pp. 30:1–30:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ **Table 1** Complexity landscape of propositional and modal logics of dependence (*DL), independence (*IL), inclusion (*Inc) and team logic (*TL). Entries are completeness results unless stated otherwise.

Logic	Satisfiability	Validity	References
PDL	NP	NEXPTIME	[26, 36]
MDL	NEXPTIME	NEXPTIME	[33, 11]
PIL	NP	NEXPTIME-hard, in Π_2^E	[13]
MIL	NEXPTIME	Π_2^E -hard	[23, 10]
PInc	EXPTIME	co-NP	[13]
MInc	EXPTIME	co-NEXPTIME-hard	[16]
PTL	ATIME-ALT(exp, poly)	ATIME-ALT(exp, poly)	[12, 14]
MTL _k	ATIME-ALT(exp _{k+1} , poly)	ATIME-ALT(exp _{k+1} , poly)	Theorem 6.1
MTL	TOWER(poly)	TOWER(poly)	Theorem 6.1

Team semantics have also been adapted to a range of propositional [39, 12], modal [35], and temporal logics [25]. Not only have *propositional dependence logic* PDL [39] and *modal dependence logic* MDL [35] been extensively studied, but propositional and modal logics of independence and inclusion as well [23, 13, 18, 11]. Here, the non-classical atoms, such as the dependence atom, range over whole formulas. For example, the instance $\text{=(}p_1, \dots, p_n, \Diamond \text{unsafe)}$ of a modal dependence atom may specify that the reachability of an unsafe state depends on an “access code” $p_1 \cdots p_n$ (and on nothing else), but instead of exhibiting the explicit function in question, it only stipulates the existence of such.

Most team logics lack a Boolean negation, and adding it as a connective \sim usually increases both the expressive power and the complexity tremendously. The respective extensions of propositional and modal logic are called *propositional team logic* PTL [12, 40, 14] and *modal team logic* MTL [31, 22]. By means of the negation \sim , these logics can express all the non-classical atoms mentioned above, and in fact are expressively complete for their respective class of models [22, 40]. For these reasons, they are both interesting and natural logics.

The expressive power of MTL is well-understood [22], and a complete axiomatization was presented by the author [27]. Yet the complexity of the satisfiability problem has been an open question [31, 22, 6, 15]. Recently, certain fragments of MTL with restricted negation were shown ATIME-ALT(exp, poly)-complete using the well-known filtration method [28]. In the same paper, however, it was shown that no elementary upper bound for full MTL can be established by the same approach, whereas the best known lower bound is ATIME-ALT(exp, poly)-hardness, inherited from the fragment PTL [14]. Analogously, the best known model size lower bound is – as for ordinary modal logic – exponential in the size of the formula.

Contribution. We show that MTL is complete for a non-elementary class we call TOWER(poly), which contains, roughly speaking, the problems decidable in a runtime that is a tower of nested exponentials with polynomial height. Likewise, we show that the fragments MTL_k of bounded modal depth k are complete for a class we call ATIME-ALT(exp_{k+1}, poly) and which corresponds to $(k + 1)$ -fold exponential runtime and polynomially many alternations. These results fill a long-standing gap in the active field of propositional and modal team logics (see Table 1).

In our approach, we consider *canonical* or *universal* models. Loosely speaking, a canonical model satisfies every satisfiable formula in some of its submodels, and such models have been long known for, e.g., many systems of modal logic [2]. In Section 3, we adapt this notion for modal logics with team semantics, and prove that such models exist for MTL. This enables us to reduce the satisfiability problem to simple model checking, albeit on models that are of non-elementary size with respect to $|\Phi| + k$, where Φ are the available propositional variables and k is a bound on the modal depth.

Nonetheless, this approach is essentially optimal: In Section 4 and 5, we show that MTL can, in a certain sense, *efficiently enforce* canonical models, that is, with formulas that are of size polynomial in $|\Phi| + k$. In this vein, we then obtain the matching complexity lower bounds in Section 6 by encoding computations of non-elementary length in such large models.

To the author's best knowledge, the classes $\text{ATIME-ALT}(\text{exp}_k, \text{poly})$ and $\text{TOWER}(\text{poly})$ have not explicitly been considered before. However, there are several candidates for other natural complete problems. More precisely, there exist problems in $\text{TOWER}(\text{poly})$ that are provably non-elementary, such as the satisfiability problem of separated first-order logic [37], the equivalence problem for star-free expressions [34], or the first-order theory of finite trees [4], to only name a few.

Another example is the two-variable fragment of first-order team logic, $\text{FO}^2(\sim)$. It is related to MTL in the same fashion as classical two-variable logic FO^2 to ML. Due to a reduction from MTL to $\text{FO}^2(\sim)$ (see [29]), the satisfiability and validity problems of $\text{FO}^2(\sim)$ are $\text{TOWER}(\text{poly})$ -complete problems as a corollary of this paper, while its fragments $\text{FO}_k^2(\sim)$ of bounded quantifier rank k are $\text{ATIME-ALT}(\text{exp}_{k+1}, \text{poly})$ -hard.

Due to space constraints, several technical proofs (which are marked with (\star)) are omitted or only sketched. They can be found in the full version of this paper [30].

2 Preliminaries

The power set of a set X is $\mathfrak{P}(X)$. We let $|X|$ denote the length of the encoding of a formula or structure X . The sets of all satisfiable resp. valid formulas of a given logic \mathcal{L} are $\text{SAT}(\mathcal{L})$ and $\text{VAL}(\mathcal{L})$, respectively.

We assume the reader to be familiar with alternating Turing machines [3]. We assume all reductions in this paper implicitly as logspace reductions \leq_m^{\log} .

The class $\text{ATIME-ALT}(\text{exp}, \text{poly})$ contains the problems decidable by an alternating Turing machine in time $2^{p(n)}$ with $p(n)$ alternations, for a polynomial p . It is a natural class that has several complete problems [13, 21, 14]. Here, we generalize it to capture the *elementary hierarchy* $\text{exp}_k(n)$, defined by $\text{exp}_0(n) := n$ and $\text{exp}_{k+1}(n) := 2^{\text{exp}_k(n)}$.

► **Definition 2.1.** For $k \geq 0$, $\text{ATIME-ALT}(\text{exp}_k, \text{poly})$ is the class of problems decided by an alternating Turing machine with at most $p(n)$ alternations and runtime at most $\text{exp}_k(p(n))$, for a polynomial p .

Note that setting $k = 0$ or $k = 1$ yields the classes PSPACE and $\text{ATIME-ALT}(\text{exp}, \text{poly})$, respectively [3]. If k is replaced by a polynomial instead, we obtain the following class.

► **Definition 2.2.** $\text{TOWER}(\text{poly})$ is the class of problems that are decided by a deterministic Turing machine in time $\text{exp}_{p(n)}(1)$ for some polynomial p .

Note that a similar class, TOWER , is defined by replacing p by an arbitrary elementary function [32]. By contrast, to the author's best knowledge, $\text{TOWER}(\text{poly})$ has not yet been explicitly studied. The reader may verify that both $\text{ATIME-ALT}(\text{exp}_k, \text{poly})$ and $\text{TOWER}(\text{poly})$ are closed under polynomial time reductions (and hence also \leq_m^{\log}).

Modal team logic

We fix a countably infinite set \mathcal{PS} of propositional symbols. *Modal team logic* MTL, introduced by Müller [31], extends classical modal logic ML as in the following grammar, where φ denotes an MTL-formula, α an ML-formula, and $p \in \mathcal{PS}$.

$$\begin{aligned}\varphi &::= \sim\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \alpha \\ \alpha &::= \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \Box\alpha \mid \Diamond\alpha \mid p \mid \top\end{aligned}$$

The set of propositional variables occurring in $\varphi \in \text{MTL}$ is denoted by $\text{Prop}(\varphi)$.

We use the common abbreviations $\perp := \neg\top$, $\alpha \rightarrow \beta := \neg\alpha \vee \beta$ and $\alpha \leftrightarrow \beta := (\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$. For easier distinction, we have classical formulas denoted by $\alpha, \beta, \gamma, \dots$ and reserve $\varphi, \psi, \vartheta, \dots$ for general team-logical formulas.

The modal depth $\text{md}(\theta)$ of an (ML or MTL) formula θ is recursively defined:

$$\begin{aligned}\text{md}(p) &:= \text{md}(\top) := 0 \\ \text{md}(\sim\varphi) &:= \text{md}(\neg\varphi) := \text{md}(\varphi) \\ \text{md}(\varphi \wedge \psi) &:= \text{md}(\varphi \vee \psi) := \max\{\text{md}(\varphi), \text{md}(\psi)\} \\ \text{md}(\Diamond\varphi) &:= \text{md}(\Box\varphi) := \text{md}(\varphi) + 1\end{aligned}$$

ML_k and MTL_k are the fragments of ML and MTL with modal depth $\leq k$, respectively. If the propositions are restricted to a fixed set $\Phi \subseteq \mathcal{PS}$ as well, then the fragment is denoted by ML_k^Φ , or MTL_k^Φ , respectively.

Let $\Phi \subseteq \mathcal{PS}$ be a finite set of propositions. A *Kripke structure* (over Φ) is a tuple $\mathcal{K} = (W, R, V)$, where W is a set of *worlds*, (W, R) is a directed graph, and $V: \Phi \rightarrow \mathfrak{P}(W)$ is the *valuation*. Occasionally, by slight abuse of notation, we use the mapping $V^{-1}: W \rightarrow \mathfrak{P}(\Phi)$ defined by $V^{-1}(w) := \{p \in \Phi \mid w \in V(p)\}$ instead of V , i.e., the set of propositions that are true in a given world.

If $w \in W$, then (\mathcal{K}, w) is called *pointed structure*. ML is evaluated on pointed structures in the classical Kripke semantics. By contrast, MTL is evaluated on pairs (\mathcal{K}, T) , called *structures with teams*, where $T \subseteq W$ is called *team* (in \mathcal{K}).

Every team T has an *image* $RT := \{v \mid w \in T, (w, v) \in R\}$, and if $w \in W$, we simply write Rw instead of $R\{w\}$. $R^i T$ is inductively defined as $R^0 T := T$ and $R^{i+1} T := RR^i T$. A *successor team* of T is a team S such that $S \subseteq RT$ and $T \subseteq R^{-1} S$, where $R^{-1} := \{(v, w) \mid (w, v) \in R\}$. Intuitively, S is formed by picking at least one successor of every world in T .

The semantics of MTL can now be defined as follows.¹

$$\begin{aligned}(\mathcal{K}, T) \models \alpha &\Leftrightarrow \forall w \in T: (\mathcal{K}, w) \models \alpha \text{ if } \alpha \in \text{ML}, \text{ and otherwise as} \\ (\mathcal{K}, T) \models \sim\psi &\Leftrightarrow (\mathcal{K}, T) \not\models \psi, \\ (\mathcal{K}, T) \models \psi \wedge \theta &\Leftrightarrow (\mathcal{K}, T) \models \psi \text{ and } (\mathcal{K}, T) \models \theta, \\ (\mathcal{K}, T) \models \psi \vee \theta &\Leftrightarrow \exists S, U \subseteq T \text{ such that } T = S \cup U, (\mathcal{K}, S) \models \psi, \text{ and } (\mathcal{K}, U) \models \theta, \\ (\mathcal{K}, T) \models \Diamond\psi &\Leftrightarrow (\mathcal{K}, S) \models \psi \text{ for some successor team } S \text{ of } T, \\ (\mathcal{K}, T) \models \Box\psi &\Leftrightarrow (\mathcal{K}, RT) \models \psi.\end{aligned}$$

We often omit \mathcal{K} and write $T \models \varphi$ or $w \models \alpha$.

¹ Often, the ‘‘atoms’’ of MTL are restricted to literals $p, \neg p$ instead of ML-formulas α . However, this implies a restriction to formulas in negation normal form, and both definitions are equivalent due to the *flatness* property of ML (cf. [22, Proposition 2.2]).

An MTL-formula φ is *satisfiable* if it is true in some structure with team over $\text{Prop}(\varphi)$, which is then called a *model* of φ . Analogously, φ is *valid* if it is true in every structure with team over $\text{Prop}(\varphi)$.

Note that the empty team is usually excluded in the above definition, since most \sim -free logics with team semantics have the *empty team property*, i.e., the empty team trivially satisfying every formula [35, 23, 18]. However, this distinction is unnecessary for MTL: φ is satisfiable iff $\top \vee \varphi$ is true in some non-empty team², and φ is true in some non-empty team iff $\sim \perp \wedge \varphi$ is satisfiable.

The modality-free fragment MTL_0 syntactically coincides with *propositional team logic* PTL [12, 14, 40]. The usual interpretations of the latter, i.e., sets of Boolean assignments, can easily be represented as teams in Kripke structures. For this reason, we identify PTL and MTL_0 in this paper.

Note that the connectives \vee , \rightarrow and \neg are not the usual truth-functional connectives on the level of teams, i.e., Boolean disjunction, implication and negation. The exception are singleton teams, on which team semantics and Kripke semantics coincide. Using \wedge and \sim however, we can define Boolean disjunction $\varphi_1 \oplus \varphi_2 := \sim(\sim\varphi_1 \wedge \sim\varphi_2)$ and implication $\varphi_1 \rightarrow \varphi_2 := \sim\varphi_1 \oplus \varphi_2$.

The notation $\Box^i \varphi$ is defined via $\Box^0 \varphi := \varphi$ and $\Box^{i+1} \varphi := \Box \Box^i \varphi$, and analogously for $\Diamond^i \varphi$. To state that at least one element of a team satisfies $\alpha \in \text{ML}$, we write $\mathbf{E}\alpha := \sim\neg\alpha$. That the truth value of α is constant in the team is expressed by the *constancy atom* $\mathbf{c}(\alpha) := \alpha \oplus \neg\alpha$.

The well-known *bisimulation* relation \equiv_k^Φ fundamentally defines the expressive power of modal logic [2] and plays a key role in our results.

► **Definition 2.3.** Let $\Phi \subseteq \mathcal{PS}$ and $k \geq 0$. For $i \in \{1, 2\}$, let (\mathcal{K}_i, w_i) be a pointed structure, where $\mathcal{K}_i = (W_i, R_i, V_i)$. Then (\mathcal{K}_1, w_1) and (\mathcal{K}_2, w_2) are (Φ, k) -bisimilar, in symbols $(\mathcal{K}_1, w_1) \equiv_k^\Phi (\mathcal{K}_2, w_2)$, if

- $\forall p \in \Phi: w_1 \in V_1(p) \Leftrightarrow w_2 \in V_2(p)$,
- and if $k > 0$,
 - $\forall v_1 \in R_1 w_1: \exists v_2 \in R_2 w_2: (\mathcal{K}_1, v_1) \equiv_{k-1}^\Phi (\mathcal{K}_2, v_2)$ (*forward condition*),
 - $\forall v_2 \in R_2 w_2: \exists v_1 \in R_1 w_1: (\mathcal{K}_1, v_1) \equiv_{k-1}^\Phi (\mathcal{K}_2, v_2)$ (*backward condition*).

The notion of bisimulation was also lifted to team semantics by Hella et al. [17]:

► **Definition 2.4** (cf. [17, 23, 22]). Let $\Phi \subseteq \mathcal{PS}$ and $k \geq 0$. For $i \in \{1, 2\}$, let (\mathcal{K}_i, T_i) be a structure with team. Then (\mathcal{K}_1, T_1) and (\mathcal{K}_2, T_2) are (Φ, k) -team-bisimilar, written $(\mathcal{K}_1, T_1) \equiv_k^\Phi (\mathcal{K}_2, T_2)$, if

- $\forall w_1 \in T_1: \exists w_2 \in T_2: (\mathcal{K}_1, w_1) \equiv_k^\Phi (\mathcal{K}_2, w_2)$,
- $\forall w_2 \in T_2: \exists w_1 \in T_1: (\mathcal{K}_1, w_1) \equiv_k^\Phi (\mathcal{K}_2, w_2)$.

If no confusion can arise, we will also refer to teams T_1, T_2 that are (Φ, k) -team-bisimilar simply as (Φ, k) -bisimilar. The proofs of the following propositions are straightforward and can be found in the full version [30].

► **Proposition 2.5** (*). Let $\Phi \subseteq \mathcal{PS}$ be finite, and $k \geq 0$. For $i \in \{1, 2\}$, let (\mathcal{K}_i, w_i) be a pointed structure, where $\mathcal{K}_i = (W_i, R_i, V_i)$. Then the following statements are equivalent:

1. $\forall \alpha \in \text{ML}_k^\Phi: (\mathcal{K}_1, w_1) \models \alpha \Leftrightarrow (\mathcal{K}_2, w_2) \models \alpha$,
2. $(\mathcal{K}_1, w_1) \equiv_k^\Phi (\mathcal{K}_2, w_2)$,

² In team semantics, $\top \vee \varphi$ is not tautologically true, but rather existentially quantifies a subteam.

3. $(\mathcal{K}_1, \{w_1\}) \equiv_k^\Phi (\mathcal{K}_2, \{w_2\})$.

Moreover, if $k > 0$, they are equivalent to:

4. $(\mathcal{K}_1, w_1) \equiv_0^\Phi (\mathcal{K}_2, w_2)$ and $(\mathcal{K}_1, R_1 w_1) \equiv_{k-1}^\Phi (\mathcal{K}_2, R_2 w_2)$.

As a result, the *forward* and *backward* condition from Definition 2.3 can be equivalently stated in terms of team-bisimilarity of the respective images. On the level of teams, a similar characterization holds:

► **Proposition 2.6** (\star). *Let $\Phi \subseteq \mathcal{PS}$ be finite, and $k \geq 0$. Let (\mathcal{K}_i, T_i) be a structure with team for $i \in \{1, 2\}$. Then the following statements are equivalent:*

1. $\forall \alpha \in \text{ML}_k^\Phi : (\mathcal{K}_1, T_1) \models \alpha \Leftrightarrow (\mathcal{K}_2, T_2) \models \alpha$,
2. $\forall \varphi \in \text{MTL}_k^\Phi : (\mathcal{K}_1, T_1) \models \varphi \Leftrightarrow (\mathcal{K}_2, T_2) \models \varphi$,
3. $(\mathcal{K}_1, T_1) \equiv_k^\Phi (\mathcal{K}_2, T_2)$,

3 Types and canonical models

Many modal logics admit a “universal” model, also called *canonical model*. Given a canonical model \mathcal{K} , and a satisfiable formula (or set of formulas), the latter is then also true in some point of \mathcal{K} . See also Blackburn et al. [2, Section 4.2] for the explicit construction of such a model for ML.

Unfortunately, a canonical model for ML is necessarily infinite, and consequently impractical for complexity theoretic considerations. Instead, we define (Φ, k) -*canonical models* for finite $\Phi \subseteq \mathcal{PS}$ and $k \in \mathbb{N}$, which are then proved canonical for the fragment ML_k^Φ . However, by Proposition 2.5, the size of a (Φ, k) -canonical model is necessarily at least the number of equivalence classes of \equiv_k^Φ .

The equivalence classes of \equiv_k^Φ are proper classes. However, speaking about teams would require *sets* of such classes. For this reason, we inductively define *types*, which properly reflect bisimulation, but exist as sets. We usually refer to types as τ .

► **Definition 3.1.** Let $\Phi \subseteq \mathcal{PS}$ be finite. The set of (Φ, k) -*types*, written Δ_k^Φ , is defined inductively as $\Delta_0^\Phi := \mathfrak{P}(\Phi) \times \{\emptyset\}$ and $\Delta_{k+1}^\Phi := \mathfrak{P}(\Phi) \times \mathfrak{P}(\Delta_k^\Phi)$.

Let $(\mathcal{K}, w) = (W, R, V, w)$ be a pointed structure. Then its (Φ, k) -type, written $\llbracket \mathcal{K}, w \rrbracket_k^\Phi$, is the unique $(\Phi', \Delta') \in \Delta_k^\Phi$ such that $V^{-1}(w) = \Phi'$ and, in case $k > 0$, additionally $\forall \tau' \in \Delta_{k-1}^\Phi : \tau' \in \Delta' \Leftrightarrow \exists v \in R w : \llbracket \mathcal{K}, v \rrbracket_{k-1}^\Phi = \tau'$.

Given a team T in \mathcal{K} , the types in T are denoted by $\llbracket \mathcal{K}, T \rrbracket_k^\Phi := \{\llbracket \mathcal{K}, w \rrbracket_k^\Phi \mid w \in T\}$.

For a type $\tau = (\Phi', \Delta')$, we define shorthands $\Phi_\tau := \Phi'$ and $\mathcal{R}\tau := \Delta'$.

Intuitively, the first component Φ_τ consists of the propositions which any model of type τ must satisfy in its root, and $\mathcal{R}\tau$ is the set of types which any model of type τ must contain in the image of its root. Roughly speaking, Φ_τ reflects the first condition of Definition 2.3, propositional equivalence, while $\mathcal{R}\tau$ reflects the forward and backward conditions.

Every type $\tau \in \Delta_k^\Phi$ is satisfiable in the sense that there is at least one pointed structure (\mathcal{K}, w) such that $\llbracket \mathcal{K}, w \rrbracket_k^\Phi = \tau$.

The following assertions are straightforward to prove by induction, and ascertain that types properly reflect the notion of bisimulation.

► **Proposition 3.2** (\star). *Let $\Phi \subseteq \mathcal{PS}$ be finite and $k \geq 0$. Then $(\mathcal{K}, w) \equiv_k^\Phi (\mathcal{K}', w')$ if and only if $\llbracket \mathcal{K}, w \rrbracket_k^\Phi = \llbracket \mathcal{K}', w' \rrbracket_k^\Phi$, and $(\mathcal{K}, T) \equiv_k^\Phi (\mathcal{K}', T')$ if and only if $\llbracket \mathcal{K}, T \rrbracket_k^\Phi = \llbracket \mathcal{K}', T' \rrbracket_k^\Phi$.*

We are now ready to state the formal definition of canonicity:

► **Definition 3.3.** A structure with team (\mathcal{K}, T) is (Φ, k) -canonical if $\llbracket \mathcal{K}, T \rrbracket_k^\Phi = \Delta_k^\Phi$.

In the following, we often omit Φ and \mathcal{K} and write only $\llbracket w \rrbracket_k$ or $\llbracket T \rrbracket_k$, and simply say that T is (Φ, k) -canonical if \mathcal{K} is clear.

It is a standard result that for every Φ and $k \geq 0$ there exists a (Φ, k) -canonical model (cf. Blackburn et al. [2]), or in other words, that the logic ML_k^Φ admits canonical models.

Canonical models in team semantics

The logic MTL is significantly more expressive than ML [22]. Nonetheless, we will show that every satisfiable MTL_k^Φ -formula can be satisfied in a (Φ, k) -canonical model. In other words, the canonical models of MTL_k^Φ and ML_k^Φ actually coincide.

► **Theorem 3.4.** Let (\mathcal{K}, T) be (Φ, k) -canonical and $\varphi \in \text{MTL}_k^\Phi$. Then φ is satisfiable if and only if $(\mathcal{K}, T') \models \varphi$ for some $T' \subseteq T$.

Proof. Assume (\mathcal{K}, T) and φ are as above. As the direction from right to left is trivial, suppose that φ is satisfiable, i.e., has a model $(\hat{\mathcal{K}}, \hat{T})$. As a team in \mathcal{K} that satisfies φ , we define

$$T' := \left\{ w \in T \mid \llbracket \mathcal{K}, w \rrbracket_k^\Phi \in \llbracket \hat{\mathcal{K}}, \hat{T} \rrbracket_k^\Phi \right\}.$$

By Proposition 2.6 and 3.2, it suffices to prove $\llbracket \hat{\mathcal{K}}, \hat{T} \rrbracket_k^\Phi = \llbracket \mathcal{K}, T' \rrbracket_k^\Phi$. Moreover, the direction “ \supseteq ” is clear by definition. As T is (Φ, k) -canonical, for every $\tau \in \llbracket \hat{\mathcal{K}}, \hat{T} \rrbracket_k^\Phi$ there exists a world $w \in T$ of type τ . Consequently, $\llbracket \hat{\mathcal{K}}, \hat{T} \rrbracket_k^\Phi \subseteq \llbracket \mathcal{K}, T' \rrbracket_k^\Phi$. ◀

How large is a (Φ, k) -canonical model at least? The number of types can be written via the function exp_k^* , which is defined by

$$\text{exp}_0^*(n) := n, \quad \text{exp}_{k+1}^*(n) := n \cdot 2^{\text{exp}_k^*(n)}.$$

Observe that this function resembles $\text{exp}_k(n)$ (cf. p. 3) except for an additional factor of n in every “level” of the nested exponents. By Definition 3.1, we immediately obtain:

► **Proposition 3.5.** $|\Delta_k^\Phi| = \text{exp}_k^*(2^{|\Phi|})$ for all $k \geq 0$ and finite $\Phi \subseteq \mathcal{PS}$.

Next, we present an algorithm that solves the satisfiability and validity problems of MTL and its fragments MTL_k by computing a canonical model. Let us first explicate this construction in a lemma.

► **Lemma 3.6.** There is an algorithm that, given $\Phi \subseteq \mathcal{PS}$ and $k \geq 0$, computes a (Φ, k) -canonical model in time polynomial in $|\Delta_k^\Phi|$.

Proof. Let $\mathcal{K} = (W, R, V)$ be the computed structure. The idea is to construct sets $L_0 \cup L_1 \cup \dots \cup L_k =: W$ of worlds in stage-wise manner such that L_i is (Φ, i) -canonical.

For L_0 , we simply add a world w for each $\Phi' \in \mathfrak{P}(\Phi)$ such that $V^{-1}(w) = \Phi'$.

For $i > 0$, we iterate over all $L' \in \mathfrak{P}(L_{i-1})$ and $\Phi' \in \mathfrak{P}(\Phi)$ and insert a new world w into L_i such that $Rw = L'$ and again $V^{-1}(w) = \Phi'$. An inductive argument shows that L_i is (Φ, i) -canonical for all $i \in \{0, \dots, k\}$. As $k \leq |\Delta_k^\Phi|$, and each L_i is constructed in time polynomial in $|\Delta_i^\Phi| \leq |\Delta_k^\Phi|$, the overall runtime is polynomial in $|\Delta_k^\Phi|$. ◀

The next lemma allows, roughly speaking, to replace a polynomial of exp_k^* by simply exp_k , with only polynomial blowup in its argument.

► **Lemma 3.7.** *For every polynomial p there is a polynomial q such that $p(\exp_k^*(n)) \leq \exp_k(q((k+1) \cdot n))$ for all $k \geq 0$ and $n \geq 1$.*

Proof. For $p(n)$ bounded by cn^d , with $c, d \in \mathbb{N}$, let $q(n) := cnd^d + c$ (cf. [30]). ◀

► **Theorem 3.8.** *SAT(MTL_k) and VAL(MTL_k) are in ATIME-ALT(\exp_{k+1} , poly).*

Proof. Consider the following algorithm. Let $\varphi \in \text{MTL}_k$ be the input, $n := |\varphi|$, and $\Phi := \text{Prop}(\varphi)$. Construct deterministically, as in Lemma 3.6, a (Φ, k) -canonical structure $(\mathcal{K}, T) = (W, R, V, T)$ in time $p(|\Delta_k^\Phi|)$ for a polynomial p .

By a result of Müller [31], the model checking problem of MTL is solvable by an alternating Turing machine that has runtime polynomial in $|\varphi| + |\mathcal{K}|$, and alternations polynomial in $|\varphi|$. We call this algorithm as a subroutine: by Theorem 3.4, φ is satisfiable (resp. valid) if and only if for at least one team (resp. all teams) $T' \subseteq T$ we have $(\mathcal{K}, T') \models \varphi$. Equivalently, this is the case if and only if (\mathcal{K}, T) satisfies $\top \vee \varphi$ (resp. $\sim(\top \vee \sim\varphi)$).

Let us turn to the overall runtime. \mathcal{K} is constructed in time polynomial in $|\Delta_k^\Phi| = \exp_k^*(2^{|\Phi|}) \leq \exp_{k+1}^*(|\Phi|) \leq \exp_{k+1}^*(n)$. The subsequent model checking runs in time polynomial in $|\mathcal{K}| + n$, and hence polynomial in $\exp_{k+1}^*(n)$ as well. By Lemma 3.7, we obtain a total runtime of $\exp_{k+1}(q((k+2) \cdot n))$ for a polynomial q . ◀

The upper bound for MTL can be proved similarly, since $k := \text{md}(\varphi)$ is polynomial in $|\varphi|$. Moreover, the alternations can be eliminated with additional exponential blowup.

► **Corollary 3.9.** *SAT(MTL) and VAL(MTL) are in TOWER(poly).*

4 Efficiently expressing bisimilarity

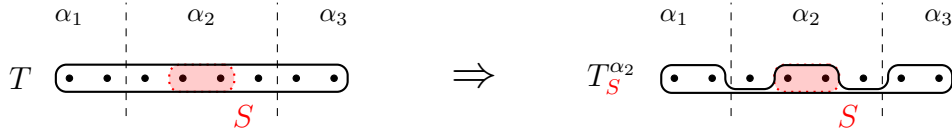
Kontinen et al. [22] proved that MTL is expressively complete up to bisimulation, i.e., it can define every property of teams that is closed under $\rightleftharpoons_k^\Phi$ for some finite Φ and k . Two such team properties are in fact (Φ, k) -bisimilarity itself – in the sense that two worlds in a team have the same type – as well as (Φ, k) -canonicity. Consequently, these properties are defined by MTL_k^Φ -formulas. However, by a simple counting argument, formulas defining arbitrary team properties are of non-elementary size w. r. t. Φ and k in the worst case.

From now on, we always assume some finite $\Phi \subseteq \mathcal{PS}$ and omit it in the notation, i.e., we write k -canonicity, k -bisimilarity, \rightleftharpoons_k , and so on.

In this section, we present an “approximation” (in a sense we clarify below) of k -bisimilarity that can be expressed in a formula χ_k that is of polynomial size in Φ and k . Likewise, in Section 5 we present a formula canon_k of polynomial size that expresses k -canonicity. Finally, in Section 6, we apply χ_k and canon_k in order to prove the lower bound for Corollary 3.9, i.e., TOWER(poly)-hardness of SAT(MTL) and VAL(MTL) (and an analogous result for Theorem 3.8). Here, the idea is to enforce a sufficiently large structure with canon_k and then to encode a non-elementary computation into it. Clearly, χ_k and canon_k being polynomial in Φ and k is crucial for the reduction.

Scopes

To implement k -bisimilarity, we pursue a recursive approach. In the spirit of Proposition 2.5, the $(k+1)$ -bisimilarity of two points w, v is expressed in terms of k -team-bisimilarity of Rw and Rv . Conversely, to verify k -team-bisimilarity of Rw and Rv , we proceed analogously to the *forward* and *backward* conditions of Definition 2.3 and reduce the problem to checking k -bisimilarity of pairs of points in Rw and Rv .



■ **Figure 1** Example of subteam selection in the scope α_2 .

A clear obstacle is that MTL cannot speak about two teams Rw, Rv simultaneously, let alone check for bisimilarity. Instead, we consider a team that is the “marked union” of Rw and Rv .

More generally, for all formulas $\alpha \in \text{ML}$ we define the subteam $T_\alpha := \{w \in T \mid w \models \alpha\}$. The corresponding “decoding” operator

$$\alpha \mapsto \varphi := \neg\alpha \vee (\alpha \wedge \varphi)$$

was considered by Kontinen and Nurmi [24] and Galliani [8]. Here, $\alpha \mapsto \varphi$ is true in T if and only if $T_\alpha \models \varphi$.

Now, instead of defining an n -ary relation on teams, a formula φ can define a unary relation – a team property – parameterized by “marker formulas” $\alpha_1, \dots, \alpha_n \in \text{ML}$. We emphasize this by writing $\varphi(\alpha_1, \dots, \alpha_n)$.

This is the “approximation” mentioned earlier: In order to compare Rw and Rv , we require that $Rw = T_\alpha$ and $Rv = T_\beta$ for some team T and distinct $\alpha, \beta \in \text{ML}$. It will be useful if the “markers” are invariant under traversing edges in the structure:

► **Definition 4.1.** Let $\mathcal{K} = (W, R, V)$ be a Kripke structure. A formula $\alpha \in \text{ML}$ is called a *scope (in \mathcal{K})* if $(w, v) \in R$ implies $w \models \alpha \Leftrightarrow v \models \alpha$. Two scopes α, β are called *disjoint (in \mathcal{K})* if W_α and W_β are disjoint.

In order to avoid interference, we always assume that scopes are formulas in $\text{ML}_0^{\mathcal{P}S \setminus \Phi}$, i.e., they are always purely propositional and do not contain propositions from Φ .

It is desirable to be able to speak about subteams in a specific scope. Formally, if S is a team, let $T_S^\alpha := T_{\neg\alpha} \cup (T_\alpha \cap S)$. For singletons $\{w\}$, we simply write T_w^α instead of $T_{\{w\}}^\alpha$. Intuitively, T_S^α is obtained from T by “shrinking” the subteam T_α down to S without impairing $T \setminus T_\alpha$ (see Figure 1 for an example).

The following observations are straightforward:

► **Proposition 4.2** ([30]). *Let α, β be disjoint scopes and S, U, T teams in a Kripke structure $\mathcal{K} = (W, R, V)$. Then the following laws hold:*

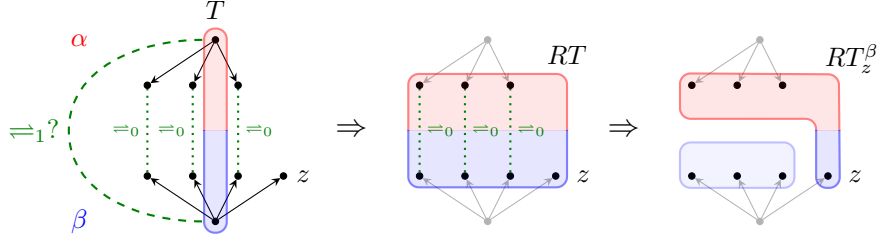
1. *Distributive laws:* $(T \cap S)_\alpha = T_\alpha \cap S = T \cap S_\alpha = T_\alpha \cap S_\alpha$ and $(T \cup S)_\alpha = T_\alpha \cup S_\alpha$.
2. *Disjoint selection commutes:* $(T_S^\alpha)_U^\beta = (T_U^\beta)_S^\alpha$.
3. *Disjoint selection is independent:* $((T_S^\alpha)_U^\beta)_\alpha = T_\alpha \cap S$.
4. *Image and scope commute:* $(RT)_\alpha = (R(T_\alpha))_\alpha = R(T_\alpha)$.
5. *Selection propagates:* If $S \subseteq T$, then $R(T_S^\alpha) = (RT)_{RS}^\alpha$.

Accordingly, we write $R^i T_\alpha$ instead of $(R^i T)_\alpha$ or $R^i(T_\alpha)$ and $T_{S_1, S_2}^{\alpha_1, \alpha_2}$ for $(T_{S_1}^{\alpha_1})_{S_2}^{\alpha_2}$.

Subteam quantifiers

We refer to the following abbreviations as *subteam quantifiers*, where $\alpha \in \text{ML}$:

$$\begin{aligned} \exists_\alpha^\subseteq \varphi &:= \alpha \vee \varphi & \forall_\alpha^\subseteq \varphi &:= \sim \exists_\alpha^\subseteq \sim \varphi \\ \exists_\alpha^1 \varphi &:= \exists_\alpha^\subseteq [\text{E}\alpha \wedge \forall_\alpha^\subseteq (\text{E}\alpha \rightarrow \varphi)] & \forall_\alpha^1 \varphi &:= \sim \exists_\alpha^1 \sim \varphi \end{aligned}$$



■ **Figure 2** As z violates the *backward* condition, $\chi_0^*(\alpha, \beta)$ detects a \Rightarrow_0 -free subteam, refuting $\exists_\alpha^1 \exists_\beta^1 \chi_0(\alpha, \beta)$.

Intuitively, they quantify over subteams $S \subseteq T_\alpha$ (in case of $\exists_\alpha^{\subseteq} / \forall_\alpha^{\subseteq}$) or over worlds $w \in T_\alpha$ (for $\exists_\alpha^1 / \forall_\alpha^1$), and require that the shrunk team T_S^α resp. T_w^α satisfies φ .

► **Proposition 4.3** (\star). $\exists_\alpha^{\subseteq}, \forall_\alpha^{\subseteq}, \exists_\alpha^1, \forall_\alpha^1$ have the following semantics:

$$\begin{aligned} T \models \exists_\alpha^{\subseteq} \varphi &\Leftrightarrow \exists S \subseteq T_\alpha : T_S^\alpha \models \varphi & T \models \exists_\alpha^1 \varphi &\Leftrightarrow \exists w \in T_\alpha : T_w^\alpha \models \varphi \\ T \models \forall_\alpha^{\subseteq} \varphi &\Leftrightarrow \forall S \subseteq T_\alpha : T_S^\alpha \models \varphi & T \models \forall_\alpha^1 \varphi &\Leftrightarrow \forall w \in T_\alpha : T_w^\alpha \models \varphi \end{aligned}$$

Proof sketch. Here, we sketch only the existential cases, as the universal ones work dually. The formula $\exists_\alpha^{\subseteq} \varphi := \alpha \vee \varphi$ allows to split T into subteams $U_1 \subseteq T_\alpha$ and U_2 , where $U_2 \models \varphi$. As U_2 must contain $T_{-\alpha}$, clearly it is of the form T_S^α for some S . Conversely, every team of the form T_S^α induces a splitting of T into U_1, U_2 as above.

The singleton quantifier, \exists_α^1 , states that for some non-empty $U \subseteq T_\alpha$ it holds that $T_S^\alpha \models \varphi$ for every non-empty $S \subseteq U$. This is equivalent to $T_U^\alpha \models \varphi$ being true for some singleton $U \subseteq T_\alpha$. ◀

Implementing bisimulation

Finally, we have all ingredients to implement k -bisimulation in the following inductive manner:

$$\chi_0(\alpha, \beta) := (\alpha \vee \beta) \leftrightarrow \bigwedge_{p \in \Phi} \Rightarrow(p)$$

$$\chi_{k+1}(\alpha, \beta) := \chi_0(\alpha, \beta) \wedge \Box \chi_k^*(\alpha, \beta)$$

$$\chi_k^*(\alpha, \beta) := (\neg\alpha \wedge \neg\beta) \otimes \left(\mathbf{E}\alpha \wedge \mathbf{E}\beta \wedge \sim [(\alpha \otimes \beta) \vee (\mathbf{E}\alpha \wedge \mathbf{E}\beta \wedge \sim \exists_\alpha^1 \exists_\beta^1 \chi_k(\alpha, \beta))] \right)$$

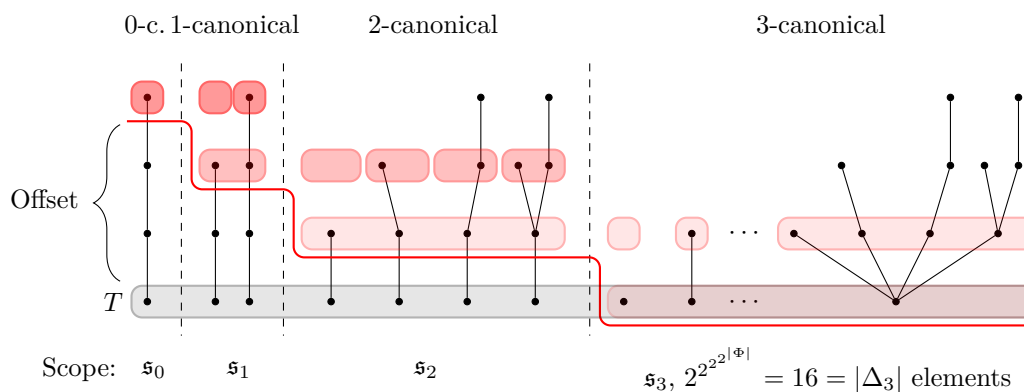
Here, \leftrightarrow is defined as on p. 9. Let us prove that these formulas define bisimulation:

► **Theorem 4.4** (\star). Let $k \geq 0$. For all Kripke structures \mathcal{K} , teams T in \mathcal{K} , disjoint scopes α, β in \mathcal{K} , and points $w \in T_\alpha$ and $v \in T_\beta$ it holds:

$$\begin{aligned} T_{w,v}^{\alpha,\beta} \models \chi_k(\alpha, \beta) &\text{ if and only if } w \Rightarrow_k v, \\ T \models \chi_k^*(\alpha, \beta) &\text{ if and only if } T_\alpha \Rightarrow_k T_\beta. \end{aligned}$$

Moreover, both $\chi_k(\alpha, \beta)$ and $\chi_k^*(\alpha, \beta)$ are MTL_k -formulas that are constructible in space $\mathcal{O}(\log(k + |\Phi| + |\alpha| + |\beta|))$.

Proof sketch. By induction on k . First, the formula $\chi_0(\alpha, \beta)$ expresses $w \Rightarrow_0 v$ when evaluated on a team $T_{w,v}^{\alpha,\beta}$. By the semantics of \leftrightarrow , $\chi_0(\alpha, \beta)$ is true if and only if $\{w, v\} \models \Rightarrow(p)$



■ **Figure 3** Visualization of the 3-staircase for $\Phi = \emptyset$, where the subteam T_{s_i} is i -canonical with offset $3 - i$.

for all $p \in \Phi$. By definition of $\equiv(\cdot)$, then $w \vDash p \Leftrightarrow v \vDash p$ for all $p \in \Phi$, i.e., $w \equiv_0 v$. For χ_{k+1} , recall that $w \equiv_{k+1} v$ is equivalent to $w \equiv_0 v$ and $Rw \equiv_k Rv$. Consequently, χ_{k+1} defines $(k+1)$ -bisimilarity on points under the assumption that χ_k^* defines k -bisimilarity on teams.

Finally, $\chi_k^*(\alpha, \beta)$ checks $T_\alpha \equiv_k T_\beta$ as follows. If at least one of these teams is empty, then it is easy to see that χ_k^* acts correctly. For non-empty T_α and T_β , the idea is to isolate any single point $z \in T_\alpha \cup T_\beta$ that serves as a *counter-example* against $\llbracket T_\alpha \rrbracket_k = \llbracket T_\beta \rrbracket_k$ by, say, $\llbracket z \rrbracket_k \in \llbracket T_\beta \rrbracket_k \setminus \llbracket T_\alpha \rrbracket_k$. We erase $T_\beta \setminus \{z\}$ from T using the disjunction \vee , as $T_\beta \setminus \{z\} \vDash \alpha \otimes \beta$. The remaining team is exactly T_z^β , in which $\exists^1_\alpha \exists^1_\beta \chi_k(\alpha, \beta)$ fails (see Figure 2). The case $\llbracket z \rrbracket_k \in \llbracket T_\alpha \rrbracket_k \setminus \llbracket T_\beta \rrbracket_k$ is detected analogously. Moreover, the formulas can be constructed in logspace in a straightforward manner, and $\text{md}(\chi_k) = \text{md}(\chi_k^*) = k$. ◀

Let us again stress that χ_k implements only an *approximation* of \equiv_k , as it relies on scopes to be labeled in the structure correctly.

5 Enforcing a canonical model

As discussed before, we now aim at constructing an MTL_k -formula that is satisfiable but permits *only* k -canonical models. For $k = 0$, Hannula et al. [13] defined the PTL-formula

$$\max(X) := \sim \bigvee_{x \in X} \equiv(x)$$

and proved that $T \vDash \max(\Phi)$ if and only if T is 0-canonical, i.e., contains all Boolean assignment over Φ . We generalize this for all k , i.e., construct a satisfiable formula canon_k that has only k -canonical models.

Staircase models

Our approach is to express k -canonicity by inductively enforcing i -canonical sets of worlds for $i = 0, \dots, k$ located in different “height” inside the model. For this purpose, we employ distinct scopes s_0, \dots, s_k (“stairs”), and introduce a specific class of models:

► **Definition 5.1.** Let $k, i \geq 0$ and let (\mathcal{K}, T) be a Kripke structure with team, $\mathcal{K} = (W, R, V)$. A team T is k -canonical with offset i if for every $\tau \in \Delta_k$ there exists $w \in T$ with $\llbracket R^i w \rrbracket_k = \{\tau\}$.

(\mathcal{K}, T) is called k -staircase if for all $i \in \{0, \dots, k\}$ we have that T_{s_i} is i -canonical with offset $k - i$.

A 3-staircase for $\Phi = \emptyset$ is depicted in Figure 3, which is easily adapted for $\Phi \neq \emptyset$ and arbitrary k . In particular, it is a *directed forest*, which means that its underlying undirected graph is acyclic and all its worlds are either *roots* (i.e., without predecessor) or have exactly one predecessor. Moreover, it has bounded *height*, where the height of a directed forest is the greatest number h such that every path traverses at most h edges.

► **Proposition 5.2.** *For each $k \geq 0$, there is a finite k -staircase (\mathcal{K}, T) such that $\mathfrak{s}_0, \dots, \mathfrak{s}_k$ are disjoint scopes in \mathcal{K} , and \mathcal{K} is a directed forest with height at most k and its set of roots being exactly T .*

Observe that a model being a k -staircase is a stronger condition than k -canonicity.

► **Corollary 5.3.** *Every satisfiable MTL_k -formula has a finite model (\mathcal{K}, T) such that \mathcal{K} is a directed forest with height at most k and its set of roots being exactly T .*

Enforcing canonicity

In the rest of the section, we illustrate how a k -staircase can be enforced in MTL inductively.

For $\Phi = \emptyset$, the inductive step – obtaining $(k + 1)$ -canonicity from k -canonicity – is captured by the formula $\forall_{\alpha}^{\subseteq} \exists_{\beta}^1 \Box \chi_k^*(\alpha, \beta)$. It states that for every *subteam* $T' \subseteq T_{\alpha}$ there exists a *point* $w \in T_{\beta}$ such that $\llbracket RT' \rrbracket_k = \llbracket Rw \rrbracket_k$. Intuitively, every possible set of types is captured as the image of some point in T_{β} . As a consequence, if T_{α} is k -canonical with offset 1, then T_{β} will be $(k + 1)$ -canonical.

Note that the straightforward formula $\Box^k \max(\Phi)$ expresses 0-canonicity of $R^k T$, but *not* 0-canonicity of T with offset k (consider, e.g., a singleton T). Instead, we use the formula

$$\text{max-off}_i(\beta) := \beta \leftrightarrow \left(\Diamond^i \top \wedge (\Box^i \max(\Phi)) \wedge \forall_{\beta}^1 \Box^i \bigwedge_{p \in \Phi} \text{=(} p \text{)} \right).$$

It states that $R^i T_{\beta}$ is 0-canonical, but that $R^i w$ admits only one propositional assignment for each $w \in T_{\beta}$. In this light, k -canonicity with offset i is altogether defined as follows:

$$\begin{aligned} \rho_0^i(\beta) &:= \exists_{\beta}^{\subseteq} \text{max-off}_i(\beta) \\ \rho_{k+1}^i(\alpha, \beta) &:= \forall_{\alpha}^{\subseteq} \exists_{\beta}^{\subseteq} (\rho_0^i(\beta) \wedge \Box^i \forall_{\beta}^1 \Box \chi_k^*(\alpha, \beta)) \\ \text{canon}_k &:= \rho_0^k(\mathfrak{s}_0) \wedge \bigwedge_{m=1}^k \rho_m^{k-m}(\mathfrak{s}_{m-1}, \mathfrak{s}_m) \end{aligned}$$

► **Theorem 5.4** (\star). *Let $k \geq 0$. The formula canon_k is an MTL_k -formula and constructible in space $\mathcal{O}(\log(|\Phi| + k))$.*

Moreover, if \mathcal{K} is a Kripke structure with disjoint scopes $\mathfrak{s}_0, \dots, \mathfrak{s}_k$, then $(\mathcal{K}, T) \models \text{canon}_k$ if and only if (\mathcal{K}, T) is a k -staircase.

Proof sketch. By induction on k . We sketch the induction step.

Suppose T_{α} is k -canonical with offset $i + 1$. For each $S \subseteq T_{\alpha}$, the formula $\rho_{k+1}^i(\alpha, \beta)$ quantifies a subteam $U \subseteq T_{\beta}$ that is 0-canonical with offset i . Additionally, it also forces all points in $R^i U$ (and hence at least one point of every 0-type) to mimic the k -types of $R^{i+1} S$ in all points of their image. Together, this results in $(k + 1)$ -canonicity with offset i . ◀

It remains to demonstrate that the restriction of the \mathfrak{s}_i being scopes *a priori* can be omitted, since we can, in a sense, define it in MTL as well. For this, let $\Psi \subseteq \mathcal{PS}$ be disjoint

from Φ . Then the formula below ensures that Ψ is a set of disjoint scopes “up to height k ”, which is sufficient for our purposes.

$$\text{scopes}_k(\Psi) := \bigwedge_{\substack{x,y \in \Psi \\ x \neq y}} \neg(x \wedge y) \wedge \bigwedge_{i=1}^k \left((x \wedge \Box^i x) \vee (\neg x \wedge \Box^i \neg x) \right).$$

► **Lemma 5.5.** *If $\varphi \in \text{MTL}_k$, then φ is satisfiable if and only if $\varphi \wedge \Box^{k+1} \perp$ is satisfiable.*

Proof. As the direction from right to left is trivial, assume that φ is satisfiable. By Corollary 5.3, it then has a model (\mathcal{K}, T) that is a directed forest of height at most k . But then $(\mathcal{K}, T) \models \Box^{k+1} \perp$, since $R^{k+1}T = \emptyset$ and (\mathcal{K}, \emptyset) satisfies all ML-formulas, including \perp . ◀

► **Theorem 5.6.** *$\text{canon}_k \wedge \text{scopes}_k(\{\mathfrak{s}_0, \dots, \mathfrak{s}_k\}) \wedge \Box^{k+1} \perp$ is satisfiable, but has only k -staircases as models.*

Proof. By combining Proposition 5.2, Theorem 5.4 and Lemma 5.5, the formula is satisfiable. Since in every model (\mathcal{K}, T) the propositions $\mathfrak{s}_0, \dots, \mathfrak{s}_k$ must be disjoint scopes due to $\Box^{k+1} \perp$ and scopes_k , we can apply Theorem 5.4. ◀

Let us stress that the formula canon_k is again only an approximation of k -canonicity, since the scopes $\mathfrak{s}_0, \dots, \mathfrak{s}_{k-1}$ are necessary for the construction as well. However, both χ_k and canon_k being efficiently constructible is crucial for our main result in the next section.

6 Complexity lower bounds

In this section, we provide the matching lower bounds for Theorem 3.8 and Corollary 3.9:

► **Theorem 6.1.** *$\text{SAT}(\text{MTL})$ and $\text{VAL}(\text{MTL})$ are complete for $\text{TOWER}(\text{poly})$. For all $k \geq 0$, $\text{SAT}(\text{MTL}_k)$ and $\text{VAL}(\text{MTL}_k)$ are complete for $\text{ATIME-ALT}(\exp_{k+1}, \text{poly})$.*

The above complexity classes are complement-closed, and MTL and MTL_k are closed under negation. For this reason, it suffices to consider $\text{SAT}(\text{MTL})$ and $\text{SAT}(\text{MTL}_k)$. Moreover, the case $k = 0$ is equivalent to $\text{SAT}(\text{PTL})$ being $\text{ATIME-ALT}(\exp, \text{poly})$ -hard, which was proven by Hannula et al. [14]. Their reduction works in logarithmic space.

Consequently, the result boils down to the following lemma:

► **Lemma 6.2.** *If $L \in \text{TOWER}(\text{poly})$, then $L \leq_m^{\log} \text{SAT}(\text{MTL})$.
If $k \geq 1$ and $L \in \text{ATIME-ALT}(\exp_{k+1}, \text{poly})$, then $L \leq_m^{\log} \text{SAT}(\text{MTL}_k)$.*

We devise for each L a reduction $x \mapsto \varphi_x$ such that φ_x is a formula that is satisfiable if and only if $x \in L$. By assumption, there exists a single-tape alternating Turing machine M that decides L (for $L \in \text{TOWER}(\text{poly})$, w.l.o.g. M is alternating as well). Then $M = (Q, \Gamma, \delta)$, where Q is the disjoint union of Q_{\exists} (existential states), Q_{\forall} (universal states), Q_{acc} (accepting states) and Q_{rej} (rejecting states). Also, Q contains some initial state q_0 . Γ is the finite tape alphabet, \flat the blank symbol, and δ the transition relation.

We design φ_x in a fashion that forces its models (\mathcal{K}, T) to encode an accepting computation of M on x . Let us call any legal sequence of configurations of M (not necessarily starting with the initial configuration) a *run*. Then, similarly as in Cook’s famous theorem [5], we encode runs as square “grids” with a vertical “time” coordinate and a horizontal “space” coordinate in the model, i.e., each row of the grid represents a configuration of M .

W.l.o.g. M has runtime at most N and tape cells $\{1, \dots, N\}$. A run of M is then a function $C: \{1, \dots, N\}^2 \rightarrow \Gamma \cup (Q \times \Gamma)$. In M 's initial configuration, for instance, we have $C(1, 1) = (q_0, x_1)$, $C(i, 1) = x_i$ for $2 \leq i \leq n$, and $C(i, 1) = \flat$ for $n < i \leq N$.

Due to the semantics of MTL, such a run must be encoded in (\mathcal{K}, T) very carefully. We let T contain N^2 worlds $w_{i,j}$ in which the respective value of $C(i, j)$ is encoded in a propositional assignment. However, we cannot simply pursue the standard approach of assembling a large $N \times N$ -grid in the edge relation R in order to compare successive configurations; by Corollary 5.3, we cannot force the model to contain R -paths longer than $|\varphi_x|$.

Instead, to define grid neighborhood, we let $w_{i,j}$ encode i and j in its *type*. More precisely, we impose a linear order \prec_k on Δ_k that is defined by an MTL_k -formula ζ_k . Then, instead of using \Box and \Diamond , we examine the grid by letting ζ_k judge whether a given pair of worlds is deemed (horizontally or vertically) adjacent. Analogously to χ_k^* , we also define an order \prec_k^* on teams via a formula ζ_k^* . Since order is a binary relation, the formulas are once more parameterized by two scopes:

$$\begin{aligned} \zeta_0(\alpha, \beta) &:= \bigvee_{p \in \Phi} \left[(\alpha \hookrightarrow \neg p) \wedge (\beta \hookrightarrow p) \wedge \bigwedge_{\substack{q \in \Phi \\ q < p}} (\alpha \vee \beta) \hookrightarrow (q) \right] \\ \zeta_{k+1}(\alpha, \beta) &:= \zeta_0(\alpha, \beta) \wp (\chi_0(\alpha, \beta) \wedge \Box \zeta_k^*(\alpha, \beta)) \\ \zeta_k^*(\alpha, \beta) &:= \exists_{\mathfrak{s}_k}^1 (\exists_{\beta}^1 \chi_k(\mathfrak{s}_k, \beta)) \wedge (\sim \exists_{\alpha}^1 \chi_k(\mathfrak{s}_k, \alpha)) \\ &\quad \wedge \left((\chi_k^*(\alpha, \beta) \wedge (\alpha \vee \beta)) \vee (\forall_{\alpha \vee \beta}^1 \sim \zeta_k(\mathfrak{s}_k, \alpha \vee \beta)) \right) \end{aligned}$$

We refer the reader to the full paper [30] for the proof that there exist orders \prec_k and \prec_k^* on Δ_k and $\mathfrak{P}(\Delta_k)$ that are defined by ζ_k and ζ_k^* in the following sense:

► **Theorem 6.3** (*). *Let $k \geq 0$, and (\mathcal{K}, T) be a k -staircase with disjoint scopes $\alpha, \beta, \mathfrak{s}_0, \dots, \mathfrak{s}_k$. If $w \in T_\alpha$ and $v \in T_\beta$, then*

$$\begin{aligned} T_{w,v}^{\alpha,\beta} &\models \zeta_k(\alpha, \beta) \text{ if and only if } \llbracket w \rrbracket_k \prec_k \llbracket v \rrbracket_k, \\ T &\models \zeta_k^*(\alpha, \beta) \text{ if and only if } \llbracket T_\alpha \rrbracket_k \prec_k^* \llbracket T_\beta \rrbracket_k. \end{aligned}$$

Furthermore, both $\zeta_k(\alpha, \beta)$ and $\zeta_k^*(\alpha, \beta)$ are MTL_k -formulas that are constructible in space $\mathcal{O}(\log(k + |\Phi| + |\alpha| + |\beta|))$.

Encoding runs in a team

Next, we discuss in more detail how runs $C: \{1, \dots, N\}^2 \rightarrow \Gamma \cup (Q \times \Gamma)$ are encoded in a team T . Given a world $w \in T$, we partition the image Rw with two special propositions $\mathfrak{t} \notin \Phi$ (“timestep”) and $\mathfrak{p} \notin \Phi$ (“position”). Then we assign to w the pair $\ell(w) := (i, j)$ such that $\llbracket (Rw)_{\mathfrak{t}} \rrbracket_{k-1}$ is the i -th element, and $\llbracket (Rw)_{\mathfrak{p}} \rrbracket_{k-1}$ is the j -th element in the order \prec_{k-1}^* . We call the pair $\ell(w)$ the *location* of w (in the grid).

Accordingly, we fix $N := |\mathfrak{P}(\Delta_{k-1}^\Phi)|$. For the case of fixed k , M has runtime bounded by $\exp_{k+1}(g(n))$ for a polynomial g . Then taking $\Phi := \{p_1, \dots, p_{g(n)}\}$ yields a sufficiently large coordinate space, as

$$\exp_{k+1}(g(n)) = \exp_{k+1}(|\Phi|) = 2^{\exp_{k-1}(2^{|\Phi|})} \leq 2^{\exp_{k-1}^*(2^{|\Phi|})} = 2^{|\Delta_{k-1}^\Phi|} = |\mathfrak{P}(\Delta_{k-1}^\Phi)|$$

by Proposition 3.5. Likewise, if in the second case M has runtime bounded by $\exp_{g(n)}(1)$, we let $\Phi := \emptyset$ and compute $k := g(|x|) + 1$, but otherwise proceed identically.

Next, let Ξ be a constant set of propositions disjoint from Φ that encodes the range of C via some bijection $c: \Xi \rightarrow \Gamma \cup (Q \times \Gamma)$. If a world w satisfies exactly one proposition p of those in Ξ , then we define $c(w) := c(p)$. Intuitively, $c(w)$ is the *content* of the grid cell represented by w .

Using ℓ and c , the function C can be encoded into a team T as follows. First, a team T is called *grid* if every point in T satisfies exactly one proposition in Ξ , and if every location $(i, j) \in \{1, \dots, N\}^2$ occurs as $\ell(w)$ for some point $w \in T$. Moreover, a grid T is called *pre-tableau* if for every location (i, j) and every element $p \in \Xi$ there is some world $w \in T$ such that $\ell(w) = (i, j)$ and $w \models p$. Finally, a grid T is a *tableau* if any two elements $w, w' \in T$ with $\ell(w) = \ell(w')$ also agree on Ξ , i.e., $c(w) = c(w')$.

Let us motivate the above definitions. Clearly, the definition of a grid T means that T captures the whole domain of C , and that c is well-defined on the level of *points*. If T is additionally a tableau, then c is also well-defined on the level of *locations*. In other words, every tableau T induces a function $C_T: \{1, \dots, N\}^2 \rightarrow \Gamma \cup (Q \times \Gamma)$ via $C_T(i, j) := c(w)$, where $w \in T$ is arbitrary such that $\ell(w) = (i, j)$. Finally, a pre-tableau is, roughly speaking, the “union” of all possible C . In particular, given any pre-tableau, the definition ensures that arbitrary tableaus can be obtained from it by the means of subteam quantification \exists^{\subseteq} (cf. p. 9).

A tableau T is *legal* if C_T is a run of M , i.e., if every row is a configuration of M , and if every pair of two successive rows represents a valid δ -transition.

The idea of the reduction is now to capture the alternating computation of M by nesting polynomially many quantifications (via \exists^{\subseteq} and \forall^{\subseteq}) of legal tableaus, of which each one is the continuation of the computation of the previous one. For this purpose, we devise formulas such as $\psi_{\text{pre-tableau}}(\alpha)$ and $\psi_{\text{legal}}(\alpha)$ that express that T_α is a pre-tableau, or a legal tableau, respectively. These formulas rely on canon_k to achieve a sufficiently large team, and on ζ_k resp. ζ_k^* for accessing adjacent grid cells in order to verify the transitions between configurations.

Due to space constraints, we cannot present their implementation here. Instead, we refer the reader to the appendix or the full version of the paper [30] for details.

7 Concluding remarks

In Theorem 6.1, we settled the open question of the complexity of MTL and established TOWER(poly)-completeness for its satisfiability and validity problem. Likewise, the fragments MTL_k are proved complete for $\text{ATIME-ALT}(\text{exp}_{k+1}, \text{poly})$, the levels of the elementary hierarchy with polynomially many alternations.

As our main tool, we introduced a suitable notion of canonical models for modal logics with team semantics. We showed that such models exist for MTL and MTL_k , and that some satisfiable MTL_k -formulas of polynomial size have *only* k -canonical models.

Our lower bounds carry over to two-variable first-order team logic $\text{FO}^2(\sim)$ and its fragment $\text{FO}_k^2(\sim)$ of bounded quantifier rank k as well [29]. While the former is TOWER(poly)-complete, the latter is $\text{ATIME-ALT}(\text{exp}_{k+1}, \text{poly})$ -hard. However, no matching upper bound for the satisfiability problem of $\text{FO}_k^2(\sim)$ exists.

In future research, it could be useful to further generalize the concept of canonical models for other logics with team semantics. Do logics such as $\text{FO}_k^2(\sim)$ permit a canonical model in the spirit of k -canonical models for MTL_k , and does this yield a tight upper bound on the complexity of their satisfiability problem? How do MTL_k and $\text{FO}_k^2(\sim)$ differ in terms of succinctness?

References

- 1 Samson Abramsky, Juha Kontinen, Jouko Väänänen, and Heribert Vollmer, editors. *Dependence Logic, Theory and Applications*. Springer, 2016. doi:10.1007/978-3-319-31803-5.
- 2 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal logic*. Cambridge University Press, New York, NY, USA, 2001.
- 3 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.
- 4 Kevin J. Compton and C. Ward Henson. A Uniform Method for Proving Lower Bounds on the Computational Complexity of Logical Theories. *Ann. Pure Appl. Logic*, 48(1):1–79, 1990. doi:10.1016/0168-0072(90)90080-L.
- 5 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971. doi:10.1145/800157.805047.
- 6 Arnaud Durand, Juha Kontinen, and Heribert Vollmer. *Expressivity and Complexity of Dependence Logic*, pages 5–32. Springer International Publishing, 2016. doi:10.1007/978-3-319-31803-5_2.
- 7 Pietro Galliani. Inclusion and exclusion dependencies in team semantics - on some logics of imperfect information. *Ann. Pure Appl. Logic*, 163(1):68–84, 2012. doi:10.1016/j.apal.2011.08.005.
- 8 Pietro Galliani. Upwards closed dependencies in team semantics. *Information and Computation*, 245:124–135, 2015. doi:10.1016/j.ic.2015.06.008.
- 9 Erich Grädel and Jouko Väänänen. Dependence and independence. *Studia Logica*, 101(2):399–410, 2013. doi:10.1007/s11225-013-9479-2.
- 10 Miika Hannula. Validity and entailment in modal and propositional dependence logics. *CoRR*, abs/1608.04301, 2016. URL: <http://arxiv.org/abs/1608.04301>.
- 11 Miika Hannula. Validity and Entailment in Modal and Propositional Dependence Logics. In *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CSL.2017.28.
- 12 Miika Hannula, Juha Kontinen, Martin Lück, and Jonni Virtema. On Quantified Propositional Logics and the Exponential Time Hierarchy. In *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016*, pages 198–212, 2016. doi:10.4204/EPTCS.226.14.
- 13 Miika Hannula, Juha Kontinen, Jonni Virtema, and Heribert Vollmer. Complexity of propositional independence and inclusion logic. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015*, pages 269–280, 2015. doi:10.1007/978-3-662-48057-1_21.
- 14 Miika Hannula, Juha Kontinen, Jonni Virtema, and Heribert Vollmer. Complexity of Propositional Logics in Team Semantic. *ACM Transactions on Computational Logic*, 19(1):1–14, jan 2018. doi:10.1145/3157054.
- 15 Lauri Hella, Antti Kuusisto, Arne Meier, and Jonni Virtema. Model checking and validity in propositional and modal inclusion logics. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017*, pages 32:1–32:14, 2017. doi:10.4230/LIPIcs.MFCS.2017.32.
- 16 Lauri Hella, Antti Kuusisto, Arne Meier, and Heribert Vollmer. Modal Inclusion Logic: Being Lax is Simpler than Being Strict. In *Mathematical Foundations of Computer Science 2015*, volume 9234, pages 281–292. Springer Berlin Heidelberg, 2015. URL: http://link.springer.com/10.1007/978-3-662-48057-1_22.
- 17 Lauri Hella, Kerkko Luosto, Katsuhiko Sano, and Jonni Virtema. The expressive power of modal dependence logic. In *Advances in Modal Logic 10, invited and contributed papers*

- from the tenth conference on “Advances in Modal Logic”. Groningen, The Netherlands, August 5-8, 2014, pages 294–312, 2014. URL: <http://www.aiml.net/volumes/volume10/Hella-Luosto-Sano-Virtema.pdf>.
- 18 Lauri Hella and Johanna Stumpf. The expressive power of modal logic with inclusion atoms. *Electronic Proceedings in Theoretical Computer Science*, 193:129–143, 2015. doi: 10.4204/EPTCS.193.10.
 - 19 Jaakko Hintikka and Gabriel Sandu. Informational Independence as a Semantical Phenomenon. In Jens Erik Fenstad, Ivan T. Frolov, and Risto Hilpinen, editors, *Logic, Methodology and Philosophy of Science VIII*, volume 126 of *Studies in Logic and the Foundations of Mathematics*, pages 571–589. Elsevier, 1989. doi:10.1016/S0049-237X(08)70066-1.
 - 20 Wilfrid Hodges. Compositional semantics for a language of imperfect information. *Logic Journal of IGPL*, 5(4):539–563, 1997. doi:10.1093/jigpal/5.4.539.
 - 21 M. Jonáš and J. Strejček. On the complexity of the quantified bit-vector arithmetic with binary encoding. *Information Processing Letters*, 2018. doi:10.1016/j.ipl.2018.02.018.
 - 22 Juha Kontinen, Julian-Steffen Müller, Henning Schnoor, and Heribert Vollmer. A Van Benthem theorem for modal team semantics. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, pages 277–291, 2015. doi:10.4230/LIPIcs.CSL.2015.277.
 - 23 Juha Kontinen, Julian-Steffen Müller, Henning Schnoor, and Heribert Vollmer. Modal independence logic. *Journal of Logic and Computation*, 27(5):1333–1352, 2017. doi:10.1093/logcom/exw019.
 - 24 Juha Kontinen and Ville Nurmi. Team logic and second-order logic. *Fundam. Inform.*, 106(2-4):259–272, 2011. doi:10.3233/FI-2011-386.
 - 25 Andreas Krebs, Arne Meier, and Jonni Virtema. A team based variant of CTL. In *22nd International Symposium on Temporal Representation and Reasoning, TIME 2015*, pages 140–149, 2015. doi:10.1109/TIME.2015.11.
 - 26 Peter Lohmann and Heribert Vollmer. Complexity results for modal dependence logic. *Studia Logica*, 101(2):343–366, 04 2013. doi:10.1007/s11225-013-9483-6.
 - 27 Martin Lück. Axiomatizations for propositional and modal team logic. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016*, pages 33:1–33:18, 2016. doi: 10.4230/LIPIcs.CSL.2016.33.
 - 28 Martin Lück. The power of the filtration technique for modal logics with team semantics. In *26th EACSL Annual Conference on Computer Science Logic, CSL 2017*, pages 31:1–31:20, 2017. doi:10.4230/LIPIcs.CSL.2017.31.
 - 29 Martin Lück. On the complexity of team logic and its two-variable fragment. MFCS 2018. To appear.
 - 30 Martin Lück. Canonical models and the complexity of modal team logic. *CoRR*, abs/1709.05253, 2017. URL: <https://arxiv.org/abs/1709.05253>.
 - 31 Julian-Steffen Müller. *Satisfiability and model checking in team based logics*. PhD thesis, University of Hanover, 2014. URL: <http://d-nb.info/1054741921>.
 - 32 Sylvain Schmitz. Complexity hierarchies beyond elementary. *TOCT*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.
 - 33 Merlijn Sevenster. Model-theoretic and computational properties of modal dependence logic. *J. Log. Comput.*, 19(6):1157–1173, 2009. doi:10.1093/logcom/exn102.
 - 34 Larry J. Stockmeyer and Albert R. Meyer. Word Problems Requiring Exponential Time: Preliminary Report. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pages 1–9, 1973. doi:10.1145/800125.804029.
 - 35 Jouko Väänänen. Modal dependence logic. *New perspectives on games and interaction*, 4:237–254, 2008.

- 36 Jonni Virtema. Complexity of validity for propositional dependence logics. *Inf. Comput.*, 253:224–236, 2017. doi:10.1016/j.ic.2016.07.008.
- 37 Marco Voigt. A fine-grained hierarchy of hard problems in the separated fragment. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005094.
- 38 Jouko Väänänen. *Dependence logic: A New Approach to Independence Friendly Logic*. Number 70 in London Mathematical Society student texts. Cambridge University Press, Cambridge ; New York, 2007.
- 39 Fan Yang and Jouko Väänänen. Propositional logics of dependence. *Ann. Pure Appl. Logic*, 167(7):557–589, 2016. doi:10.1016/j.apal.2016.03.003.
- 40 Fan Yang and Jouko Väänänen. Propositional team logics. *Ann. Pure Appl. Logic*, 168(7):1406–1441, 2017. doi:10.1016/j.apal.2017.01.007.

A Details of the reduction (Lemma 6.2)

In the appendix, we present our lower bound in detail:

- **Lemma 6.2.** *If $L \in \text{TOWER}(\text{poly})$, then $L \leq_m^{\log} \text{SAT}(\text{MTL})$.
If $k > 0$ and $L \in \text{ATIME-ALT}(\exp_{k+1}, \text{poly})$, then $L \leq_m^{\log} \text{SAT}(\text{MTL}_k)$.*

We describe the reduction $x \mapsto \varphi_x$. In what follows, let $n := |x|$. The correctness proof for the reduction will be built on several claims. These claims are not hard to derive, and for detailed proofs of all steps we refer the reader to the full version of the paper [30].

As discussed in Section 6, we choose to represent a location (i, j) in a point w as a pair (Δ', Δ'') by stipulating that $\Delta' = \llbracket (Rw)_t \rrbracket_{k-1}$ and $\Delta'' = \llbracket (Rw)_p \rrbracket_{k-1}$, where \mathbf{t} (“time”) and \mathbf{p} (“position”) are special propositions in $\mathcal{PS} \setminus \Phi$. To access the two components of an encoded location independently, we introduce the operator $|\mathbf{q}^\alpha \psi := (\alpha \wedge \neg \mathbf{q}) \vee ((\alpha \hookrightarrow \mathbf{q}) \wedge \psi)$, where $\mathbf{q} \in \{\mathbf{t}, \mathbf{p}\}$ and $\alpha \in \text{ML}$. It is easy to check that $T \models |\mathbf{q}^\alpha \psi$ iff $T_{T_{\mathbf{q}}}^\alpha \models \psi$.

In order to *compare* the locations of grid cells, for $\mathbf{q} \in \{\mathbf{t}, \mathbf{p}\}$ we define the formulas $\psi_{\prec}^{\mathbf{q}}(\alpha, \beta)$, which tests whether the location in T_α is less than the one in T_β w. r. t. its \mathbf{q} -component (assuming singleton teams T_α and T_β), and $\psi_{\equiv}^{\mathbf{q}}(\alpha, \beta)$ which checks for equality of the respective component:

$$\psi_{\prec}^{\mathbf{q}}(\alpha, \beta) := \Box |\mathbf{q}^\alpha \zeta_{k-1}^*(\alpha, \beta) \quad \psi_{\equiv}^{\mathbf{q}}(\alpha, \beta) := \Box |\mathbf{q}^\alpha \chi_{k-1}^*(\alpha, \beta)$$

For this purpose, $\psi_{\prec}^{\mathbf{q}}$ is built upon the formula ζ_{k-1}^* from Theorem 6.3, while $\psi_{\equiv}^{\mathbf{q}}$ checks for equality with the help of χ_{k-1}^* from Theorem 4.4.

- **Claim (a).** *Let \mathcal{K} be a structure with a team T and disjoint scopes α and β .
Suppose $w \in T_\alpha$ and $v \in T_\beta$, where $\ell(w) = (i_w, j_w)$ and $\ell(v) = (i_v, j_v)$. Then:*

$$T_{w,v}^{\alpha,\beta} \models \psi_{\equiv}^{\mathbf{t}}(\alpha, \beta) \Leftrightarrow i_w = i_v \quad T_{w,v}^{\alpha,\beta} \models \psi_{\equiv}^{\mathbf{p}}(\alpha, \beta) \Leftrightarrow j_w = j_v.$$

Moreover, if $\alpha, \beta, \mathfrak{s}_0, \dots, \mathfrak{s}_k$ are disjoint scopes in \mathcal{K} and (\mathcal{K}, T) is a k -staircase, then:

$$T_{w,v}^{\alpha,\beta} \models \psi_{\prec}^{\mathbf{t}}(\alpha, \beta) \Leftrightarrow i_w < i_v \quad T_{w,v}^{\alpha,\beta} \models \psi_{\prec}^{\mathbf{p}}(\alpha, \beta) \Leftrightarrow j_w < j_v.$$

Next, we construct formulas that check whether a given team is a grid, pre-tableau, or a tableau, respectively. To check that every location $(i, j) \in \{1, \dots, N\}^2$ of the grid occurs as $\ell(w)$ of some $w \in T$, we quantify over all pairs $(\Delta', \Delta'') \in \mathfrak{P}(\Delta_{k-1})^2$. To cover all these sets of types we can quantify, for instance, over the images of all points of $T_{\mathfrak{s}_k}$. As we cannot

pick *two* subteams from the same scope at once, we enforce a k -canonical copy $T_{\mathfrak{s}'_k}$ of $T_{\mathfrak{s}_k}$ in the spirit of Theorem 5.4:

$$\text{canon}' := \rho_0^k(\mathfrak{s}_0) \wedge \bigwedge_{m=1}^k \rho_m^{k-m}(\mathfrak{s}_{m-1}, \mathfrak{s}_m) \wedge \rho_k^0(\mathfrak{s}_{k-1}, \mathfrak{s}'_k)$$

► **Claim (b).** *If $\mathfrak{s}_0, \dots, \mathfrak{s}_k, \mathfrak{s}'_k$ are disjoint scopes in \mathcal{K} , then $(\mathcal{K}, T) \models \text{canon}'$ if and only if (\mathcal{K}, T) is a k -staircase and $T_{\mathfrak{s}'_k}$ is k -canonical.*

Moreover, $\text{canon}' \wedge \text{scopes}_k(\{\mathfrak{s}_0, \dots, \mathfrak{s}_k, \mathfrak{s}'_k\}) \wedge \square^{k+1} \perp$ is satisfiable, but is only satisfied by k -staircases (\mathcal{K}, T) in which both $T_{\mathfrak{s}_k}$ and $T_{\mathfrak{s}'_k}$ are k -canonical. Furthermore, both formulas are constructible in space $\mathcal{O}(\log(|\Phi| + k))$.

The next formulas define grids resp. pre-tableaus.

$$\begin{aligned} \psi_{\text{pair}}(\alpha) &:= \square \left[\left(\left| \chi_{k-1}^* \right|_{\mathfrak{s}_k}(\mathfrak{s}_k, \alpha) \right) \wedge \left(\left| \chi_{k-1}^* \right|_{\mathfrak{s}'_k}(\mathfrak{s}'_k, \alpha) \right) \right] \\ \psi_{\text{grid}}(\alpha) &:= \left(\alpha \hookrightarrow \bigvee_{e \in \Xi} e \wedge \bigwedge_{\substack{e' \in \Xi \\ e' \neq e}} \neg e' \right) \wedge \forall_{\mathfrak{s}_k}^1 \forall_{\mathfrak{s}'_k}^1 \exists_{\alpha}^1 \psi_{\text{pair}}(\alpha) \\ \psi_{\text{pre-tableau}}(\alpha) &:= \psi_{\text{grid}}(\alpha) \wedge \forall_{\mathfrak{s}_k}^1 \forall_{\mathfrak{s}'_k}^1 \bigwedge_{e \in \Xi} \exists_{\alpha}^1 (\psi_{\text{pair}}(\alpha) \wedge (\alpha \hookrightarrow e)) \end{aligned}$$

In all subsequent claims, we always assume that T is a team in a Kripke structure \mathcal{K} such that (\mathcal{K}, T) satisfies $\text{canon}' \wedge \square^{k+1} \perp$. Moreover, all stated scopes are always assumed pairwise disjoint in \mathcal{K} (as we can enforce this later in the reduction with $\text{scopes}_k(\dots)$).

► **Claim (c).** *$T \models \psi_{\text{grid}}(\alpha)$ if and only if T_{α} is a grid and $T \models \psi_{\text{pre-tableau}}(\alpha)$ if and only if T_{α} is a pre-tableau.*

The other special case of a grid, that is, a *tableau*, requires a more elaborate approach to define in MTL. The difference to a grid or pre-tableau is that we have to quantify over all *pairs* (w, w') of points in T , and check that they agree on Ξ if $\ell(w) = \ell(w')$. However, as discussed before, while \forall^1 can quantify over all points in a team, it cannot quantify over pairs. As a workaround, we consider not only a tableau T_{α} , but also a *second* tableau that acts as a copy of T_{α} . Formally, for grids T_{α}, T_{β} , let $T_{\alpha} \approx T_{\beta}$ denote that for all pairs $(w, w') \in T_{\alpha} \times T_{\beta}$ it holds that $\ell(w) = \ell(w')$ implies $c(w) = c(w')$.

As \approx is symmetric and transitive, $T_{\alpha} \approx T_{\beta}$ in fact implies both $T_{\alpha} \approx T_{\alpha}$ and $T_{\beta} \approx T_{\beta}$, and hence that both T_{α} and T_{β} are tableaus such that $C_{T_{\alpha}} = C_{T_{\beta}}$, where $C_{T_{\alpha}}, C_{T_{\beta}}: \{1, \dots, N\}^2 \rightarrow \Gamma \cup (Q \times \Gamma)$ are the induced runs as discussed on p. 15.

$$\begin{aligned} \psi_{\text{tableau}}(\alpha) &:= \psi_{\text{grid}}(\alpha) \wedge \exists_{\gamma_0}^{\subseteq} \psi_{\text{grid}}(\gamma_0) \wedge \psi_{\approx}(\alpha, \gamma_0) \\ \psi_{\approx}(\alpha, \beta) &:= \forall_{\alpha}^1 \forall_{\beta}^1 \left(\left(\psi_{\Xi}^{\mathfrak{t}}(\alpha, \beta) \wedge \psi_{\Xi}^{\mathfrak{p}}(\alpha, \beta) \right) \rightarrow \bigvee_{e \in \Xi} ((\alpha \vee \beta) \hookrightarrow e) \right) \end{aligned}$$

In the following claim (and in the subsequent ones), we use the scopes $\gamma_0, \gamma_1, \gamma_2, \dots$ as “auxiliary pre-tableaus”. Later, we will also use them as domains to quantify extra locations or rows from. (The index of γ_i is incremented whenever necessary to avoid quantifying from the same scope twice.) For this reason, from now on we always assume, for sufficiently large i , that T_{γ_i} is a pre-tableau. This can be later enforced in the reduction with $\psi_{\text{pre-tableau}}(\gamma_i)$.

► **Claim (d).** *$T \models \psi_{\text{tableau}}(\alpha)$ if and only if T_{α} is a tableau.*

For grids T_{α}, T_{β} , it holds $T \models \psi_{\approx}(\alpha, \beta)$ if and only if $T_{\alpha} \approx T_{\beta}$.

To ascertain that a tableau contains a run of M , we have to check whether each row indeed is a configuration of M and whether consecutive configurations adhere to the transition relation δ of M . For the latter, in the spirit of Cook's theorem [5], it suffices to consider all *legal windows* in the grid, i.e., cells that are adjacent as follows, where $e_1, \dots, e_6 \in \Gamma \cup (Q \times \Gamma)$:

e_1	e_2	e_3
e_4	e_5	e_6

If, say, $(q, a, q', a', R) \in Q \times \Gamma \times Q \times \Gamma \times \{L, R, N\}$ is a transition – M switches to state q' from q , replacing a on the tape by a' , and moves to the right – then the windows obtained by setting $e_1 = e_4 = b$, $e_2 = (q, a)$, $e_5 = a'$, $e_3 = b'$, $e_6 = (q', b')$ are legal for all $b, b' \in \Gamma$. Using this scheme, δ is completely represented by some constant finite set $\text{win} \subseteq \Xi^6$ of tuples (e_1, \dots, e_6) that represent the allowed windows in a run of M .

Let us next explain how adjacency of cells is expressed. Suppose that two points $w \in T_\alpha$ and $v \in T_\beta$ are given. That v is the immediate (t- or p-)successor of w then means that no element of the order exists between them. Simultaneously, w and v have to agree on the other component of their location, which is expressed by the first conjunct below. Formally, if $q \in \{\mathfrak{t}, \mathfrak{p}\}$ and $\bar{q} \in \{\mathfrak{t}, \mathfrak{p}\} \setminus \{q\}$, then we define:

$$\psi_{\text{succ}}^q(\alpha, \beta) := \psi_{\bar{q}}^{\bar{q}}(\alpha, \beta) \wedge \psi_{\bar{q}}^q(\alpha, \beta) \wedge \sim \exists_{\gamma_0}^1 (\psi_{\bar{q}}^q(\alpha, \gamma_0) \wedge \psi_{\bar{q}}^q(\gamma_0, \beta))$$

► **Claim (e).** *If $w \in T_\alpha$ and $v \in T_\beta$, then:*

$$T_{w,v}^{\alpha,\beta} \models \psi_{\text{succ}}^{\mathfrak{t}}(\alpha, \beta) \Leftrightarrow \exists i, j \in \{1, \dots, N\}: \ell(w) = (i, j) \text{ and } \ell(v) = (i+1, j)$$

$$T_{w,v}^{\alpha,\beta} \models \psi_{\text{succ}}^{\mathfrak{p}}(\alpha, \beta) \Leftrightarrow \exists i, j \in \{1, \dots, N\}: \ell(w) = (i, j) \text{ and } \ell(v) = (i, j+1)$$

In this vein, we proceed by quantifying windows in the tableau T_α by quantifying elements from *six* tableaux $T_{\gamma_1}, \dots, T_{\gamma_6}$ that are copies of T_α . For this purpose, we abbreviate

$$\exists_{\gamma_i}^{\approx \alpha} \varphi := \exists_{\gamma_i}^{\subseteq} \psi_{\text{grid}}(\gamma_i) \wedge \psi_{\approx}(\alpha, \gamma_i) \wedge \varphi.$$

Intuitively, under the premise that T_{γ_i} is a pre-tableau and T_α is a tableau, it “copies the tableau T_α into T_{γ_i} ” by shrinking T_{γ_i} accordingly. This is proven analogously to Claim (d). The next formula states that the picked points are adjacent as shown in the picture below:

$$\psi_{\text{window}}(\gamma_1, \dots, \gamma_6) := \bigwedge_{i \in \{1,2,3\}} \psi_{\text{succ}}^{\mathfrak{t}}(\gamma_i, \gamma_{i+3}) \wedge \psi_{\text{succ}}^{\mathfrak{p}}(\gamma_1, \gamma_2) \wedge \psi_{\text{succ}}^{\mathfrak{p}}(\gamma_2, \gamma_3)$$

Based on the above two, the formula defining legal tableaux follows.

$$\psi_{\text{legal}}(\alpha) := \psi_{\text{tableau}}(\alpha) \wedge \exists_{\gamma_1}^{\approx \alpha} \dots \exists_{\gamma_6}^{\approx \alpha} \vartheta_1 \wedge \vartheta_2 \wedge \vartheta_3$$

We check that no two distinct cells in any row both contain a state of M :

$$\begin{aligned} \vartheta_1 := & \forall_{\gamma_1}^1 \forall_{\gamma_2}^1 \left(\psi_{\bar{q}}^{\mathfrak{t}}(\gamma_1, \gamma_2) \wedge \psi_{\bar{q}}^{\mathfrak{p}}(\gamma_1, \gamma_2) \right) \rightarrow \\ & \bigwedge_{(q_1, a_1), (q_2, a_2) \in Q \times \Gamma} \sim \left((\gamma_1 \leftrightarrow c^{-1}(q_1, a_1)) \wedge (\gamma_2 \leftrightarrow c^{-1}(q_2, a_2)) \right) \end{aligned}$$

We also check that every row contains a state. Intuitively, $\forall_{\gamma_1}^1$ fixes some row and $\exists_{\gamma_2}^1 \psi_{\bar{q}}^{\mathfrak{t}}(\gamma_1, \gamma_2)$ searches that particular row for a state:

$$\vartheta_2 := \forall_{\gamma_1}^1 \exists_{\gamma_2}^1 \psi_{\bar{q}}^{\mathfrak{t}}(\gamma_1, \gamma_2) \wedge \bigvee_{(q,a) \in Q \times \Gamma} (\gamma_2 \leftrightarrow c^{-1}(q, a))$$

Finally, every window must be valid:

$$\vartheta_3 := \forall_{\gamma_1}^1 \cdots \forall_{\gamma_6}^1 \left(\psi_{\text{window}}(\gamma_1, \dots, \gamma_6) \rightarrow \bigvee_{(e_1, \dots, e_6) \in \text{win}} \bigwedge_{i=1}^6 (\gamma_i \leftrightarrow e_i) \right)$$

► **Claim (f).** $T \models \psi_{\text{legal}}(\alpha)$ iff T_α is a legal tableau, i.e., C_{T_α} exists and is a run of M .

To now encode the initial configuration on input $x = x_1 \cdots x_n$ in a tableau, we access the first n cells of the first row and assign the respective letter of x , as well as the initial state to the first cell. Moreover, we assign \flat to all other cells in that row. For each $\mathfrak{q} \in \{\mathfrak{t}, \mathfrak{p}\}$, we can check whether the location of a point in T_α is minimal in its \mathfrak{q} -component:

$$\psi_{\min}^{\mathfrak{q}}(\alpha) := \sim \exists_{\gamma_0}^1 \psi_{\prec}^{\mathfrak{q}}(\gamma_0, \alpha)$$

This enables us to fix the first row of the configuration:

$$\begin{aligned} \psi_{\text{input}}(\alpha) := & \exists_{\gamma_1}^{\approx \alpha} \cdots \exists_{\gamma_{n+1}}^{\approx \alpha} \exists_{\gamma_1}^1 \cdots \exists_{\gamma_n}^1 \psi_{\min}^{\mathfrak{t}}(\gamma_1) \wedge \psi_{\min}^{\mathfrak{p}}(\gamma_1) \wedge (\gamma_1 \leftrightarrow c^{-1}(q_0, x_1)) \\ & \bigwedge_{i=2}^n \psi_{\text{succ}}^{\mathfrak{p}}(\gamma_{i-1}, \gamma_i) \wedge (\gamma_i \leftrightarrow c^{-1}(x_i)) \\ & \wedge \forall_{\gamma_{n+1}}^1 \left((\psi_{\equiv}^{\mathfrak{t}}(\gamma_n, \gamma_{n+1})) \wedge \psi_{\prec}^{\mathfrak{p}}(\gamma_n, \gamma_{n+1}) \rightarrow (\gamma_{n+1} \leftrightarrow c^{-1}(\flat)) \right) \end{aligned}$$

► **Claim (g).** Let T_α be a tableau. Then $T \models \psi_{\text{input}}(\alpha)$ if and only if $C_{T_\alpha}(1, 1) = (q_0, x_1)$, $C_{T_\alpha}(1, i) = x_i$ for $2 \leq i \leq n$, and $C_{T_\alpha}(1, i) = \flat$ for $n < i \leq N$.

Until now, we ignored the fact that M alternates between universal and existential branching polynomially often. To simulate this, we quantify polynomially many tableaus in an alternating fashion, each containing a part of the computation of M .

Each of these tableaus should possess a *tail configuration*, which is the configuration where M either accepts, rejects, or alternates from existential to universal branching or vice versa. Formally, a number $i \in \{1, \dots, N\}$ is a *tail index* of C if there exists j such that either

1. $C(i, j)$ has an accepting or rejecting state,
2. or $C(i, j)$ has an existential state and there are $i' < i$ and j' with a universal state in $C(i', j')$,
3. or $C(i, j)$ has a universal state and there are $i' < i$ and j' with an existential state in $C(i', j')$.

The least such i is called *first tail index*, and the corresponding configuration is the *first tail configuration*.

The idea is that we can split the computation of M into multiple tableaus if any tableau (except the initial one) contains a run that continues from the previous tableau's first tail configuration.

We formalize the above as follows. Assume that T_α is a tableau, and that $T_\beta = \{w\}$ with $\ell(w) = (i, j)$ for some i . Then the formula $\psi_{\text{tail}}(\alpha, \beta)$ is meant to be true if and only if the i -th row of C_{T_α} is a tail configuration. Roughly speaking, with the parameters α and β we pass to $\psi_{\text{tail}}(\alpha, \beta)$ a tableau (viz. T_α) and the index of a row (viz. i). By using the shortcut

$$Q'\text{-state}(\beta) := \bigvee_{(q, a) \in Q' \times \Gamma} (\beta \leftrightarrow c^{-1}(q, a)),$$

we check if a given singleton $T_\beta = \{w\}$ encodes an accepting, rejecting, existential, universal, or an arbitrary state by setting Q' to Q_{acc} , Q_{rej} , Q_{\exists} , Q_{\forall} or Q , respectively. As a result, we can define:

$$\psi_{\text{first-tail}}(\alpha, \beta) := \psi_{\text{tail}}(\alpha, \beta) \wedge \sim \exists_{\gamma_1}^1 \left(\psi_{\prec}^{\mathfrak{t}}(\gamma_1, \beta) \wedge \psi_{\text{tail}}(\alpha, \gamma_1) \right)$$

$$\begin{aligned} \psi_{\text{tail}}(\alpha, \beta) := & \exists_{\gamma_0}^{\approx \alpha} \exists_{\alpha}^1 \psi_{\Xi}^{\text{t}}(\alpha, \beta) \wedge Q\text{-state}(\alpha) \wedge \left[Q_{\text{acc-state}}(\alpha) \otimes Q_{\text{rej-state}}(\alpha) \otimes \right. \\ & \left. \exists_{\gamma_0}^1 \left(\psi_{\prec}^{\text{t}}(\gamma_0, \alpha) \wedge (Q_{\exists\text{-state}}(\alpha) \wedge Q_{\forall\text{-state}}(\gamma_0)) \otimes (Q_{\forall\text{-state}}(\alpha) \wedge Q_{\exists\text{-state}}(\gamma_0)) \right) \right] \end{aligned}$$

► **Claim (h).** Suppose that T_α is a tableau, $T_\beta = \{w\}$, and $\ell(w) = (i, j)$.

Then $T \models \psi_{\text{tail}}(\alpha, \beta)$ if and only if i is a tail index of C_{T_α} ; and $T \models \psi_{\text{first-tail}}(\alpha, \beta)$ if and only if i is the first tail index of C_{T_α} .

Formally, given a run C of M that has a tail configuration, C *accepts* if the state q in its first tail configuration is in Q_{acc} , C *rejects* if $q \in Q_{\text{rej}}$, and C *alternates* otherwise. That a run of the form C_{T_α} accepts resp. rejects is expressed by

$$\begin{aligned} \psi_{\text{acc}}(\alpha) &:= \exists_{\gamma_2}^{\approx \alpha} \exists_{\gamma_2}^1 Q_{\text{acc-state}}(\gamma_2) \wedge \psi_{\text{first-tail}}(\alpha, \gamma_2), \\ \psi_{\text{rej}}(\alpha) &:= \exists_{\gamma_2}^{\approx \alpha} \exists_{\gamma_2}^1 Q_{\text{rej-state}}(\gamma_2) \wedge \psi_{\text{first-tail}}(\alpha, \gamma_2). \end{aligned}$$

In this formula, first the tableau T_α is copied to T_{γ_2} to extract with $\exists_{\gamma_2}^1$ the world carrying an accepting/rejecting state, while $\psi_{\text{first-tail}}(\alpha, \gamma_2)$ ensures that no alternation or rejecting/accepting state occurs at some earlier point in C_{T_α} . If the first tail configuration of the run contains an alternation, and if the run was existentially quantified, then it should be continued in a universally quantified tableau, and vice versa. The following formula expresses, given two tableaux T_α, T_β , that C_{T_β} is a *continuation* of C_{T_α} , i.e., that the first configuration of C_{T_β} equals the first tail configuration of C_{T_α} . In other words, if i is the first tail index of C_{T_α} , then $C_{T_\alpha}(i, j) = C_{T_\beta}(1, j)$ for all $j \in \{1, \dots, N\}$.

$$\begin{aligned} \psi_{\text{cont}}(\alpha, \beta) := & \exists_{\gamma_2}^1 \psi_{\text{first-tail}}(\alpha, \gamma_2) \wedge \forall_{\alpha}^1 \forall_{\beta}^1 \\ & \left[\left(\psi_{\text{min}}^{\text{t}}(\beta) \wedge \psi_{\Xi}^{\text{t}}(\alpha, \gamma_2) \wedge \psi_{\Xi}^{\text{p}}(\alpha, \beta) \right) \rightarrow \bigwedge_{e \in \Xi} (\alpha \vee \beta) \leftrightarrow = (e) \right] \end{aligned}$$

The above formula first obtains the first tail index i of C_{T_α} and stores it in a singleton $y \in T_{\gamma_2}$. Then for all worlds $w \in T_\alpha$ and $v \in T_\beta$, where v is t -minimal (i.e., in the first row) and w is in the same row as y , and which additionally agree on their p -component, the third line states that w and v agree on Ξ . Altogether, the i -th row of C_{T_α} and the first row of C_{T_β} then have to coincide.

The number of alternations is polynomially bounded, i.e., M performs at most $r(n) - 1$ alternations for a polynomial r . In other words, we require at most $r = r(n)$ tableaux, which we call $\alpha_1, \dots, \alpha_r$. In the following, the formula $\psi_{\text{run}, i}$ describes the behaviour of the i -th run. W.l.o.g. r is even and $q_0 \in Q_{\exists}$. We may then define the final run by

$$\psi_{\text{run}, r} := \forall_{\alpha_r}^{\subseteq} \left[\left(\psi_{\text{legal}}(\alpha_r) \wedge \psi_{\text{cont}}(\alpha_{r-1}, \alpha_r) \right) \rightarrow \left(\sim \psi_{\text{rej}}(\alpha_r) \wedge \psi_{\text{acc}}(\alpha_r) \right) \right].$$

For $1 < i < r$ and even i , let

$$\psi_{\text{run}, i} := \forall_{\alpha_i}^{\subseteq} \left[\left(\psi_{\text{legal}}(\alpha_i) \wedge \psi_{\text{cont}}(\alpha_{i-1}, \alpha_i) \right) \rightarrow \left(\sim \psi_{\text{rej}}(\alpha_i) \wedge (\psi_{\text{acc}}(\alpha_i) \otimes \psi_{\text{run}, i+1}) \right) \right]$$

and for $1 < i < r$ and odd i

$$\psi_{\text{run}, i} := \exists_{\alpha_i}^{\subseteq} \left[\psi_{\text{legal}}(\alpha_i) \wedge \psi_{\text{cont}}(\alpha_{i-1}, \alpha_i) \wedge \sim \psi_{\text{rej}}(\alpha_i) \wedge \left(\psi_{\text{acc}}(\alpha_i) \otimes \psi_{\text{run}, i+1} \right) \right].$$

Analogously, the initial run is described by

$$\psi_{\text{run}, 1} := \exists_{\alpha_1}^{\subseteq} \left(\psi_{\text{legal}}(\alpha_1) \wedge \psi_{\text{input}}(\alpha_1) \wedge \sim \psi_{\text{rej}}(\alpha_1) \wedge \left(\psi_{\text{acc}}(\alpha_1) \otimes \psi_{\text{run}, 2} \right) \right)$$

Let us state the set $\Psi \subseteq \mathcal{PS}$ of all relevant scopes and the set $\Psi' \subseteq \Psi$ of scopes that accommodate pre-tableaus:

$$\begin{aligned}\Psi &:= \{\mathfrak{s}_i \mid 0 \leq i \leq k\} \cup \{\mathfrak{s}'_k\} \cup \{\gamma_i \mid 0 \leq i \leq n+1\} \cup \{\alpha_i \mid 1 \leq i \leq r\} \\ \Psi' &:= \{\gamma_i \mid 0 \leq i \leq n+1\} \cup \{\alpha_i \mid 1 \leq i \leq r\}\end{aligned}$$

W.l.o.g. $n \geq 5$, as $\gamma_1, \dots, \gamma_6$ are always used. Then we ultimately define

$$\varphi_x := \text{canon}' \wedge \text{scopes}_k(\Psi) \wedge \bigwedge_{p \in \Psi'} \psi_{\text{pre-tableau}}(p) \wedge \psi_{\text{run},1},$$

which is an MTL_k -formula since we deliberately omitted the conjunct $\Box^{k+1}\perp$ here. However, by Lemma 5.5, φ_x is satisfiable if and only if $\varphi_x \wedge \Box^{k+1}\perp$ is satisfiable. Finally, it is not hard using the above claims to prove that $\varphi_x \wedge \Box^{k+1}\perp$ is satisfiable if and only if M accepts x .


A Decidable Fragment of Second Order Logic With Applications to Synthesis

P. Madhusudan

University of Illinois, Urbana Champaign, Urbana, IL, USA
madhu@illinois.edu

Umang Mathur

University of Illinois, Urbana Champaign, Urbana, IL, USA
umathur3@illinois.edu

 <https://orcid.org/0000-0002-7610-0660>

Shambwaditya Saha

University of Illinois, Urbana Champaign, Urbana, IL, USA
ssaha6@illinois.edu

Mahesh Viswanathan

University of Illinois, Urbana Champaign, Urbana, IL, USA
vmahesh@illinois.edu

Abstract

We propose a fragment of many-sorted second order logic called EQSMT and show that checking satisfiability of sentences in this fragment is decidable. EQSMT formulae have an $\exists^*\forall^*$ quantifier prefix (over variables, functions and relations) making EQSMT conducive for modeling synthesis problems. Moreover, EQSMT allows reasoning using a combination of background theories provided that they have a decidable satisfiability problem for the $\exists^*\forall^*$ FO-fragment (e.g., linear arithmetic). Our decision procedure reduces the satisfiability of EQSMT formulae to satisfiability queries of $\exists^*\forall^*$ formulae of each individual background theory, allowing us to use existing efficient SMT solvers supporting $\exists^*\forall^*$ reasoning for these theories; hence our procedure can be seen as *effectively quantified SMT* (EQSMT) reasoning.

2012 ACM Subject Classification Theory of computation \rightarrow Logic and verification

Keywords and phrases second order logic, synthesis, decidable fragment

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.31

Funding This work has been supported by NSF grants 1422798, 1329991, 1138994 and 1527395.

1 Introduction

The goal of program synthesis is to automatically construct a program that satisfies a given specification. This problem has received a lot of attention from the research community in recent years [33, 4, 14]. Several different approaches have been proposed to address this challenge (see [4, 17] for some of these). One approach to program synthesis is to reduce the problem to the satisfiability problem in a decidable logic by constructing a sentence whose existentially quantified variables identify the program to be synthesized, and the inner formula expresses the requirements that the program needs to meet.

This paper furthers this research program by identifying a decidable *second-order* logic that is suitable for encoding problems in program synthesis. To get useful results, one needs to constrain the semantics of functions and relations used in encoding the synthesis problem. Therefore our logic has a set of *background theories*, where each of the background theories is



© P. Madhusudan, Umang Mathur, Shambwaditya Saha, and Mahesh Viswanathan;
licensed under Creative Commons License CC-BY

27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 31; pp. 31:1–31:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

assumed to be independently axiomatized and equipped with a solver. Finally, to leverage the advances made by logic solvers, our aim is to develop a decision procedure for our logic that makes *black-box* calls to the decision procedures (for $\exists^*\forall^*$ satisfiability) for the background theories.

With the above goal in mind, let us describe our logic. It is a *many-sorted* logic that can be roughly described as an *uninterpreted combination of theories* (UCT) [20]. A UCT has a many-sorted universe where there is a special sort σ_0 that is declared to be a *foreground* sort, while the other sorts $(\sigma_1, \dots, \sigma_n)$ are declared to be background sorts. We assume that there is some fixed signature of functions, relations, and constants over each individual background sort that is purely over that sort. Furthermore, we assume that each background sort σ_i ($i > 0$) comes with an associated *background theory* T_i ; T_i can be arbitrary, even infinite, but is constrained to formulae with functions, relations and constants that only involve the background sort σ_i . Our main contribution is a decidability result for the satisfiability problem modulo these theories for boolean combinations of sentences of the form

$$(\exists \mathbf{x})(\exists \mathcal{R})(\exists \mathbf{F})(\forall \mathbf{y})(\forall \mathcal{P})(\forall \mathbf{G})\psi, \quad (1)$$

- \mathbf{x} is a set of existentially quantified first order variables. These variables can admit values in any of the sorts (background or foreground);
- \mathcal{R} is a set of existentially quantified relational variables, whose arguments are restricted to be over the foreground sort σ_0 ;
- \mathbf{F} is a set of existentially quantified function variables, which take as arguments elements from the foreground sort σ_0 , and return a value in any of the background sorts σ_i ;
- \mathbf{y} is a set of universally quantified first order variables over any of the sorts;
- \mathcal{P} is a set of universally quantified relational variables, whose arguments could be of any of the sorts; and
- \mathbf{G} is a set of universally quantified function variables, whose arguments can be from any sort and could return values of any sort.

Thus our logic has sentences with prefix $\exists^*\forall^*$, allowing for quantification over both first order variables and second-order variables (relational and functional). To obtain decidability, we have carefully restricted the sorts (or *types*) of second-order variables that are existentially and universally quantified, as described above.

Our decidability result proceeds as follows. By crucially exploiting the disjointness of the universes of background theories and through a series of transformations, we reduce the satisfiability problem for our logic to the satisfiability of several pure $\exists^*\forall^*$ *first-order* logic formulas over the individual background theories T_1, \dots, T_n . Consequently, if the background theories admit (individually) a decidable satisfiability problem for the first-order $\exists^*\forall^*$ fragment, then satisfiability for our logic is *decidable*. Examples of such background theories include Presburger arithmetic, the theory of real-closed fields, and the theory of linear real arithmetic. Our algorithm for satisfiability makes finitely many black-box calls to the engines for the individual background theories.

Salient aspects of our logic and our decidability result

Design for decidability. Our logic is defined to carefully avoid the undecidability that looms in any logic of such power. We do not know of any decidable second-order logic fragment that supports background theories such as arithmetic and uninterpreted functions. While *quantifier-free* decidable logics can be combined to get decidable logics using Nelson-Oppen combinations [23], or local theory extensions [32], combining quantified logics is notoriously

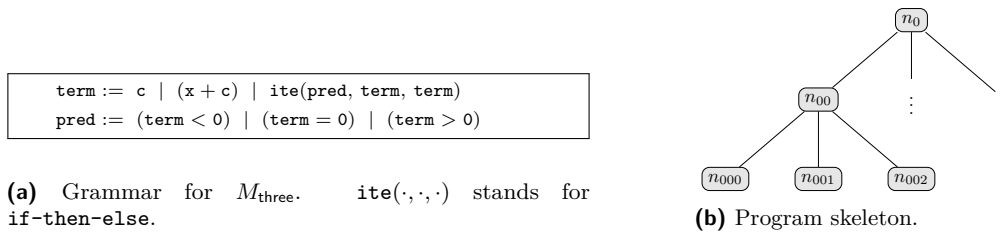
hard, and there are only few restricted classes of first-order logic that are known to be decidable.

Our design choice forces *communication* between theories using the foreground sort, keeping the universes of the different sorts *disjoint*, which allows a decidable combination of $\exists^*\forall^*$ theories. We emphasize that, unlike existing work on quantified first-order theories that are decidable by reduction to quantifier-free SMT, our logic allows existential and universal quantification over the background theories as well, and the decision procedure reduces satisfiability to $\exists^*\forall^*$ fragment of the underlying theories. Our result can hence be seen as a decidable combination of $\exists^*\forall^*$ theories that further supports second-order quantification.

Undecidable Extensions. We show that our logic is on the edge of the decidability barrier, by showing that lifting some of the restrictions we have will render the logic undecidable. In particular, we show that if we allow outer existential quantification over functions (which is related to the condition demanding that all function variables are universally quantified in the inner block of quantifiers), then satisfiability of the logic is undecidable. Second, if we lift the restriction that the underlying background sorts are pairwise disjoint, then again the logic becomes undecidable. The design choices that we have made hence seem crucial for decidability.

Expressing Synthesis Problems. Apart from decidability, a primary motivational design principle of our logic is to express *synthesis* problems. Synthesis problems typically can be expressed in $\exists^*\forall^*$ fragments, where we ask whether there exists an object of the kind we wish to synthesize (using the block of existential quantifiers) such that the object satisfies certain properties (expressed by a universally quantified formula). For instance, if we are synthesizing a program snippet that is required to satisfy a Hoare triple (pre/post condition), we can encode this by asking whether there is a program snippet such that for all values of variables (modeling the input to the snippet), the verification condition corresponding to the Hoare triple holds. In this context, the existentially quantified variables (first order and second order) can be used to model program snippets. Furthermore, since our logic allows second-order universal quantification over *functions*, we can model aspects of the program state that require *uninterpreted functions*, in particular pointer fields that model the heap.

Evaluation on Synthesis Problems. We illustrate the applicability of our logic for two classes of synthesis problems. The first class involves synthesizing *recursive programs* that work over inductive data-structures. Given the precise pre/post condition for the program to be synthesized, we show how to model recursive program synthesis by synthesizing only a straight-line program (by having the output of recursive calls provided as inputs to the straightline program). The verification condition of the program requires universal quantification over both scalar variables as well as heap pointers, modeled as uninterpreted functions. Since such verification-conditions are already very expressive (even for the purpose of verification), we adapt a technique in the literature called *natural proofs* [20, 28, 25], that soundly abstracts the verification condition to a decidable theory. This formulation still has universal quantification over variables and functions, and combines standard background theories such as arithmetic and theory of uninterpreted functions. We then show that synthesis of bounded-sized programs (possibly involving integer constants that can be unbounded) can be modeled in our logic. In this modeling, the universal quantification over functions plays a crucial role in modeling the pointers in heaps, and modeling uninterpreted relations that capture inductive data-structure predicates (such as `lseg`, `bstree`, etc.).



■ **Figure 1** Synthesizing M_{three} using EQSMT.

The second class of synthesis involves taking a recursive definition of a function, and synthesizing a non-recursive (and iteration free) function equivalent to it. In our modeling, the existential quantification over the foreground sort as well as the background sort of integers is utilized, as the synthesized function involves integers.

The crux of our contribution, therefore, is providing a decidable logic that can express synthesis problems succinctly. Such a logic promises to provide a useful interface between researchers working on practical synthesis applications and researchers working on engineering efficient tools for solving them, similar to the role SMT plays in verification.

2 Motivating EQSMT for synthesis applications

In program synthesis, the goal is to search for programs, typically of bounded size, that satisfy a given specification. The \exists -Block of an EQSMT formula can be used to express the search for the syntactic program. The inner formula, then, must interpret the semantics of this syntactic program, and express that it satisfies the specification. If the specification is a universally quantified formula, then, we can encode the synthesis problem in EQSMT.

One of the salient features of the fragment EQSMT is the ability to universally quantify over functions and relations. Often, specifications for programs, such as those that manipulate heaps, involve a universal quantification over uninterpreted functions (that model pointers). EQSMT aptly provides this functionality, while still remaining within the boundaries of decidability. Further, EQSMT supports combination of background theories/sorts; existential quantification over these sorts can thus be used to search for programs with arbitrary elements from these background sorts. As a result, the class of target programs that can be expressed by an EQSMT formula is infinite. Consequently, when our decision procedure returns unsatisfiable, we are assured that no program (from an infinite class of programs) exists, (most CEGIS solvers for program synthesis cannot provide such a guarantee.)

We now proceed to give a concrete example of a synthesis problem which will demonstrate the power of EQSMT. Consider the specification of the following function M_{three} , which is a slight variant of the classical McCarthy’s 91 function [22], whose specification is given below.

$$M_{\text{three}}(n) = \begin{cases} n - 30 & \text{if } n > 13 \\ M_{\text{three}}(M_{\text{three}}(M_{\text{three}}(n + 61))) & \text{otherwise} \end{cases} \quad (2)$$

We are interested in synthesizing a straight line program that implements the recursive function M_{three} , and can be expressed as a **term** over the grammar specified in Figure 1a.

Here, we only briefly discuss how to encode this synthesis problem in EQSMT, and the complete details can be found in Appendix A. First, let us fix the maximum height of the term we are looking for, say to be 2. Then, the program we want to synthesize can be represented as a tree of height at most 2 such that every node in the tree can have ≤ 3 child nodes (because the maximum arity of any function in the above grammar is 3, corresponding to **ite**). The skeleton of such an expression tree is shown in Figure 1b. Every node in the tree is named according to its path from the root node.

The synthesis problem can then be encoded as the following formula

$$\begin{aligned} \phi_{M_{\text{three}}} \equiv & (\exists n_0, n_{00}, n_{01}, \dots, n_{022} : \sigma_0) (\exists \text{Left, Mid, Right} : \sigma_0 \sigma_0) \\ & (\exists \text{ADD, ITE, LTZero, EQZero, GTZero, INPUT, } C_1, C_2, C_3 : \sigma_{\text{label}}) \\ & (\exists c_1, c_2, c_3 : \mathbb{N}) (\exists f_{\text{label}} : \sigma_0, \sigma_{\text{label}}) \\ & \varphi_{\text{well-formed}} \wedge (\forall x : \mathbb{N}) (\forall g_{\text{val}} : \sigma_0, \mathbb{N}) (\varphi_{\text{semantics}} \implies \varphi_{\text{spec}}) \end{aligned}$$

Here, the nodes n_0, n_{00}, \dots are elements of the foreground sort σ_0 . The binary relations *Left, Mid, Right* over the foreground sort will be used to assert that a node n is the left, middle, right child respectively of node n' : $\text{Left}(n', n)$, $\text{Mid}(n', n)$, $\text{Right}(n', n)$. The operators or *labels* for nodes belong to the background sort σ_{label} , and can be one of ADD (+), ITE (*ite*), LTZero (< 0), GTZero (> 0), (EQZero ($= 0$)), INPUT (denoting the input to our program), or constants C_1, C_2, C_3 (for which we will synthesize natural constants c_1, c_2, c_3 in the (infinite) background sort \mathbb{N}). The function f_{label} assigns a label to every node in the program, and the formula $\varphi_{\text{well-formed}}$ asserts some sanity conditions:

$$\begin{aligned} \varphi_{\text{well-formed}} \equiv & \bigwedge_{\rho \neq \rho'} n_\rho \neq n_{\rho'} \wedge \text{Left}(n_0, n_{00}) \wedge \bigwedge_{\rho \neq 00} \neg(\text{Left}(n_0, n_\rho)) \wedge \dots \\ & \wedge \neg(\text{ADD} = \text{ITE}) \wedge \neg(\text{ADD} = \text{LTZero}) \wedge \dots \wedge \neg(C_1 = C_3) \wedge \neg(C_2 = C_3) \\ & \wedge \bigwedge_{\rho} (f_{\text{label}}(n_\rho) = \text{ADD}) \vee (f_{\text{label}}(n_\rho) = \text{ITE}) \vee \dots \vee (f_{\text{label}}(n_\rho) = C_3) \end{aligned}$$

The formula $\varphi_{\text{semantics}}$ asserts that the “meaning” of the program can be inferred from the meaning of the components of the program. We will use the function g_{val} , that assigns value to nodes from \mathbb{N} , for this purpose :

$$\varphi_{\text{semantics}} \equiv \bigwedge_{\rho, \rho_1, \rho_2} \left[\begin{array}{l} (f_{\text{label}}(n_\rho) = \text{ADD} \wedge \text{Left}(n_\rho, n_{\rho_1}) \wedge \text{Mid}(n_\rho, n_{\rho_2})) \\ \implies g_{\text{val}}(n_\rho) = g_{\text{val}}(n_{\rho_1}) + g_{\text{val}}(n_{\rho_2}) \\ \vdots \\ \wedge f_{\text{label}}(n_\rho) = C_3 \implies g_{\text{val}}(n_\rho) = c_3 \end{array} \right]$$

Finally, the formula φ_{spec} expresses the specification of the program as in Equation (2). A complete description is provided in Appendix A.

Observe that the formula $\phi_{M_{\text{three}}}$ has existential and universal quantification over functions and relations, as allowed by our decidable fragment EQSMT. The existentially quantified functions map the foreground sort σ_0 to one of the background sorts, and the existentially quantified relations span only over the foreground sort.

We encoded the above EQSMT formula in the **z3** [12] SMT solver (see Section 6 for details), which synthesized the expression $\text{fun}(n) = \text{ite}(n > 13, n - 30, -16)$. In Section 6, we show that we can synthesize a large class of such programs amongst others.

3 Many-sorted Second Order Logic and the EQSMT Fragment

We briefly recall the syntax and semantics of general many-sorted second order logic, and then present the EQSMT fragment of second order logic.

Many-sorted second-order logic

A many-sorted signature is a tuple $\Sigma = (\mathcal{S}, \mathcal{F}, \mathfrak{R}, \mathcal{V}, \mathcal{V}^{\text{fun}}, \mathcal{V}^{\text{rel}})$ where, \mathcal{S} is a nonempty finite set of sorts, $\mathcal{F}, \mathfrak{R}, \mathcal{V}, \mathcal{V}^{\text{fun}}, \mathcal{V}^{\text{rel}}$ are, respectively, sets of function symbols, relation symbols, first order variables, function variables and relation variables. Each variable $x \in \mathcal{V}$

is associated with a sort $\sigma \in \mathcal{S}$, represented as $x : \sigma$. Each function symbol or function variable also has an associated type $(w, \sigma) \in \mathcal{S}^* \times \mathcal{S}$, and each relation symbol and relation variable has a type $w \in \mathcal{S}^+$. We assume that the set of symbols in \mathcal{F} and \mathfrak{R} are either finite or countably infinite, and that \mathcal{V} , \mathcal{V}^{fun} , and \mathcal{V}^{rel} are all countably infinite. Constants are modeled using 0-ary functions. We say that Σ is *unsorted* if \mathcal{S} consists of a single sort.

Terms over a many-sorted signature Σ have an associated sort and are inductively defined by the grammar

$$t : \sigma \quad := \quad x : \sigma \quad | \quad f(t_1 : \sigma_1, t_2 : \sigma_2, \dots, t_m : \sigma_m) \quad | \quad F(t_1 : \sigma_1, t_2 : \sigma_2, \dots, t_n : \sigma_n)$$

where $f : (\sigma_1 \sigma_2 \dots \sigma_m, \sigma) \in \mathcal{F}$, and $F : (\sigma_1 \sigma_2 \dots \sigma_n, \sigma) \in \mathcal{V}^{\text{fun}}$. Formulae over Σ are inductively defined as

$$\begin{aligned} \phi \quad := \quad & \perp \quad | \quad \phi \Rightarrow \phi \quad | \quad t : \sigma = t' : \sigma \quad | \quad R(t_1 : \sigma_1, t_2 : \sigma_2, \dots, t_m : \sigma_m) \quad | \\ & \mathcal{R}(t_1 : \sigma_1, t_2 : \sigma_2, \dots, t_n : \sigma_n) \quad | \quad (\exists x : \sigma) \phi \quad | \quad (\exists F : w, \sigma) \phi \quad | \quad (\exists \mathcal{R}' : w) \phi \end{aligned}$$

where $R : (\sigma_1 \sigma_2 \dots \sigma_m) \in \mathfrak{R}$, $\mathcal{R}, \mathcal{R}'$ are relation variables, F is a function variable, of appropriate types. Note that equality is allowed only for terms of same sort. A formula is said to be *first-order* if it does not use any function or relation variables.

The semantics of many sorted logics are described using many-sorted structures. A Σ -structure is a tuple $\mathcal{M} = (\mathcal{U}, \mathcal{I})$ where $\mathcal{U} = \{M_\sigma\}_{\sigma \in \mathcal{S}}$ is a collection of pairwise disjoint \mathcal{S} indexed universes, and \mathcal{I} is an interpretation function that maps each each variable $x : \sigma$ to an element in the universe M_σ , each function symbol and each function variable to a function of the appropriate type on the underlying universe. Similarly, relation symbols and relation variables are also assigned relations of the appropriate type on the underlying universe. For an interpretation \mathcal{I} , as is standard, we use $\mathcal{I}[c^x/x]$ to denote the interpretation that maps x to c^x , and is otherwise identical to \mathcal{I} . For function variable F and relation variable \mathcal{R} , $\mathcal{I}[f^F/F]$ and $\mathcal{I}[R^{\mathcal{R}}/\mathcal{R}]$ are defined analogously.

Interpretation of terms in a model is the usual one obtained by interpreting variables, functions, and function variables using their underlying interpretation in the model; we skip the details. The satisfaction relation $\mathcal{M} \models \phi$ is also defined in the usual sense, and we will skip the details.

A first-order theory is a tuple $T = (\Sigma_T, \mathcal{A}_T)$, where \mathcal{A}_T is a set of (possibly infinite) first-order sentences. Theory T is *complete* if every sentence α or its negation is entailed by \mathcal{A}_T , i.e., either every model satisfying \mathcal{A}_T satisfies α , or every model satisfying \mathcal{A}_T satisfies $\neg\alpha$. A theory \mathcal{A}_T is *consistent* if it is not the case that there is a sentence α such that both α and $\neg\alpha$ are entailed.

The logic EQSMT

We now describe EQSMT, the fragment of many-sorted second order logic that we prove decidable in this paper and that we show can model synthesis problems.

Let $\Sigma = (\mathcal{S}, \mathcal{F}, \mathfrak{R}, \mathcal{V}, \mathcal{V}^{\text{fun}}, \mathcal{V}^{\text{rel}})$ be a many sorted signature. Σ is a *pure signature* if (a) the type of every function symbol and every relation symbol is over a single sort (however, function variables and relation variables are allowed to mix sorts), (b) there is a special sort σ_0 (which we call the *foreground sort*, while other sorts $\sigma_1, \dots, \sigma_k$ are called *background sorts*) and (c) there are no function or relation symbols involving σ_0 .

The fragment EQSMT is the set of sentences defined over a pure signature Σ , with foreground sort σ_0 and background sorts $\sigma_1, \dots, \sigma_k$, by the following grammar

$$\phi \quad := \quad \varphi \quad | \quad \exists(x : \sigma) \phi \quad | \quad (\exists \mathcal{R} : w) \phi \quad | \quad (\exists F : w, \sigma_i) \phi$$

where, $\sigma \in \mathcal{S}$, $w \in \sigma_0^+$ (i.e., only foreground sort), $1 \leq i \leq k$, and φ is a universally quantified formula defined by the grammar

$$\varphi := \psi \mid \forall(y : \sigma) \varphi \mid (\forall \mathcal{R} : w') \varphi \mid (\forall F : w', \sigma) \varphi$$

where, $\sigma \in \mathcal{S}$, $w' \in \mathcal{S}^+$, and ψ is quantifier free over Σ .

The formulas above consist of an existential quantification block followed by a universal quantification block. The existential block can have first-order variables of any sort, relation variables that are over the foreground sort only, and function variables that map tuples of the foreground sort to a background sort. The inner universal block allows all forms of quantification – first-order variables, function variables, and relation variables of all possible types. The inner formula is quantifier-free. We will restrict our attention to *sentences* in this logic, i.e., we will assume that all variables (first-order/function/relation) are quantified. We will denote by \mathbf{x}_i (resp. \mathbf{y}_i), the set of existentially (resp. universally) quantified first order variables of sort σ_i , for every $0 \leq i \leq k$.

The problem

The problem we consider is that of deciding satisfiability of EQSMT sentences with *background theories* for the background sorts. First we introduce some concepts.

An *uninterpreted combination of theories (UCT)* over a pure signature, with $\{\sigma_0, \sigma_1, \dots, \sigma_k\}$ as the set of sorts, is the union of theories $\{T_{\sigma_i}\}_{1 \leq i \leq k}$, where each T_{σ_i} is a theory over signature σ_i . A sentence ϕ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable if there is a multi-sorted structure \mathcal{M} that satisfies ϕ and all the sentences in $\bigcup_{i=1}^k T_{\sigma_i}$.

The satisfiability problem for EQSMT with background theories is the following. Given a UCT $\{T_{\sigma_i}\}_{1 \leq i \leq k}$ and a sentence $\phi \in \text{EQSMT}$, determine if ϕ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable. We show that this is a decidable problem, and furthermore, there is a decision procedure that uses a finite number of black-box calls to satisfiability solvers of the underlying theories to check satisfiability of EQSMT sentences.

For the rest of this paper, for technical convenience, we will assume that the boolean theory T_{bool} is one of the background theories. This means $\text{bool} \in \mathcal{S}$ and the constants $\top : \text{bool}, \perp : \text{bool} \in \mathcal{F}$. The set of sentences in T_{bool} is $\mathcal{A}_{\text{bool}} = \{\top \neq \perp, \forall(y : \text{bool}) \cdot (y = \top \vee y = \perp)\}$. Note that checking satisfiability of a $\exists^* \forall^*$ sentence over T_{bool} is decidable.

4 The Decision Procedure for EQSMT

In this section we present our decidability result for sentences over EQSMT in presence of background theories. Let us first state the main result of this paper.

► **Theorem 1.** *Let Σ be a pure signature with foreground sort σ_0 and background sorts $\sigma_1, \dots, \sigma_k$. Let $\{T_{\sigma_i}\}_{1 \leq i \leq k}$ be a UCT such that, for each i , checking T_{σ_i} -satisfiability of $\exists^* \forall^*$ first-order sentences is decidable. Then the problem of checking $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiability of EQSMT sentences is decidable.*

We will prove the above theorem by showing that any given EQSMT sentence ϕ over a UCT signature Σ can be transformed, using a sequence of satisfiability preserving transformation steps, to the satisfiability of $\exists \forall$ first-order formulae over the individual theories.

We give a brief overview of the sequence of transformations (Steps 1 through 4). In Step 1, we replace the occurrence of every relation variable \mathcal{R} (quantified universally or existentially) of sort w by a function variable F of sort (w, bool) . Note that doing this for the outer

existentially quantified relation variables keeps us within the syntactic fragment. In Step 2, we eliminate function variables that are existentially quantified. This crucially relies on the *small model property* for the foreground universe, similar to EPR [5]. This process, however, adds both existential first-order variables and universally quantified function variables. In Step 3, we eliminate the universally quantified function variables using a standard Ackermann reduction [27], which adds more universally quantified first-order variables.

The above steps result in a first-order $\exists^*\forall^*$ sentence over the combined background theories, and the empty theory for the foreground sort. In Step 4, we show that the satisfiability of such a formula can be reduced to a finite number of satisfiability queries of $\exists^*\forall^*$ sentences over *individual* theories.

Step 1: Eliminating relation variables

The idea here is to introduce, for every relational variable \mathcal{R} (with type w), a function variable $f^{\mathcal{R}}$ (with type $(w, \sigma_{\text{bool}})$) that corresponds to the characteristic function of \mathcal{R} .

Let ϕ be EQSMT formula over Σ . We will transform ϕ to an EQSMT formula $\phi^{\text{Step-1}}$ over the same signature Σ . Every occurrence of an atom of the form $\mathcal{R}(t_1:\sigma_{i_1}, \dots, t_k:\sigma_{i_k})$ in ϕ , is replaced by $f^{\mathcal{R}}(t_1:\sigma_{i_1}, \dots, t_k:\sigma_{i_k}) = \top$ in $\phi^{\text{Step-1}}$. Further, every quantification $Q(\mathcal{R} : w)$ is replaced by $Q(f^{\mathcal{R}} : w, \text{bool})$, where $Q \in \{\forall, \exists\}$. Thus, the resultant formula $\phi^{\text{Step-1}}$ has no relation variables. Further, it is a EQSMT formula, since the types of the newly introduced existentially quantified function variables are of the form $(\sigma_0^+, \sigma_{\text{bool}})$. The correctness of the above transformation is captured by the following lemma.

► **Lemma 2.** ϕ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable iff $\phi^{\text{Step-1}}$ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable.

Step 2: Eliminating existentially quantified function variables

We first note a small-model property with respect to the foreground sort for EQSMT sentences. This property crucially relies on the fact that existentially quantified function variables do not have their ranges over the foreground sort.

► **Lemma 3** (Small-model property for σ_0). *Let ϕ be an EQSMT sentence with foreground sort σ_0 and background sorts $\sigma_1, \dots, \sigma_k$. Let n be the number of existentially quantified first-order variables of sort σ_0 in ϕ . Then, ϕ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable iff there is a structure $\mathcal{M} = (\{M_{\sigma_i}\}_{i=0}^k, \mathcal{I})$, such that $|M_{\sigma_0}| \leq n$, $\mathcal{M} \models \bigcup_{i=1}^k T_{\sigma_i}$ and $\mathcal{M} \models \phi$.*

Proof (Sketch). We present the more interesting direction here. Consider a model $\mathcal{M} = (\mathcal{U}, \mathcal{I})$ such that $\mathcal{M} \models \bigcup_{i=1}^k T_{\sigma_i}$ and $\mathcal{M} \models \phi$. Let \mathcal{I}_{\exists} be the interpretation function that extends \mathcal{I} so that $(\mathcal{U}, \mathcal{I}_{\exists}) \models \varphi$, where φ is the inner universally quantified subformula of ϕ . Let $U = \{\mathcal{I}_{\exists}(x) \in M_{\sigma_0} \mid x \in \mathbf{x}_0\}$ be the restriction of the foreground universe to the interpretations of the variables \mathbf{x}_0 . Clearly, $|U| \leq |\mathbf{x}_0|$.

Let us first show that $(\mathcal{U}|_U, \mathcal{I}_{\exists}|_U) \models \varphi$. For this, first see that for every extension $\mathcal{I}_{\exists\forall}$ of \mathcal{I}_{\exists} with interpretations of all the universal FO variables, we must have $(\mathcal{U}, \mathcal{I}_{\exists\forall}) \models \psi$, where ψ is the quantifier free part of φ (and thus also of ϕ). Now, clearly $(\mathcal{U}, \mathcal{I}_{\exists\forall}) \models \psi$ must also hold for those extensions $\mathcal{I}_{\exists\forall}^U$ which map all universal variables in \mathbf{y}_0 to the set U and maps all universally quantified function variables of range sort σ_0 to function interpretations whose ranges are limited to the set U .

Thus, it must also be the case that when we restrict the universe M_{σ_0} to the set U , we have that $(\mathcal{U}|_U, \mathcal{I}_{\exists}|_U) \models \forall^* \psi$. This is because every universal extension \mathcal{I}' of $\mathcal{I}_{\exists}|_U$ is also a projection of one of these $\mathcal{I}_{\exists\forall}^U$ interpretations. ◀

The proof of the above statement shows that if there is a model that satisfies ϕ (in Lemma 3), then there is a model that satisfies ϕ and in which the foreground universe contains *only* elements that are interpretations of the first-order variables \mathbf{x}_0 over the foreground sort (and hence bounded). Consequently, instead of existentially quantifying over a function F (of arity r) from the foreground sort σ_0 to some background sort σ_i , we can instead quantify over first-order variables \mathbf{x}^F of sort σ_i that capture the image of these functions for each r -ary combination of \mathbf{x}_0 .

Let $\phi^{\text{Step-1}}$ be the EQSMT sentence over Σ obtained after eliminating relation variables. Let $\psi^{\text{Step-1}}$ be the quantifier free part (also known as the *matrix*) of $\phi^{\text{Step-1}}$. Now, define

$$\tilde{\psi} \equiv \psi_{\text{restrict}} \wedge \psi^{\text{Step-1}}, \text{ where, } \psi_{\text{restrict}} \equiv \bigwedge_{y \in \mathbf{y}_0} \left(\bigvee_{x \in \mathbf{x}_0} y = x \right).$$

Let $\tilde{\phi}$ the sentence obtained by replacing the matrix $\psi^{\text{Step-1}}$ in $\phi^{\text{Step-1}}$, by $\tilde{\psi}$. Then, the correctness of this transformation is noted below.

► **Lemma 4.** $\phi^{\text{Step-1}}$ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable iff $\tilde{\phi}$ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable.

We now eliminate the existentially quantified function variables in $\tilde{\phi}$, one by one. Let $\tilde{\phi} = (\exists F : \sigma_0^m, \sigma) \exists^* \forall^* \tilde{\psi}$, where σ is a background sort. For every m -tuple $t = (t[1], \dots, t[m])$ over the set \mathbf{x}_0 , we introduce a variable x_t^F of sort σ . Let \mathbf{x}^F be the set of all such n^m variables, where $n = |\mathbf{x}_0|$ is the number of existential first order variables of sort σ_0 in $\tilde{\phi}$. Next, we introduce a fresh function variable G^F of sort σ_0^m, σ , and quantify it universally. G^F will be used to *emulate* the function F . Let us define

$$\psi^{\text{Step-2}} \equiv (\forall G^F : \sigma_0^m, \sigma) (\psi_{\text{emulate}} \implies \tilde{\psi})$$

where, $\psi_{\text{emulate}} \equiv \bigwedge_{t \in \mathbf{x}_0^m} (G^F(t[1], \dots, t[m]) = x_t^F)$ and $\tilde{\psi}$ is obtained by replacing all occurrences of F in $\tilde{\psi}$ by G^F . Now define $\phi^{\text{Step-2}}$ to be the sentence

$$\phi^{\text{Step-2}} \equiv (\exists \mathbf{x}^F : \sigma) \exists^* \forall^* (\forall G^F : \sigma_0^m, \sigma) \psi^{\text{Step-2}}.$$

The following lemma states the correctness guarantee of this transformation.

► **Lemma 5.** $\phi^{\text{Step-2}}$ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable iff $\phi^{\text{Step-1}}$ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable.

Step 3: Eliminating universal function variables

The recipe here is to perform Ackermann reduction [2] for every universally quantified function variable.

Let $\phi^{\text{Step-2}} \equiv \exists^* \forall^* (\forall F : w, \sigma) \psi^{\text{Step-2}}$, where $\psi^{\text{Step-2}}$ is the quantifier free part of $\phi^{\text{Step-2}}$, and let $|w| = m$. For every term t of the form $F(t_1, \dots, t_m)$ in $\psi^{\text{Step-2}}$, we introduce a fresh first order variable $y_{(t_1, t_2, \dots, t_m)}^F$ of sort σ , and replace every occurrence of the term t in $\psi^{\text{Step-2}}$ with $y_{(t_1, t_2, \dots, t_m)}^F$. Let $\hat{\psi}$ be the resulting quantifier free formula. Let \mathbf{y}^F be the collection of all the newly introduced variables. Let us now define $\psi^{\text{Step-3}} \equiv (\psi_{\text{ack}} \implies \hat{\psi})$. Here, $\psi_{\text{ack}} \equiv \bigwedge_{y_t^F, y_{t'}^F \in \mathbf{y}^F} \left[\left(\bigwedge_{j=1}^m t_j = t'_j \right) \implies (y_t^F = y_{t'}^F) \right]$ where, $t = F(t_1, \dots, t_m), t' = F(t'_1, \dots, t'_m)$.

Then, the transformed formula $\phi^{\text{Step-3}} \equiv \exists^* \forall^* (\forall \mathbf{y}^F : \sigma) \psi^{\text{Step-3}}$ is correct:

► **Lemma 6.** $\phi^{\text{Step-2}}$ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable iff $\phi^{\text{Step-3}}$ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable.

Step-4: Decomposition and black box calls to $\exists^*\forall^*$ Theory solvers

The EQSMT sentence $\phi^{\text{Step-3}}$ obtained after the sequence of steps 1 through 3 is a first order $\exists^*\forall^*$ sentence over Σ . This sentence, however, may possibly contain occurrences of variables of the foreground sort σ_0 . Intuitively, the objective of this step is to decompose $\phi^{\text{Step-3}}$ into $\exists^*\forall^*$ sentences, one for each sort, and then use decision procedures for the respective theories to decide satisfiability of the decomposed (single sorted) sentences. Since such a decomposition can result into $\exists^*\forall^*$ sentences over the foreground sort, we must ensure that there is indeed a decision procedure to achieve this. For this purpose, let us define T_{σ_0} be the empty theory (that is $\mathcal{A}_{\sigma} = \emptyset$). Checking satisfiability of $\exists^*\forall^*$ sentences over T_{σ_0} is decidable. Also, satisfiability is preserved in the presence of T_{σ_0} in the following sense.

► **Lemma 7.** $\phi^{\text{Step-3}}$ is $\bigcup_{i=1}^k T_{\sigma_i}$ -satisfiable iff $\phi^{\text{Step-3}}$ is $\bigcup_{i=0}^k T_{\sigma_i}$ -satisfiable.

We first transform the quantifier free part $\psi^{\text{Step-3}}$ of $\phi^{\text{Step-3}}$ into an equivalent CNF formula ψ_{CNF} . Let ϕ_{CNF} be obtained by replacing $\psi^{\text{Step-3}}$ by ψ_{CNF} . Let $\phi_{\text{CNF}} \equiv \exists^*\forall^*\psi_{\text{CNF}}$, where $\psi_{\text{CNF}} \equiv \bigwedge_{i=1}^r \psi_i$ and each ψ_i is a disjunction of atoms. Since ϕ_{CNF} is a first order formula over a pure signature, all atoms are either of the form $R(\dots)$ or $t = t'$ (with possibly a leading negation). Now, equality atoms are restricted to terms of the same sort. Also since Σ is pure, the argument terms of all relation applications have the same sort. This means, for every atom α , there is a unique associated sort $\sigma \in \mathcal{S}$, which we will denote by $\text{sort}(\alpha)$.

For a clause ψ_i in ψ_{CNF} , let $\text{atoms}(\psi_i)$ be the set of atoms in ψ_i . Let $\text{atoms}^\sigma(\psi_i) = \{\alpha \in \text{atoms}(\psi_i) \mid \text{sort}(\alpha) = \sigma\}$, and let $\psi_i^\sigma \equiv \bigvee_{\alpha \in \text{atoms}^\sigma(\psi_i)} \alpha$. Then, we have the identity $\psi_{\text{CNF}} \equiv \bigwedge_{j=1}^r \bigvee_{\sigma \in \mathcal{S}} \psi_j^\sigma$. We now state our decomposition lemma.

► **Lemma 8.** ϕ_{CNF} is $\bigcup_{i=0}^k T_{\sigma_i}$ -satisfiable iff there is a mapping $L : \{1, \dots, r\} \rightarrow \mathcal{S}$ such that for each $0 \leq i \leq k$, the formula $\phi_i^L \equiv (\exists \mathbf{x}_i : \sigma_i)(\forall \mathbf{y}_i : \sigma_i) \bigwedge_{j \in L^{-1}(\sigma_i)} \psi_j^{\sigma_i}$ is T_{σ_i} -satisfiable.

Proof (Sketch). We present the more interesting direction here. Let ϕ_{Skolem} be an equisatisfiable Skolem norm form of ϕ_{CNF} . That is, $\phi_{\text{Skolem}} = \forall^*\psi_{\text{Skolem}}$, where ψ_{Skolem} is obtained from ψ_{CNF} by replacing all existential variables $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_k$ by *Skolem* constants. We will use the same notation ψ_i for the i^{th} clause of ψ_{Skolem} . Then, consider a structure \mathcal{M} such that $\mathcal{M} \models \bigcup_{i=0}^k T_{\sigma_i}$ and $\mathcal{M} \models \phi_{\text{Skolem}}$. Now, suppose, on the contrary, that there is a clause ψ_j such that for every sort σ_i , we have $\mathcal{M} \not\models \forall(\mathbf{y}_i : \sigma_i)\psi_j$. This means, for every sort σ_i , there is an interpretation \mathcal{I}_i (that extends \mathcal{I} with valuations of \mathbf{y}_i), such that either \mathcal{I}_i leads to falsity of T_{σ_i} or the clause ψ_j . Let $c_1^{\sigma_i}, c_2^{\sigma_i}, \dots, c_{|\mathbf{y}_i|}^{\sigma_i}$ be the values assigned to the universal variables \mathbf{y}_i in \mathcal{I}_i . Then, construct an interpretation \mathcal{I}' by extending \mathcal{I} with the variables \mathbf{y}_i interpreted with c^{σ_i} 's. This interpretation \mathcal{I}' can be shown to either violate one of the theory axioms or the formula ψ_j . In either case, we have a contradiction. ◀

The *contract* L above identifies, for each clause ψ_j , one sort σ_i such that the restriction $\psi_j^{\sigma_i}$ of ψ_j to σ_i can be set to true. Thus, in order to decide satisfiability of ϕ_{CNF} , a straightforward decision procedure involves enumerating all contracts, $L \in [\{1, \dots, r\} \rightarrow \mathcal{S}]$. For each contract L and for each sort σ_i , we construct the sentence ϕ_i^L , and make a black-box call to the $\exists^*\forall^*$ theory solver for T_{σ_i} . If there is a contract L for which each of these calls return “SATISFIABLE”, then ϕ_{CNF} (and thus, the original formula ϕ) is satisfiable. Otherwise, ϕ is unsatisfiable.

5 Undecidability Results

The logic that we have defined was carefully chosen to avoid undecidability of the satisfiability problem. We now show that natural generalizations or removal of restrictions in our logic renders the satisfiability problem undecidable. We believe our results are hence not simple to generalize any further.

One restriction that we have is that the functions that are existentially quantified cannot have σ_0 as their range sort. A related restriction is that the *universal* quantification block quantifies all uninterpreted function symbols, as otherwise they must be existentially quantified on the outside block.

Let us now consider the fragment of logic where formulas are of the form $(\exists \mathbf{x}_0)(\exists \mathbf{F})(\forall \mathbf{y}_0) \psi$ where in fact we do not even have any background theory. Since the formula is over a single sort, we have dropped the sort annotations on the variables. It is not hard to see that this logic is undecidable.

► **Theorem 9.** *Consider signature with a single sort σ_0 (and no background sorts). The satisfiability problem for sentences of the following form is undecidable.*

$$(\exists \mathbf{x}_0)(\exists \mathbf{F})(\forall \mathbf{y}_0) \psi$$

Proof (Sketch). We can show this as a mild modification of standard proofs of the undecidability of first-order logic. We can existentially quantify over a variable *Zero* and a function *succ*, demand that for any element y , *succ*(y) is not *Zero*, and for every y, y' , if *succ*(y) = *succ*(y'), then $y = y'$. This establishes an infinite model with distinct elements *succ* ^{n} (*Zero*), for every $n \geq 0$. We can then proceed to encode the problem of *non-halting* of a 2-counter machine using a relation $R(t, q, c_1, c_2)$, which stands for the 2CM is in state q at time t with counters c_1 and c_2 , respectively. It is easy to see that all this can be done using only universal quantification (the relation R can be modeled as a function easily). ◀

The theorem above has a simple proof, but the theorem is not new; in fact, even more restrictive logics are known to be undecidable (see [8]).

Another important restriction that we have is that the foreground sort and the various background sorts are *pariwise disjoint*. This requirement is also not negotiable if decidability is desired, as it is easy to show the following result. Once again we have dropped sort annotations, since we only have a single sort.

► **Theorem 10.** *Consider a signature with a single sort σ_1 and let T_{σ_1} be the theory of Presburger arithmetic. The satisfiability problem is decidable for sentences of the form*

$$(\exists \mathbf{x}_1)(\exists \mathbf{R})(\forall \mathbf{y}_1) \psi$$

Proof (Sketch). We can use a similar proof as the theorem above, except now that we use the successor function available in Presburger arithmetic. We can again reduce non-halting of Turing machines (or 2-counter machines) to satisfiability of such formulas. ◀

Stepping further back, there are very few subclasses of first-order logic with equality that have a decidable satisfiability problem, and the only standard class that admits $\exists^* \forall^*$ prefixes is the Bernays-Schönfinkel-Ramsey class (see [5]). Our results can be seen as an extension of this class with background theories, where the background theories admit locally a decidable satisfiability problem for the $\exists^* \forall^*$ fragment.

6 Applications to Synthesis

6.1 Synthesis: Validity or Satisfiability?

Though we argued in Section 2 that synthesis problems can be modeled using satisfiability of EQSMT sentences, there is one subtlety that we would like to highlight. In synthesis problems, we are asked to find an expression such that the expression satisfies a specification expressed as a formula in some logic. Assuming the specification is modeled as a universally quantified formula over background theories, we would like to know if $\forall y\varphi(e, y)$ holds for the synthesized expression e . However, in a logical setting, we have to qualify what “holds” means; the most natural way of phrasing this is that $\forall y\varphi(e, y)$ is valid over the underlying background theories, i.e., holds in all models that satisfy the background theories. However, the existential block that models the existence of an expression is clearly best seen as a satisfiability problem, as it asks whether there is *some foreground model* that captures an expression. Requiring that it holds in *all* foreground models (including those that might have only one element) would be unreasonable.

To summarize, the synthesis problem is most naturally modeled as a logical problem where we ask whether there is *some* foreground model (capturing a program expression) such that *all* background models, that satisfy their respective background theories, also satisfy the quantifier free formula expressing that the synthesized expression satisfies the specification. This is, strictly speaking, neither a satisfiability problem nor a validity problem!

We resolve this by considering only *complete and consistent* background theories. Hence validity of a formula under a background theory T is *equivalent* to T -satisfiability. Consequently, synthesis problems using such theories can be seen as asking whether there is a foreground universe (modeling the expression to be synthesized) *and* some background models where the specification holds for the expression. We can hence model synthesis purely as a satisfiability problem of EQSMT, as described in Section 2.

Many of the background theories used in verification/synthesis and SMT solvers are complete theories (like Presburger arithmetic, FOL over reals, etc.). One incomplete theory often used in verification is the theory of *uninterpreted functions*. However, in this case, notice that since the functions over this sort are uninterpreted, validity of formulas can be modeled using a *universal quantification* over functions, which is supported in EQSMT! The only other adjustment is to ensure that this background theory has only infinite models (we can choose this background theory to be the theory of $(\mathbb{N}, =)$, which has a decidable satisfiability problem). Various scenarios such as modeling pointers in heaps, arrays, etc., can be naturally formulated using uninterpreted functions over this domain.

The second issue in modeling synthesis problems as satisfiability problems for EQSMT is that in synthesis, we need to construct the expression, rather than just know one exists. It is easy to see that if the individual background theory solvers support finding concrete values for the existentially quantified variables, then we can pull back these values across our reductions to give the values of the existentially quantified first-order variables (over all sorts), the existentially quantified function variables as well as the existentially quantified relation variables, from which the expression to be synthesized can be constructed.

6.2 Evaluation

We illustrate the applicability of our result for solving synthesis problems.

Synthesis of recursive programs involving lists. We model the problem of synthesizing *recursive* programs with lists, that will meet a pre/post contract C assuming that recursive

calls on smaller data-structures satisfy the same contract C . Though the programs we seek are recursive, we can model certain classes of programs using straight-line programs.

To see this, let us take the example of synthesizing a program that finds a particular key in a linked list (`list-find`). We can instead ask whether there is a straight-line program which takes an additional input which models the return value of a possible recursive call made on the tail of the list. The straight-line program must then work on the head of the list and this additional input (which is assumed to satisfy the contract C) to produce an output that meets the same contract C .

For this problem, we modeled the program to be synthesized using existential quantification (over a grammar that generates bounded length programs) as described in Section 2. The pointer `next` and recursive data structures `list`, `lseg` in the verification condition were modeled using *universal quantification* over function variables and relation variables, respectively. Moreover, in order to have a tractable verification condition, we used the technique of *natural proofs* [20, 25, 28] that soundly formulates the condition in a decidable theory. We used `z3` [12] to ackermanize the universally quantified functions/relations (`lseg`, `list` and `next`). We encoded the resulting formula as a synthesis problem in the SYGUS format [4] and used an off-the-shelf enumerative counter-example guided synthesis (CEGIS) solver. A program was synthesized within 1s, which was manually verified to be correct.

We also encoded other problems involving lists : `list-length` (calculating the length of a list), `list-sum` (computing sum of the keys in a list), `list-sorted` (checking if the sequence of keys in the list is sorted) and `list-count-occurrence` (counting the number of occurrences of a key in the list), using a CEGIS solver, and report the running times and the number of programs explored in Table 1.

We are convinced that EQSMT can handle recursive program synthesis (of bounded size) against separation logics specifications expressed using natural proofs (as in [25]).

Synthesis of straight-line programs equivalent to given recursive programs. In the second class of examples, we turn to synthesizing straight-line programs given a recursive function as their specification. For example, consider Knuth’s generalization of the recursive McCarthy 91 function:

$$M(n) = \begin{cases} n - b & \text{if } n > a \\ M^c(n + d) & \text{otherwise} \end{cases}$$

for every integer n , and where $(c - 1)b < d$. For the usual McCarthy function, we have $a = 100$, $b = 10$, $c = 2$, and $d = 11$.

Consider the problem of synthesizing an equivalent recursion-free expression. The programs we consider may have if-then-else statements of nesting depth 2, with conditionals over linear expressions having unbounded constants. Existential quantification over the background arithmetic sort allowed us to model synthesizing these unbounded constants. Our specification demanded that the value of the expression for n satisfy the recursive equations given above.

We modeled the foreground sort *inside* arithmetic, and converted our synthesis problem to a first-order $\exists^*\forall^*$ sentence over Presburger arithmetic and Booleans. We experimented with several values for a, b, c, d (with $(c - 1)b < d$), and interestingly, solutions were synthesized only when $(d - (c - 1)b) = 1$. Given Knuth’s result that a closed form expression involves taking remainder modulo this expression (and since we did not have the modulo operation in our syntax), it turns out that simple expressions do not exist otherwise. Also, whenever the solution was found, it matched the recursion-free expression given by Knuth (see Theorem 1

■ **Table 1** Synthesis of list programs and recursive programs.

Program	# Programs Explored in SYGUS	Time(s)
list-find	~5k	0.5
list-length	~40k	5
list-sum	~160k	15
list-sorted	~206k	45
list-count-occurrence	~1.3 million	134
Knuth : ($a = 100, b = 10, c = 2, d = 11$)	-	2
Knuth : ($a = 15, b = 30, c = 3, d = 61$)	-	6
Knuth : ($a = 3, b = 20, c = 4, d = 62$)	-	27
Knuth : ($a = 9, b = 11, c = 5, d = 45$)	-	49
Knuth : ($a = 99, b = 10, c = 6, d = 51$)	-	224
Takeuchi	-	100

in [19]). In Table 1, we provide the running times of our implementation on various parameters. We also compared our implementation with the popular synthesis tool SKETCH [33] on these examples. For the purpose of comparison, we used the same template for both SKETCH and our implementation. Further, since SKETCH does not allow encoding integers with unbounded size (unlike our encoding in integer arithmetic), we represented these constants, to be synthesized, using bitvectors of size 8. SKETCH does not return an answer within the set time-limit of 10 minutes for most of these programs.

We also modeled the Tak function (by Takeuchi) given by the specification below.

$$t(x, y, z) = \begin{cases} y & \text{if } x \leq y \\ t(t(x - 1, y, z), t(y - 1, z, x), t(z - 1, x, y)) & \text{otherwise} \end{cases}$$

Our implementation synthesized the program $t(x, y, z) = \text{ite}(x \leq y, y, \text{ite}(y \leq z, z, x))$ in about 100s.

7 Related Work

There are several logics known in the literature that can express synthesis problems and are decidable. The foremost example is the monadic second-order theory over trees, which can express Church’s synthesis problem [10] and other reactive synthesis problems over *finite data domains*, and its decidability (Rabin’s theorem [30]) is one of the most celebrated theorems in logic that is applicable to computer science. Reactive synthesis has been well studied and applied in computer science (see, for example, [7]). The work reported in [21] is a tad closer to program synthesis as done today, as it synthesizes syntactically restricted programs with recursion that work on finite domains.

Caulfield et al [11] have considered the decidability of *syntax-guided synthesis* (SyGuS) problems, where the synthesized expressions are constrained to belong to a grammar (with operators that have the usual semantics axiomatized by a standard theory such as arithmetic) that satisfy a universally quantified constraint. They show that the problem is undecidable in many cases, but identify a class that asks for expressions satisfying a regular grammar with uninterpreted function theory constraints to be decidable.

The $\exists^*\forall^*$ fragment of pure predicate logic (without function symbols) was shown to be decidable by Bernays and Schönfinkel (without equality) and by Ramsey (with equality) [5], and is often called Effectively Propositional Reasoning (EPR) class. It is one of the few

fragments of first-order logic known to be decidable. The EPR class has been used in program verification [16, 24], and efficient SMT solvers supporting EPR have been developed [26].

The work by [1] extends EPR to stratified typed logics, which has some similarity with our restriction that the universes of the foreground and background be disjoint. However, the logic therein does not allow background SMT theories unlike ours and restricts the communication between universally and existentially quantified variables via equality between existential variables and terms with universally quantified variables as arguments. In [15], EPR with simple linear arithmetic (without addition) is shown to be decidable.

Theory extensions [32] and model theoretic and syntactic restrictions theorem [31] have been explored to devise decidable fragment for quantified fragments of first order logic. Here, reasoning in *local* theory extensions of a base theory can be reduced to the reasoning in the base theory (possibly with an additional quantification). Combination of theories which are extensions of a common base theory can similarly be handled by reducing the reasoning to a decidable base theory. Similar ideas have been employed in the context of combinations of linear arithmetic and the theory of uninterpreted functions with applications to construct interpolants [18] and invariants [6] for program verification. EQSMT does not require the background theories to be extensions of a common base theory.

Verification of programs with arrays and heaps can be modeled using second order quantification over the arrays/heaps and quantifier alternation over the elements of the array/heaps which belong to the theory of Presburger arithmetic. While such a logic is, in general, undecidable, careful syntactic restrictions such as limiting quantifier alternation [9] and *flatness* restrictions [3]. We do not restrict the syntax of our formulae, but ensure decidability via careful sort restrictions. A recent paper [20] develops sound and complete reasoning for a so-called *safe* FO fragment of an uninterpreted combination of theories. However, the logic is undecidable, in general, and also does not support second-order quantification.

The SyGuS format has recently been proposed as a language to express *syntax guided* synthesis problems, and there have been several synthesis engines developed for various tracks of SyGuS [4]. However, the syntax typically allows unbounded programs, and hence the synthesis problem is not decidable. In [13], the candidate program components are “decorated” with annotations that represent transformers of the components in a sound abstract domain. This reduces the synthesis problem ($\exists^*\forall^*$) to the search for a proof ($\exists^*\exists^*$) in the abstract domain.

When expressing synthesis problems for programs that manipulate heaps, we rely on natural-proofs style sound abstraction of the verification conditions. Natural synthesis [29] extends this idea to an inductive synthesis procedure.

8 Conclusions and Future Work

The logic EQSMT defined herein is meant to be a decidable logic for communication between researchers modeling program synthesis problems and researchers developing efficient logic solvers. Such liaisons have been extremely fruitful in verification, where SMT solvers have served this purpose. We have shown the logic to be decidable and its efficacy in modeling synthesis problems. However, the decision procedure has several costs that should not be paid *up front* in any practical synthesis tool. Ways to curb such costs are known in the literature of building efficient synthesis tools. In particular, searching for foreground models is similar to EPR where efficient engines have been developed [26], and the search can also be guided by CEGIS-like approaches [4]. And the exponential blow-up caused by guessing

contracts between solvers (in Step 4 of our procedure) is similar to *arrangements* agreed upon by theories combined using the Nelson-Oppen method, again for which efficient solvers have been developed. Our hope is that researchers working on logic engines will engineer an efficient decision procedure for EQSMT that can solve synthesis problems.

References

- 1 Aharon Abadi, Alexander Rabinovich, and Mooly Sagiv. Decidable fragments of many-sorted logic. *Journal of Symbolic Computation*, 45(2):153–172, 2010.
- 2 Wilhelm Ackermann. *Solvable cases of the decision problem*. North-Holland Publishing Company Amsterdam, 1962.
- 3 Francesco Alberti, Silvio Ghilardi, and Natasha Sharygina. Decision procedures for flat array properties. *J. Autom. Reason.*, 54(4):327–352, 2015. doi:10.1007/s10817-015-9323-7.
- 4 Rajeev Alur, Rastislav Bodík, Eric Dallal, Dana Fisman, Pranav Garg, Garvit Juniwal, Hadas Kress-Gazit, P. Madhusudan, Milo M. K. Martin, Mukund Raghothaman, Shambwaditya Saha, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Dependable Software Systems Engineering*, pages 1–25. IOS Press, 2015. doi:10.3233/978-1-61499-495-4-1.
- 5 Paul Bernays and Moses Schönfinkel. Zum entscheidungsproblem der mathematischen logik. *Mathematische Annalen*, 1928.
- 6 Dirk Beyer, Thomas A. Henzinger, Rupak Majumdar, and Andrey Rybalchenko. Invariant synthesis for combined theories. In Byron Cook and Andreas Podelski, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 378–394, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- 7 Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. Interactive presentation: Automatic hardware synthesis from specifications: A case study. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE '07*, 2007.
- 8 Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem*. Springer Science & Business Media, 2001.
- 9 Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. What’s decidable about arrays? In E. Allen Emerson and Kedar S. Namjoshi, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 427–442, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- 10 J Richard Buchi and Lawrence H Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 1969.
- 11 Benjamin Caulfield, Markus N. Rabe, Sanjit A. Seshia, and Stavros Tripakis. What’s decidable about syntax-guided synthesis? *CoRR*, abs/1510.08393, 2015.
- 12 Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *TACAS*, 2008.
- 13 Adrià Gascón, Ashish Tiwari, Brent Carmer, and Umang Mathur. Look for the proof to find the program: Decorated-component-based program synthesis. In *Computer Aided Verification*, 2017.
- 14 Sumit Gulwani. Dimensions in program synthesis. In *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, PPDP ’10, pages 13–24, New York, NY, USA, 2010. ACM. doi:10.1145/1836089.1836091.
- 15 Matthias Horbach, Marco Voigt, and Christoph Weidenbach. On the combination of the Bernays–Schönfinkel–Ramsey fragment with simple linear integer arithmetic. In *Proceedings of the International Conference on Automated Deduction*, pages 202–219, 2017.
- 16 Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Aleksandar Nanevski, and Mooly Sagiv. Effectively-propositional reasoning about reachability in linked data structures. In *International Conference on Computer Aided Verification*, 2013.

- 17 Susmit Jha, Sumit Gulwani, Sanjit A Seshia, and Ashish Tiwari. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 215–224. ACM, 2010.
- 18 Deepak Kapur, Rupak Majumdar, and Calogero G. Zarba. Interpolation for data structures. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, pages 105–116, New York, NY, USA, 2006. ACM. doi:10.1145/1181775.1181789.
- 19 Donald E Knuth. Textbook examples of recursion. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, 1991.
- 20 Christof Löding, P. Madhusudan, and Lucas Peña. Foundations for natural proofs and quantifier instantiation. *Proc. ACM Program. Lang.*, 2(POPL):10:1–10:30, 2017. doi:10.1145/3158098.
- 21 Parthasarathy Madhusudan. Synthesizing Reactive Programs. In *Computer Science Logic (CSL'11) - 25th International Workshop/20th Annual Conference of the EACSL*, 2011.
- 22 Zohar Manna and John McCarthy. Properties of programs and partial function logic. Technical report, Stanford University Computer Science Department, 1969.
- 23 Greg Nelson and Derek C Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(2):245–257, 1979.
- 24 Oded Padon, Kenneth L McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. Ivy: safety verification by interactive generalization. *ACM SIGPLAN Notices*, 2016.
- 25 Edgar Pek, Xiaokang Qiu, and Parthasarathy Madhusudan. Natural proofs for data structure manipulation in c using separation logic. In *ACM SIGPLAN Notices*, 2014.
- 26 Ruzica Piskac, Leonardo de Moura, and Nikolaj Bjørner. Deciding effectively propositional logic with equality. Technical report, Technical Report MSR-TR-2008-181, Microsoft Research, 2008.
- 27 Amir Pnueli, Yoav Rodeh, Ofer Strichman, and Michael Siegel. The small model property: How small can it be? *Information and computation*, 2002.
- 28 Xiaokang Qiu, Pranav Garg, Andrei Ştefănescu, and Parthasarathy Madhusudan. Natural proofs for structure, data, and separation. *ACM SIGPLAN Notices*, 2013.
- 29 Xiaokang Qiu and Armando Solar-Lezama. Natural synthesis of provably-correct data-structure manipulations. *Proc. ACM Program. Lang.*, 1(OOPSLA):65:1–65:28, oct 2017. doi:10.1145/3133889.
- 30 Michael O Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 1969.
- 31 Viorica Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In Robert Nieuwenhuis, editor, *Automated Deduction – CADE-20*, pages 219–234, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- 32 Viorica Sofronie-Stokkermans. On combinations of local theory extensions. In *Programming Logics: Essays in Memory of Harald Ganzinger*, pages 392–413, 2013.
- 33 Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. *ACM SIGOPS Operating Systems Review*, 2006.

A Encoding M_{three} in EQSMT

We are interested in synthesizing a straight line program that implements the function M_{three} , and can be expressed as a `term` over the grammar in Figure 1a.

Let us see how to encode this synthesis problem in EQSMT. First, let us fix the maximum height of the term we are looking for, say to be 2. Then, the program we want to synthesize can be represented as a tree of height at most 2 such that every node in the tree can have

≤ 3 child nodes (because the maximum arity of any function in the above grammar is 3, corresponding to `ite`). A skeleton of such an expression tree is shown in Figure 1b. Every node in the tree is named according to its path from the root node.

The synthesis problem can then be encoded as the formula

$$\begin{aligned}
 \phi_{M_{\text{three}}} \equiv & (\exists n_0, n_{00}, n_{01}, \dots, n_{022} : \sigma_0) \left(\underbrace{\exists \text{Left, Mid, Right} : \sigma_0, \sigma_0}_{\text{Existentially quantified relations}} \right) \\
 & (\exists \text{ADD, ITE, LTZero, EQZero, GTZero, INPUT, } C_1, C_2, C_3 : \sigma_{\text{label}}) \\
 & (\exists c_1, c_2, c_3 : \mathbb{N}) \left(\underbrace{\exists f_{\text{label}} : \sigma_0, \sigma_{\text{label}}}_{\text{Existentially quantified functions}} \right) \\
 & \varphi_{\text{well-formed}} \\
 & \wedge (\forall x : \mathbb{N}) \left(\underbrace{\forall g_{\text{val}}^0, g_{\text{val}}^1, g_{\text{val}}^2, g_{\text{val}}^3 : \sigma_0, \mathbb{N}}_{\text{Universally quantified functions}} \right) (\varphi_{\text{semantics}} \implies \varphi_{\text{spec}}) \quad (3)
 \end{aligned}$$

Here, the nodes are elements of the foreground sort σ_0 . The binary relations `Left`, `Mid`, `Right` over the foreground sort will be used to assert that a node n is the left, middle, right child respectively of node $n' : \text{Left}(n', n), \text{Mid}(n', n), \text{Right}(n', n)$. The operators or *labels* for nodes belong to the background sort σ_{label} , and can be one of `ADD` (`+`), `ITE` (`ite`), `LTZero` (`< 0`), `GTZero` (`> 0`), (`EQZero` (`= 0`)), `INPUT` (denoting the input to our program), or constants C_1, C_2, C_3 (for which we will synthesize natural constants c_1, c_2, c_3 in the (infinite) background sort \mathbb{N}). The function f_{label} assigns a label to every node in the program, and the formula $\varphi_{\text{well-formed}}$ asserts some sanity conditions:

$$\begin{aligned}
 \varphi_{\text{well-formed}} \equiv & \bigwedge_{\rho \neq \rho'} n_\rho \neq n_{\rho'} \wedge \text{Left}(n_0, n_{00}) \wedge \bigwedge_{\rho \neq 00} \neg(\text{Left}(n_0, n_\rho)) \wedge \dots \\
 & \wedge \neg(\text{ADD} = \text{ITE}) \wedge \neg(\text{ADD} = \text{LTZero}) \wedge \dots \wedge \neg(C_1 = C_3) \wedge \neg(C_2 = C_3) \\
 & \wedge \bigwedge_{\rho} (f_{\text{label}}(n_\rho) = \text{ADD}) \vee (f_{\text{label}}(n_\rho) = \text{ITE}) \vee \dots \vee (f_{\text{label}}(n_\rho) = C_3) \quad (4)
 \end{aligned}$$

The formula $\varphi_{\text{semantics}}$ asserts that the “meaning” of the program can be inferred from the meaning of the components of the program. The functions $g_{\text{val}}^0, g_{\text{val}}^1, g_{\text{val}}^2, g_{\text{val}}^3$, will assign value to nodes from \mathbb{N} , for this purpose :

$$\varphi_{\text{semantics}} \equiv \varphi_{\text{ADD}} \wedge \varphi_{\text{ITE}} \wedge \varphi_{\text{LTZero}} \wedge \varphi_{\text{EQZero}} \wedge \varphi_{\text{GTZero}} \wedge \varphi_{\text{INPUT}} \wedge \varphi_{C_1} \wedge \varphi_{C_2} \wedge \varphi_{C_3} \quad (5)$$

where each of the formulae $\varphi_{\text{ADD}}, \dots, \varphi_{C_3}$ specify the semantics of each node when labeled with these operations:

$$\begin{aligned}
 \varphi_{\text{ADD}} \equiv & \bigwedge_{\rho, \rho_1, \rho_2} (f_{\text{label}}(n_\rho) = \text{ADD} \wedge \text{Left}(n_\rho, n_{\rho_1}) \wedge \text{Mid}(n_\rho, n_{\rho_2})) \\
 & \implies \bigwedge_{i=0,1,2,3} g_{\text{val}}^i(n_\rho) = g_{\text{val}}^i(n_{\rho_1}) + g_{\text{val}}^i(n_{\rho_2}) \quad (6)
 \end{aligned}$$

$$\begin{aligned}
 \varphi_{\text{ITE}} \equiv & \bigwedge_{\rho, \rho_1, \rho_2, \rho_3} \left[f_{\text{label}}(n_\rho) = \text{ITE} \wedge \text{Left}(n_\rho, n_{\rho_1}) \wedge \text{Mid}(n_\rho, n_{\rho_2}) \wedge \text{Right}(n_\rho, n_{\rho_3}) \right. \\
 & \implies \bigwedge_{i=0,1,2,3} (g_{\text{val}}^i(n_{\rho_1}) = 1 \implies g_{\text{val}}^i(n_\rho) = g_{\text{val}}^i(n_{\rho_2}) \\
 & \quad \left. \wedge g_{\text{val}}^i(n_{\rho_1}) = 0 \implies g_{\text{val}}^i(n_\rho) = g_{\text{val}}^i(n_{\rho_3})) \right] \quad (7)
 \end{aligned}$$

$$\begin{aligned}
\varphi_{\text{LTZero}} &\equiv \bigwedge_{\rho, \rho_1} \left[f_{\text{label}}(n_\rho) = \text{LTZero} \wedge \text{Left}(n_\rho, n_{\rho_1}) \right. \\
&\implies \bigwedge_{i=0,1,2,3} \left(g_{\text{val}}^i(n_{\rho_1}) < 0 \implies g_{\text{val}}^i(n_\rho) = 1 \right. \\
&\quad \left. \left. \wedge g_{\text{val}}^i(n_{\rho_1}) \geq 0 \implies g_{\text{val}}^i(n_\rho) = 0 \right) \right] \tag{8}
\end{aligned}$$

$$\begin{aligned}
\varphi_{\text{EQZero}} &\equiv \bigwedge_{\rho, \rho_1} \left[f_{\text{label}}(n_\rho) = \text{LTZero} \wedge \text{Left}(n_\rho, n_{\rho_1}) \right. \\
&\implies \bigwedge_{i=0,1,2,3} \left(g_{\text{val}}^i(n_{\rho_1}) = 0 \implies g_{\text{val}}^i(n_\rho) = 1 \right. \\
&\quad \left. \left. \wedge g_{\text{val}}^i(n_{\rho_1}) \neq 0 \implies g_{\text{val}}^i(n_\rho) = 0 \right) \right] \tag{9}
\end{aligned}$$

$$\begin{aligned}
\varphi_{\text{GTZero}} &\equiv \bigwedge_{\rho, \rho_1} \left[f_{\text{label}}(n_\rho) = \text{LTZero} \wedge \text{Left}(n_\rho, n_{\rho_1}) \right. \\
&\implies \bigwedge_{i=0,1,2,3} \left(g_{\text{val}}^i(n_{\rho_1}) > 0 \implies g_{\text{val}}^i(n_\rho) = 1 \right. \\
&\quad \left. \left. \wedge g_{\text{val}}^i(n_{\rho_1}) \leq 0 \implies g_{\text{val}}^i(n_\rho) = 0 \right) \right] \tag{10}
\end{aligned}$$

The formula φ_{INPUT} states that for a node labeled **INPUT**, the value of that node is the input to M_{three} . Hence, such a node n_ρ evaluates to x , $x+61$, $g_{\text{val}}^1(n_0)$ and $g_{\text{val}}^2(n_0)$ respectively under g_{val}^0 , g_{val}^1 , g_{val}^2 and g_{val}^3 :

$$\begin{aligned}
\varphi_{\text{INPUT}} &\equiv \bigwedge_{\rho} \left[f_{\text{label}}(n_\rho) = \text{INPUT} \implies \right. \\
&\quad \left. \begin{aligned}
&g_{\text{val}}^0(n_\rho) = x \\
&\wedge g_{\text{val}}^1(n_\rho) = x + 61 \\
&\wedge g_{\text{val}}^2(n_\rho) = g_{\text{val}}^1(n_0) \\
&\wedge g_{\text{val}}^3(n_\rho) = g_{\text{val}}^2(n_0)
\end{aligned} \right] \tag{11}
\end{aligned}$$

Finally we have the semantics of constant labels:

$$\varphi_{C_1} \equiv \bigwedge_{\rho} \left[f_{\text{label}}(n_\rho) = C_1 \implies \bigwedge_{i=0,1,2,3} g_{\text{val}}^i(n_\rho) = c_1 \right] \tag{12}$$

The formulae φ_{C_2} and φ_{C_3} are similar and thus skipped.

Last, the formula φ_{spec} expresses the specification of the program as in Equation (2).

$$\begin{aligned}
\varphi_{\text{spec}} &\equiv (x > 13 \implies g_{\text{val}}^0(n_0) = x - 30) \\
&\quad \wedge (x \leq 13 \implies g_{\text{val}}^0(n_0) = g_{\text{val}}^3(n_0)) \tag{13}
\end{aligned}$$

Quantitative Foundations for Resource Theories

Dan Marsden

University of Oxford, Oxford, United Kingdom
daniel.marsden@cs.ox.ac.uk

Maaïke Zwart

University of Oxford, Oxford, United Kingdom
maaike.zwart@cs.ox.ac.uk

Abstract

Considering resource usage is a powerful insight in the analysis of many phenomena in the sciences. Much of the current research on these resource theories focuses on the analysis of specific resources such as quantum entanglement, purity, randomness or asymmetry. However, the mathematical foundations of resource theories are at a much earlier stage, and there has been no satisfactory account of quantitative aspects such as costs, rates or probabilities.

We present a categorical foundation for *quantitative* resource theories, derived from enriched category theory. Our approach is compositional, with rich algebraic structure facilitating calculations. The resulting theory is parameterized, both in the quantities under consideration, for example costs or probabilities, and in the structural features of the resources such as whether they can be freely copied or deleted. We also achieve a clear separation of concerns between the resource conversions that are freely available, and the costly resources that are typically the object of study. By using an abstract categorical approach, our framework is naturally open to extension. We provide many examples throughout, emphasising the resource theoretic intuitions for each of the mathematical objects under consideration.

2012 ACM Subject Classification Theory of computation → Logic, Theory of computation → Categorical semantics

Keywords and phrases Resource Theory, Enriched Category, Profunctor, Monad, Combinatorial Species, Multicategory, Operad, Bimodule

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.32

Acknowledgements This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2015-0-00565, Development of Vulnerability Discovery Technologies for IoT Software Security). We would like to thank Bob Coecke, Tobias Fritz and Rob Spekkens for enlightening discussions about resource theories. We would also like to thank the anonymous referees for their feedback.

1 Introduction

The importance of analyzing phenomena from the perspective of resource conversions and consumption is an insight that pervades many disciplines. Logicians have long understood the significance of this point of view. For example, strong resource based intuitions underlie linear logic [14] and the resource and differential lambda calculi [2, 6].

In the natural sciences, many aspects of physics are now investigated using what are loosely termed *resource theories*. There are many different resource theories, for example, for quantum information alone, researchers have considered a multitude of possibilities, including



© Dan Marsden and Maaïke Zwart;
licensed under Creative Commons License CC-BY
27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 32; pp. 32:1–32:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

asymmetry [24], non-uniformity [15], athermalality [3] and superposition [29]. Much of the current work on resource theories focuses on specific situations. An exception is [4], where a pleasing categorical abstraction of resource theories is proposed.

In order to facilitate discussions, we describe a very simple culinary example, which hopefully does not require any domain specific expertise. Consider the “recipe”:

$$\text{egg} + \text{egg} + \text{cream} + \text{sugar} \rightarrow \text{custard} \quad (1)$$

We read this as saying if we take two eggs, a standard unit of both cream and sugar, we can produce one unit of custard. Obviously we would like to combine such conversions, for example as a second step, we may want to combine our custard with an apple pie to form a pleasant dessert. Therefore a model of resource conversions should be *compositional*.

The recipe (1) already encodes some simple quantitative data about *resources* - two eggs are required as an input. In this paper we are interested not in quantifying the resources themselves, but in adding the ability to provide quantitative data about the *conversions* that can take place. For example:

- There may be a *cost* to producing custard, in elapsed time, energy consumed, or simply in paying a chef to do the cooking.
- Producing custard is unfortunately probabilistic, the custard may split or get burnt during cooking. We may therefore wish to quantify the *success probability* of a conversion taking place.
- If we are running a restaurant we may be interested in the *rate* of production so that we can keep our customers happy.

One can imagine quantifying similar features for chemical and biological reactions, economic behaviour, network communications, physical interactions and so on. Refining these ideas, resource theories typically separate resources into “free” resources conversions that are readily available, and “costly” processes that are often the focus of attention. An abstract model of resources should provide a clear separation of concerns between these two classes of resources.

Although some specific quantitative elements of resource theories are touched upon towards the end of [4], the approach is ad-hoc and no general purpose account of quantitative aspects is provided. They also fix the structural aspects of resources once and for all, rather than identifying this as a parameter of their theory. We provide a more general framework that allows variation in both the quantitative and structural aspects of resource theories.

We propose a foundation for *quantitative* resource theories, in which quantitative data can be attached to resource conversions. Our approach is based on two central ideas:

1. Exploiting enriched category theory allows us to incorporate quantitative data in a categorical framework. This is a classical idea, originating in Lawvere’s seminal paper on generalized metric spaces [22]. By varying the base of enrichment, we can then adjust our quantities to the needs of a given application.
2. More recent theory on generalized algebraic structures [17, 23, 9] allows us to incorporate structural aspects of resources, such as whether they can always be deleted, or copied, or if the order in which they are provided matters. These models of generalized algebraic structures are closely related to relational models of linear logic, and many of the structures we exploit can intuitively be viewed as *generalized binary relations*.

By successfully combining these two elements, and systematically applying categorical methods, a satisfactory mathematical theory emerges. Pleasingly, many meaningful resource theoretic features emerge naturally as standard categorical structures such as monads, profunctors and bimodules.

Providing a general purpose foundation for quantitative resource theories opens up the opportunity for the unification and transfer of ideas between many fields of mathematics and the sciences. It also allows us to analyze such models in the abstract, letting us compare theories and understand their essential features, uncluttered by application specific details. At this level of abstraction, unexpected connections become apparent, for example there is clearly a link with Pavlovic’s quantitative formal concept analysis [27] that should be explored.

1.1 Features

We highlight the following key features of our framework:

- **Modularity:** Our approach is parametric in two key directions. Firstly, how resource conversions are quantified can be configured to suit application needs, for example probabilities, rates or costs. Secondly, we can choose the structural aspects of resources, does their order matter? Can they be copied or deleted?
- **Compositionality:** The ability to compose and combine resources is intrinsic to our categorical approach. As we develop the underlying mathematics a great deal of algebraic structure emerges. This structure enables a calculational approach to reasoning about resource theories.
- **Separation of concerns:** We provide a clear separation between the “free” resource conversions that are readily available to everybody, and the “costly” conversions that are typically the main object of study.
- **Extensibility:** A categorical framework is naturally open to further extensions. This is a necessary feature of any realistic approach to quantifying resources. Given the breadth of potential applications, it is unrealistic to expect to anticipate every possible model of resources, their composition and quantification.
- **Practicality:** Although we work with abstractions such as enriched categories, monads and bicategories, in the special cases we deal with they have simple concrete descriptions as special sorts of matrices. This means that calculations in particular instances of our framework should be straightforward, and will not require advanced mathematical techniques.

1.2 Contribution

We outline our contribution:

- We provide a consistent resource theoretic interpretation of all the mathematical structure under consideration, building upon classical ideas of Lawvere [22]. This begins with material that will be familiar to some in the community, as we introduce mathematical background in sections 2 and 3, and continues with the newer concepts in later sections.
- In section 4 we give concrete descriptions of a hierarchy of five different free constructions on quantale enriched categories, that can be used to model the structural aspects of resources.
- Also in section 4, we show that each of the monads corresponding to the hierarchy of free constructions distributes over the free cocompletion monad. This allows us to extend our notions of resource interaction with new structural features.
- In section 5 we demonstrate how the resulting comonads yield “thin” variations on the notion of multicategory or operad, suitable for quantitative reasoning.
- In section 6 we show bimodules are the correct mathematical framework for incorporating freely available conversions requiring multiple components.

- In section 7 we address practical methods for closing resource conversions under composition in various ways. We establish that these constructions are canonical, by showing that each of them yields a free internal monad in an appropriate bicategory.

2 Quantale Enriched Categories

This section sets up standard technical background and notation. Throughout the paper, we aim for a self contained account with respect to enriched category theory. We will assume some basic knowledge of category theory, at the level of categories, functors, natural transformations, and (co)monads and their (co)Kleisli categories. The ideas in this section are well known, and the basic resource theoretic interpretations will be familiar to some in the community.

Throughout the document, we will specialize definitions to our situation of interest, without spelling out the details in full generality, as this will often significantly reduce the complexity involved. This applies to notions such as enriched categories, free constructions, bimodules and internal monads that occur in later sections. Experts will be able to recover our definitions from the more abstract formulations.

2.1 Quantales

We will use quantales to describe the abstract mathematical structure needed to quantify the costs of resource conversions.

► **Definition 1** (Quantale). A **quantale** is a complete join semilattice with a monoid structure (\otimes, k) such that the following axioms hold ¹:

$$p \otimes \left(\bigvee_i q_i \right) = \bigvee_i p \otimes q_i \quad \text{and} \quad \left(\bigvee_i p_i \right) \otimes q = \bigvee_i p_i \otimes q$$

A **commutative quantale** is a quantale whose underlying monoid is commutative. All the quantales we consider in this paper will be commutative. Throughout, we shall use the symbol \mathbb{Q} to denote an arbitrary commutative quantale.

The structure of a commutative quantale has a clear resource theoretic interpretation, with two key components:

1. The monoid structure allows us to combine quantities across the different steps of a process or algorithm, for example costs, success probabilities or connection strengths.
2. The join semilattice structure is then an optimizer. Having calculated aggregate values for various candidate procedures to achieve a desired aim, we can then quantify the best value attainable. For example, this might be the cheapest price, highest success probability or best connection strength achievable.

We introduce four quantales that will be used repeatedly in examples throughout the paper.

► **Example 2.** The **Boolean quantale** \mathbb{B} has the two Boolean truth values as its underlying set, with logical disjunction and conjunction providing the join semilattice and monoid structure respectively.

¹ Sometimes the term *unital quantale* is used, but we will have no interest in the case without a unit.

► **Example 3.** The **interval quantale** \mathbb{I} has underlying set the closed real interval $[0, 1]$. The join semilattice structure is given by the usual supremum, and the binary monoid operation takes the minimum of two elements.

► **Example 4.** The **Lawvere quantale** \mathbb{L} has underlying set the extended positive reals $[0, \infty]$ with the join semilattice structure given by *infima* and the monoid structure given by addition of real numbers.

► **Example 5.** The **multiplicative quantale** \mathbb{M} has underlying set the closed real interval $[0, 1]$. The join semilattice is given by suprema, and the binary monoid is ordinary multiplication of real numbers.

Finally, we remark that there are many more examples of commutative quantales. In particular, every locale [18] is a commutative quantale, including all complete Boolean algebras, finite distributive lattices and complete chains.

From a categorical perspective, a commutative quantale is a (small, thin, skeletal) complete and cocomplete symmetric monoidal closed category. It is this structure that makes them very pleasant to work with in enriched category theory.

2.2 Quantale Enriched Category Theory

The use of enriched category theory will be an essential tool for this paper. The standard source for enriched category theory is [19], but as we suggested earlier, the general definitions simplify significantly in the quantale enriched case. This is because there are many axioms to enforce structure, such as composition being associative or functors preserving identities, that are phrased in terms of certain diagrams commuting. As quantales are thin categories, all these axioms become trivial. We therefore provide concrete descriptions of the various enriched mathematical objects that we use, specialized to the simpler quantale enriched setting. Via examples, we take the opportunity to introduce our resource theoretic perspective on each of the various notions.

All our quantale enriched categories will be small, that is, we will require that they have a *set* of objects.

► **Definition 6** (\mathbb{Q} -enriched Categories). A \mathbb{Q} -enriched category \mathcal{A} consists of:

- A set of **objects** $\mathbf{obj}\mathcal{A}$. We will typically denote these objects as a, b, c, \dots
- For each pair of objects, there is a **hom object** $\mathcal{A}(a, b) \in \mathbb{Q}$.

The hom objects are required to satisfy two axioms:

- The **identity axiom**, for all a :

$$k \leq \mathcal{A}(a, a)$$

- The **composition axiom**, for all a, b, c :

$$\mathcal{A}(b, c) \otimes \mathcal{A}(a, b) \leq \mathcal{A}(a, c)$$

Enrichment over each of our example quantales has a natural resource theoretic interpretation.

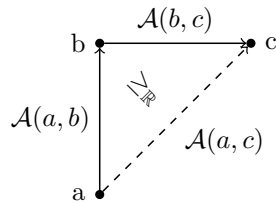
► **Example 7** (Boolean Quantale Enrichment). A \mathbb{B} -enriched category is the same thing as a preorder. We can interpret $a \leq b$ as meaning it is possible to convert resource a to b . The identity axiom corresponds to reflexivity, we can always convert a resource to itself. The composition axiom corresponds to transitivity, and captures the idea that if we can convert resource a to b and we can convert b to c , then we can combine these conversions to convert a to c .

► **Example 8** (Interval Quantale Enrichment). An \mathbb{I} -enriched category \mathcal{A} is a “fuzzy” generalization of a preorder. From a resource perspective, we interpret $\mathcal{A}(a, b)$ as a connection strength between a and b . Connection strengths are valued in a worst case manner, a composite connection is only as good as its weakest link. Then:

- The identity axiom tells us we can always connect any a to itself with maximum strength.
- The composition axiom tell us that if we can connect a to b and b to c , we should be able to connect a to c at least as strongly as going via the intermediate b .

► **Example 9** (Lawvere Quantale Enrichment). For an \mathbb{L} -enriched category \mathcal{A} , $\mathcal{A}(a, b)$ can be seen as the cost of converting a to b .

- The identity axiom tells us that we can freely convert a to itself. In Lawvere’s original metric space reading [22] the absence of the axiom $\mathcal{A}(a, b) = 0 \Rightarrow a = b$ is inconvenient. However, from a resource conversion perspective it is entirely natural that two distinct resources could be interconvertible.
- The composition axiom is a triangle inequality, saying that the cost of converting from a to c should be at least as cheap as converting via any intermediate resource b .



► **Example 10** (Multiplicative Quantale Enrichment). For an \mathbb{M} -enriched category \mathcal{A} , we interpret $\mathcal{A}(a, b)$ as the probability of successfully converting a to b . Conversion probabilities are assumed to be independent, so they multiply.

- The identity axiom tells us we can always convert a resource to itself with certainty.
- The composition axiom tells us that we can convert a to c with a success probability at least as high as that achievable by chaining two conversions via any intermediate resource b .

This concludes our examples for this section. It remains to define the enriched notions of \mathbb{Q} -functors and \mathbb{Q} -natural transformations in preparation for later sections.

► **Definition 11** (\mathbb{Q} -enriched Functor). Let \mathcal{A} and \mathcal{B} be \mathbb{Q} -enriched categories. A **\mathbb{Q} -enriched functor** F of type $\mathcal{A} \rightarrow \mathcal{B}$ consists of an object assignment function:

$$F : \text{obj } \mathcal{A} \rightarrow \text{obj } \mathcal{B}$$

such that:

$$\mathcal{A}(a, b) \leq \mathcal{B}(Fa, Fb)$$

Identity and composite functors are given in the obvious way, and the resulting structure yields a category $\mathbf{Cat}(\mathbb{Q})$ of \mathbb{Q} -categories and functors between them.

► **Definition 12** (\mathbb{Q} -enriched Natural Transformations). Let $F, G : \mathcal{A} \rightarrow \mathcal{B}$ be parallel \mathbb{Q} -enriched functors. The existence of a **\mathbb{Q} -enriched natural transformation** α of type $F \Rightarrow G$ simply states that the following inequalities hold for all objects of \mathcal{A} :

$$k \leq \mathcal{B}(Fa, Ga)$$

That is, there can be at most one \mathbb{Q} -natural transformation between two such functors.

We do not dwell on examples of functors and natural transformations now, as there will be many examples later in cases of particular importance.

3 Presheaves and Profunctors

We first introduce some constructions on quantale enriched categories.

► **Definition 13.** Let \mathcal{A} and \mathcal{B} be \mathbb{Q} -enriched categories.

- There is a **unit** \mathbb{Q} -category \mathcal{I} with a single object and the quantale unit as the unique hom object.
- The **tensor category** $\mathcal{A} \otimes \mathcal{B}$ has set of objects $\mathbf{obj} \mathcal{A} \times \mathbf{obj} \mathcal{B}$, and hom objects:

$$(\mathcal{A} \otimes \mathcal{B})((a, b), (a', b')) = \mathcal{A}(a, a') \otimes \mathcal{B}(b, b')$$

- The **opposite category** \mathcal{A}^{op} has the same objects as \mathcal{A} , and hom objects:

$$\mathcal{A}^{op}(a, a') = \mathcal{A}(a', a)$$

- A quantale \mathbb{Q} also carries a canonical structure as a \mathbb{Q} -category, with objects the elements of \mathbb{Q} , and hom objects:

$$\mathbb{Q}(q, q') = q \multimap q'$$

Where $q \multimap q'$ denotes the internal hom in \mathbb{Q} .

► **Definition 14 (Presheaf).** Let \mathcal{A} be a \mathbb{Q} -category.

- A **copresheaf** is a functor of type $\mathcal{A} \rightarrow \mathbb{Q}$. This is a function $F : \mathbf{obj} \mathcal{A} \rightarrow \mathbf{obj} \mathbb{Q}$ such that:

$$F(a) \otimes \mathcal{A}(a, b) \leq F(b)$$

- A **presheaf** is a functor of type $\mathcal{A}^{op} \rightarrow \mathbb{Q}$. This is a function $F : \mathbf{obj} \mathcal{A} \rightarrow \mathbf{obj} \mathbb{Q}$ such that:

$$\mathcal{A}(a, b) \otimes F(b) \leq F(a)$$

► **Definition 15 (Profunctor).** For a commutative quantale \mathbb{Q} , and \mathbb{Q} -enriched categories \mathcal{A} and \mathcal{B} , a **profunctor** from \mathcal{A} to \mathcal{B} is a functor of type:

$$\mathcal{A}^{op} \otimes \mathcal{B} \rightarrow \mathbb{Q}$$

Concretely, this is a function $R : \mathbf{obj} \mathcal{A} \times \mathbf{obj} \mathcal{B} \rightarrow \mathbf{obj} \mathbb{Q}$ such that:

$$\mathcal{A}(a', a) \otimes R(a, b) \otimes \mathcal{B}(b, b') \leq R(a', b')$$

We write $R : \mathcal{A} \multimap \mathcal{B}$ to indicate R is a profunctor from \mathcal{A} to \mathcal{B} .

A profunctor can be thought of as a categorical generalization of the notion of binary relation, taking truth values in the underlying quantale. They generalize both presheaves and copresheaves, as they are profunctors of type $\mathcal{A} \multimap \mathcal{I}$ and $\mathcal{I} \multimap \mathcal{A}$ respectively.

► **Example 16.** (Co)presheaves have natural resource theoretic interpretations. For example, if we consider \mathbb{L} -enrichment:

- A copresheaf on \mathcal{A} is a coherent set of costs for acquiring the resources in \mathcal{A} . The copresheaf condition:

$$F(a) + \mathcal{A}(a, b) \geq_{\mathbb{R}} F(b)$$

requires that it is always cheaper to buy a resource b directly, rather than purchase some other resource a and then pay $\mathcal{A}(a, b)$ to turn it into b .

32:8 Quantitative Foundations for Resource Theories

- A presheaf on \mathcal{A} is a coherent set of costs for disposing of the resource in \mathcal{A} . The presheaf condition:

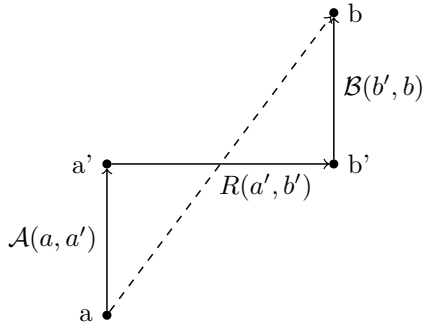
$$\mathcal{A}(a, b) + F(b) \geq_{\mathbb{R}} F(a)$$

requires that it is always cheaper to dispose of a resource a directly, rather than pay the cost $\mathcal{A}(a, b)$ to convert it to some b and then pay the cost to destroy b .

- **Example 17.** We consider profunctors from a resource perspective, using the multiplicative quantale. A profunctor $R : \mathcal{A} \leftrightarrow \mathcal{B}$ satisfies:

$$\mathcal{A}(a', a) \times R(a, b) \times \mathcal{B}(b, b') \leq_{\mathbb{R}} R(a', b')$$

If we interpret R as describing a probabilistic device for converting \mathcal{A} resources to \mathcal{B} resources, the profunctor axiom says that the device will convert a' to b' with a success probability higher than the product of the probabilities of converting a' to a in \mathcal{A} , and then using R to convert a to b , and then converting b to b' in \mathcal{B} , as shown below:



Notice the probabilities here describe the chances of success of a chosen conversion, rather than which conversion will take place, as might be seen in stochastic relations for example.

- **Remark (Separation of Concerns).** Profunctors are the first point at which we see that the enriched categorical framework provides a clear separation of concerns between free and costly resources. The domain and codomain model the resources freely available. The transition costs encoded by the profunctor then provide additional resources conversions, with the profunctor axiom requiring that these all these conversions are better than can be achieved by additionally exploiting free resources.

- **Definition 18.** Given profunctors $R : \mathcal{A} \leftrightarrow \mathcal{B}$ and $S : \mathcal{B} \leftrightarrow \mathcal{C}$, we can form their composite $S \circ R : \mathcal{A} \leftrightarrow \mathcal{C}$, defined pointwise as follows:

$$(S \circ R)(a, c) = \bigvee_b R(a, b) \otimes S(b, c)$$

This composition is associative, and has identity at \mathcal{A} given by:

$$1_{\mathcal{A}}(a, a') = \mathcal{A}(a, a')$$

Therefore \mathbb{Q} -profunctors form a category $\mathbf{Prof}(\mathbb{Q})$.

- **Example 19.** Continuing example 17, we consider the composition of two \mathbb{M} -profunctors, $R : \mathcal{A} \leftrightarrow \mathcal{B}$ and $S : \mathcal{B} \leftrightarrow \mathcal{C}$. Intuitively, the value:

$$(S \circ R)(a, c) = \sup_b \{R(a, b) \times S(b, c)\}$$

describes the best probability achievable for converting a to c via some intermediate b using the two probabilistic devices described by R and S .

► **Remark.** In general, composition of profunctors is defined using colimits in the enriching category. Therefore we can only expect associativity and unitality of composition to hold up to isomorphism, pointing us in the more complicated direction of bicategories. Fortunately, the only isomorphisms in a quantale are the identities, and so composition is defined “on the nose”, yielding a genuine category.

The tensor structure of definition 13 gives $\mathbf{Prof}(\mathbb{Q})$ the structure of a symmetric monoidal category. In fact it is a compact closed category [20], and so has a powerful graphical calculus that can be exploited in calculations.

As profunctors are a generalization of binary relations, and relations are closed under taking unions, we may expect similar structure of profunctors.

► **Definition 20.** A complete join semilattice enriched category is an ordinary category such that the hom sets are complete join semilattices, and the following axioms hold:

$$\left(\bigsqcup_i S_i \right) \circ R = \bigsqcup_i (S_i \circ R) \quad \text{and} \quad S \circ \left(\bigsqcup_i R_i \right) = \bigsqcup_i (S \circ R_i)$$

Complete join semilattice enrichment also implies that hom sets have a partial order \subseteq such that composition is monotone in both components.

The following then gives us a straightforward generalization of taking unions of ordinary binary relations.

► **Lemma 21.** For a commutative quantale \mathbb{Q} , the category $\mathbf{Prof}(\mathbb{Q})$ is complete join semilattice enriched with:

$$\left(\bigsqcup_i R_i \right) (a, b) = \bigvee_i R_i(a, b)$$

If we return to our resource theoretic perspective, $\bigsqcup_i R_i$ combines the best capabilities of a family of different resource conversion options. The induced order $R \subseteq S$ is equivalent to there being a \mathbb{Q} -natural transformation $R \Rightarrow S$. We require another specialized definition.

► **Definition 22 (Internal Monad).** An **internal monad** in a complete join semilattice enriched category is an endomorphism $R : \mathcal{A} \rightarrow \mathcal{A}$ such that both:

$$1_A \subseteq R \quad \text{and} \quad R \circ R \subseteq R$$

Internal monads are an important concept. From the point of view of resources, an internal monad captures closure under repeated application of the available conversions. We can think of an internal monad on \mathcal{A} as describing a “better” \mathbb{Q} -enriched category structure on the objects of \mathcal{A} .

► **Example 23 (Internal Monads as Better Structures).** An internal monad $R : \mathcal{A} \rightarrow \mathcal{A}$ in $\mathbf{Prof}(\mathbb{L})$ is a selection of resource conversion costs that is closed under composition. That is, the cost $R(a, a')$ will be cheaper than the cost of any iterated conversion:

$$a \rightarrow b_1 \rightarrow \dots \rightarrow b_n \rightarrow a'$$

Such a monad provides resource conversion costs that are closed under composition, and better than those of the underlying category \mathcal{A} .

Similarly, an internal monad $P : \mathcal{A} \rightarrow \mathcal{A}$ in $\mathbf{Prof}(\mathbb{B})$ is a preorder stronger than the original order on \mathcal{A} .

We shall encounter internal monads again in sections 5, 6 and 7 as we introduce richer structure to our resources.

4 A Hierarchy of Resource Structures

So far, we have considered only conversions between individual resources. In this section, we introduce additional structure that will allow us to consider conversions that require multiple inputs, such as the custard recipe of the introduction.

► **Definition 24.** A \mathbb{Q} -category \mathcal{A} is:

- **Strictly monoidal** if the objects carry a monoid structure \otimes, I such that:

$$\mathcal{A}(a_1, b_1) \otimes \mathcal{A}(a_2, b_2) \leq \mathcal{A}(a_1 \otimes a_2, b_1 \otimes b_2)$$

From here on, we will drop explicitly saying “strictly” and simply use the term monoidal \mathbb{Q} -category.

- **Symmetric monoidal** if it is monoidal and for all $a, b \in \mathcal{A}$:

$$k \leq \mathcal{A}(a \otimes b, b \otimes a)$$

- **Deleting** if it is symmetric monoidal, and for all $a \in \mathcal{A}$:

$$k \leq \mathcal{A}(a, I)$$

- **Copying** if it is symmetric monoidal, and for all $a \in \mathcal{A}$:

$$k \leq \mathcal{A}(a, a \otimes a)$$

- **Cartesian** if it is both copying and deleting.

A homomorphism of each of these special sorts of \mathbb{Q} -categories is a \mathbb{Q} -functor that is a monoid homomorphism with respect to the monoid structure on objects.

Each of these structures has a resource theoretic reading. A monoidal \mathbb{Q} -category allows us to combine ordered collections of resources. This setting is very restrictive, we are not necessarily able to even adjust the order of the resources provided. A symmetric monoidal \mathbb{Q} -category allows us to cheaply interchange the order of resources. If a \mathbb{Q} -category is deleting, we can also delete resources we do not need, and if it is copying, we can copy available resources, effectively making them reusable. This perspective will be most apparent in the forthcoming free constructions, in which the objects are lists of resources.

We also introduce some additional properties of quantales that we will require, using terminology paralleling that used for \mathbb{Q} -categories.

► **Definition 25.** We say that a quantale \mathbb{Q} is:

- **Deleting** if the monoid unit k is the top element.
- **Copying** if the for all $q \in \mathbb{Q}$, $q \leq q \otimes q$.
- **Cartesian** if it is both copying and deleting².

In this section we describe a hierarchy of free constructions on \mathbb{Q} -enriched categories. This family of constructions is reminiscent of the Boom type hierarchy [26] familiar to the functional programming community, in which varying the axioms required of a construction of a particular shape results in a family of different datatypes. In our case, the objects of each free construction will be lists of resources. The interesting structure is in the hom objects, which will encode the resource conversions we wish to provide as standard. We will therefore frequently need to work with finite lists.

² A commutative quantale is Cartesian if and only if it is a locale.

► **Definition 26** (List Notation). We will write $[a]$ for the singleton list. For a list of elements A , we will write A_i for the i^{th} element of the list and $\#A$ for the length of the list. We will also write $i : \#A$ to mean $1 \leq i \leq \#A$, and $\otimes_{i:\#A} \tau_i$ as shorthand for the iterated tensor product $\tau_1 \otimes \dots \otimes \tau_{\#A}$. We will also abuse notation, and identify $\#A$ with the set $\{1, \dots, \#A\}$.

► **Theorem 27.** For a commutative quantale \mathbb{Q} , \mathbb{Q} -category \mathcal{A} , and lists of \mathcal{A} -objects A, B , define:

$$\bigvee_{\psi:\#B \rightarrow \#A} \otimes_{i:\#B} \mathcal{A}(A_{\psi i}, B_i) \quad (2)$$

The following categories all have objects finite lists of elements from \mathcal{A} :

- The free monoidal \mathbb{Q} -category $L(\mathcal{A})$ has hom objects $L(\mathcal{A})(A, B)$ given by expression (2) with ψ restricted to identity functions.
- The free symmetric monoidal \mathbb{Q} -category $M(\mathcal{A})$ has hom objects $M(\mathcal{A})(A, B)$ given by expression (2) with ψ restricted to permutations.
- If \mathbb{Q} is deleting, the free deleting \mathbb{Q} -category $D(\mathcal{A})$ has hom objects $D(\mathcal{A})(A, B)$ given by expression (2) with ψ restricted to injective functions.
- If \mathbb{Q} is copying, the free copying \mathbb{Q} -category $C(\mathcal{A})$ has hom objects $C(\mathcal{A})(A, B)$ given by expression (2) with ψ restricted to surjective functions.
- If \mathbb{Q} is Cartesian, the free Cartesian \mathbb{Q} -category $K(\mathcal{A})$ has hom objects $K(\mathcal{A})(A, B)$ given by expression (2) with ψ ranging over all functions.

Proof. We sketch the required argument. In each case, the universal morphism is given by the map to the singleton list, which can be verified to be a \mathbb{Q} -functor. It follows from the universal property of the free monoid construction on sets that there is a unique possible fill in \mathbb{Q} -functor. This can be confirmed by direct calculation, exploiting the additional properties of \mathbb{Q} in the deleting, copying and Cartesian cases. ◀

Given they result from a free / forgetful adjunction, each of the constructions of theorem 27 yields a monad on $\mathbf{Cat}(\mathbb{Q})$. We wish to lift this structure to profunctors. As profunctors are analogous to binary relations, we might expect they arise as the Kleisli category of a generalization of the powerset monad. Recall [19] that the presheaves on a \mathbb{Q} -category form a \mathbb{Q} -category themselves. In fact, this is the free cocompletion, in the enriched sense. In general this construction does not induce a monad as there are size issues, leading to the need for more complex machinery [9]. In the case of quantale enrichment, we are fortunate as this problem goes away, and it can be shown that $\mathbf{Prof}(\mathbb{Q})$ is the Kleisli category of the free cocompletion monad. Lifting a monad to $\mathbf{Prof}(\mathbb{Q})$ can then be done by exhibiting an appropriate distributive law [1].

► **Theorem 28.** Let \mathbb{Q} be a commutative quantale, P the free cocompletion comonad, \mathcal{A} a \mathbb{Q} -category, A a list of \mathcal{A} -objects, and F a list of presheaves on \mathcal{A} . Define:

$$\bigvee_{\psi:\#F \rightarrow \#A} \otimes_{i:\#F} F_i A_{\psi i} \quad (3)$$

- There is a distributive law $\lambda^L : LP \Rightarrow PL$ with $\lambda^L_{\mathcal{A}}(F)(A)$ given by expression (3), with ψ restricted to identity functions.
- There is a distributive law $\lambda^M : MP \Rightarrow PM$ with $\lambda^M_{\mathcal{A}}(F)(A)$ given by expression (3), with ψ restricted to permutations.
- If \mathbb{Q} is deleting, there is a distributive law $\lambda^D : DP \Rightarrow PD$ with $\lambda^D_{\mathcal{A}}(F)(A)$ given by expression (3), with ψ restricted to injective functions.

- If \mathbb{Q} is copying, there is a distributive law $\lambda^C : CP \Rightarrow PC$ with $\lambda_{\mathcal{A}}^C(F)(A)$ given by expression (3), with ψ restricted to surjective functions.
- If \mathbb{Q} is Cartesian, there is a distributive law $\lambda^K : KP \Rightarrow PK$ with $\lambda_{\mathcal{A}}^K(F)(A)$ given by expression (3), with ψ ranging over all functions.

Proof. We can only sketch the proof. We first confirm that the components of each law are valid \mathbb{Q} -functors, and naturality of their components. With this in place, we verify Beck’s axioms [1] by direct calculation. This is a long series of calculations to cover all the cases. Generally establishing the unit laws is routine. The naturality checks and multiplication laws are less straightforward, particularly in the deleting, copying and Cartesian cases. In these cases, we must carefully apply the additional quantale axioms to confirm the required properties, effectively by “copying” and “deleting” sub-terms in calculations. ◀

► **Corollary 29.** *As $\mathbf{Prof}(\mathbb{Q})$ is self-dual, each of the constructions L, M, D, C, K induces a comonad $(!, \epsilon, \delta)$ ³ on $\mathbf{Prof}(\mathbb{Q})$, with action on morphisms:*

$$!R(A, B) = \bigvee_{\psi: \#B \rightarrow \#A} \otimes_{i: \#B} R(A_{\psi i}, B_i)$$

Where ψ is restricted appropriately as in theorem 27. The component of the counit and comultiplication at \mathcal{A} are:

$$\begin{array}{ll} \epsilon_{\mathcal{A}} : !\mathcal{A} \rightarrow \mathcal{A} & \delta_{\mathcal{A}} : !\mathcal{A} \rightarrow !!\mathcal{A} \\ \epsilon_{\mathcal{A}}(A, a) = !\mathcal{A}(A, [a]) & \delta_{\mathcal{A}}(A, \underline{A}) = !\mathcal{A}(A, \text{concat } \underline{A}) \end{array}$$

Here, *concat* denotes list concatenation.

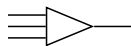
Proof. Although this is a natural construction, it is necessary to be careful with the various dualities involved, as some of the constructions on profunctors are necessarily oriented in nature. ◀

From the point of view of resources, the quantale value $\epsilon_{\mathcal{A}}(A, a)$ is the best way to convert the list A into the single $[a]$, using the structural features of $!\mathcal{A}$. Similarly, $\delta_{\mathcal{A}}(A, \underline{A})$ is the best way to convert the lists \mathcal{A} into the concatenation of the list of list \underline{A} using the structural features of $!\mathcal{A}$.

► **Remark.** These co-Kleisli categories carry a lot of additional structure that unfortunately we have insufficient space to exploit here. This includes further enrichment, various type constructors, and operations induced by the Day convolution [5]. Depending on the choice of comonad, there may also be higher order and differential structure [6]. This provides a rich algebra for calculations involving quantitative resources, formally similar to the calculus of generalized species presented in [7, 8].

5 Multicategories

If we examine a morphism $\mathcal{A} \rightarrow \mathcal{B}$ in the co-Kleisli category of one of the comonads in section 4, concretely, this is a profunctor of the form $!\mathcal{A} \rightarrow \mathcal{B}$. From a resource perspective, we can read this as describing conversions from lists of \mathcal{A} -resources to \mathcal{B} -resources. So the comonad allows us to describe many-to-one resource conversions, diagrammatically:



³ Our notation is a nod to connections with relational linear logic models.

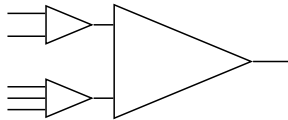
Depending on our choice of comonad, we can incorporate different structural aspects of the free conversions available, for example we may be able to cheaply reorder, copy or delete resources.

It is instructive to consider the co-Kleisli composition $S \bullet R$ of two such morphisms. This is given in $\mathbf{Prof}(\mathbb{Q})$ by the composite:

$$!A \xrightarrow{\delta_A} !!A \xrightarrow{!R} !B \xrightarrow{S} C$$

Intuitively, we can read the three steps as follows:

1. We first break our list of resources up into a list of lists, using the comultiplication δ_A .
2. We then use the resource conversions provided by R to process each of the sub-lists.
3. Finally, we process the resulting list using S , resulting in a two step multi-input conversion, which we might depict:



Then $(S \bullet R)(A, c)$ gives the best two stage conversion achievable converting the list of resources A to the resource c . The choice of comonad incorporates the structural aspects, such as copying or deleting, that we are prepared to permit.

What if we want to consider repeated many-to-one conversions? For that we must confirm a bit more structure is available.

► **Proposition 30.** *Each of the comonads of corollary 29 preserves non-empty joins.*

► **Corollary 31.** *Each of the co-Kleisli categories of these comonads is a non-empty join semilattice enriched category.*

It therefore makes sense to consider internal monads in our co-Kleisli categories. These internal monads quantify what we might call **multi-conversions**, in a manner that is closed under identities and composition. That is, they are generalizations of coloured operads [25], otherwise termed multicategories [21].

► **Remark.** This perspective on internal monads in such co-Kleisli bicategories is discussed in [8, 17]. There, they restrict to internal monads on discrete categories. However, in our setting, multicategories with non-discrete endpoints are a *virtue*. They describe the freely available resource conversions. The discrete case would say that the only freely available resource conversions are the trivial ones.

► **Example 32.** Even in the \mathbb{B} -enriched case, such multicategories are interesting objects. They are a multi-input generalization of preorders, describing the possibility of various multi-conversions being achievable. Possible conversions can be chained together, and trivial conversions are available. The choice of comonad introduces additional structure. For example in the deleting case, the list of resources $[a, b, c]$ is always convertible to $[a]$, by discarding the other resources.

6 Bimodules

In section 3 we showed how single input - single output resource conversions could be modelled as profunctors. In sections 4 and 5 we introduced additional comonadic structure that allowed us to introduce many-to-one costly resource conversions. In this section we show that an extra layer of abstraction allows us to model freely available many-to-one conversions with our categorical framework. We require the notion of bimodule between monads.

► **Definition 33.** Let \mathcal{C} be a preorder enriched category. For internal monads $(\mathcal{A}, R^{\mathcal{A}})$ and $(\mathcal{B}, R^{\mathcal{B}})$, a **bimodule** of type $(\mathcal{A}, R^{\mathcal{A}}) \multimap (\mathcal{B}, R^{\mathcal{B}})$ is a \mathcal{C} -morphism $S : \mathcal{A} \rightarrow \mathcal{B}$ such that:

$$S \circ R^{\mathcal{A}} \subseteq S \quad \text{and} \quad R^{\mathcal{B}} \circ S \subseteq S$$

As with our previous mathematical structures, it is helpful to think of bimodules as binary relations respecting some additional structure.

► **Proposition 34.** *In a non-empty join semilattice enriched category \mathcal{C} , bimodules between monads include the identity morphisms, and are closed under both composition and joins in \mathcal{C} . They therefore form a non-empty join semilattice enriched category $\mathbf{Bimod}(\mathcal{C})$.*

Bimodules between monads can be defined more generally, but their composition becomes more complicated, requiring a coequalizer construction not present in proposition 34. Fortunately, the quantale enriched setting circumvents this additional complexity. Resource theoretically, bimodules on our co-Kleisli categories have good properties:

- As **coKleisli(!)** morphisms, they model multi-conversions between \mathcal{A} and \mathcal{B} resources.
- These conversions are closed under precomposition with the multi-conversions described by the monad $(\mathcal{A}, R^{\mathcal{A}})$.
- The conversions are also closed under post composition with the multi-conversions described by the monad $(\mathcal{B}, R^{\mathcal{B}})$.

That is, they are exactly the right categorical object for describing resource conversions respecting freely available multi-conversions. As a corollary of proposition 34, we note that:

► **Corollary 35.** *The category **coKleisli(!)** is complete join semilattice enriched for any of the comonads introduced in section 4.*

Corollary 35 tells us that we can take composites and unions of bimodules to build more interesting structures. Also, we can consider internal monads in the categories of bimodules of interest. These can be seen as *multicategories that respect freely available multi-conversions*.

7 Reflexive Transitive Closure

Given the importance of internal monads in earlier sections, we briefly consider how they can be constructed from simpler data in complete join semilattice enriched categories. We require a new definition.

► **Definition 36.** Let \mathcal{C} be a complete join semilattice enriched category. A monad $T : \mathcal{A} \rightarrow \mathcal{A}$ is **free** over an arbitrary endomorphism $R : \mathcal{A} \rightarrow \mathcal{A}$ if it is the least monad containing R .

We fall back on our intuition that each of our categories of interest can be interpreted as a category of generalized binary relations. It is therefore natural to ask if some operations on ordinary relations have analogues in this setting. The construction of immediate interest is a generalization of reflexive transitive closure.

► **Proposition 37.** *In a non-empty join semilattice enriched category \mathcal{C} , the free monad induced by an endomorphism $R : A \rightarrow A$ is given by:*

$$F(R) = \bigsqcup_i R^i \quad \text{where} \quad R^0 = 1_A \quad \text{and} \quad R^{n+1} = R \circ R^n$$

Recalling lemma 21, and corollaries 31 and 35, the categories **Prof**(\mathbb{Q}), **coKleisli**(!) for any of the hierarchy of comonads of section 4, and the categories of bimodules on these co-Kleisli categories are all appropriately enriched. Therefore, we can conclude:

► **Corollary 38.** *Every endomorphism in our categories of interest can be used to construct a free internal monad using the reflexive transitive closure construction of proposition 37.*

In this way we can take some basic data specifying one-to-one or many-to-one resource conversions of interest. We can close them under composition in a canonical way.

8 Conclusion

We presented a flexible foundation for constructing compositional, quantitative models of resources, within which:

- There is a clear separation of concerns between freely available resource conversion, encoded as objects in our categories, and the costly conversions, encoded in the morphisms.
- Profunctors quantify one-to-one resource conversions, parameterized by a choice of quantity such as costs or probabilities.
- Morphisms in suitable co-Kleisli categories describe many-to-one resource conversions, parameterized by a choice of structural features such as copying and deleting.
- Bimodules model many-to-one resource conversions in which the freely available conversions may also include such multi-conversions.
- Throughout, internal monads capture closure under composition, yielding generalizations of categories or multicategories suitable for the quantitative setting.
- Free internal monads provide a convenient mechanism for building these (multi)categories from simpler data.
- The underlying objects can be considered as generalized relations, or just certain matrices of truth values, meaning calculations do not require difficult mathematical machinery.

Our approach is open to extension. For example, it is natural to also consider multi-input to multi-output conversions, in the style of polycategories [28]. These can be formulated in a similar manner to that used in sections 4 and 5. For the ordinary categorical setting this is technically complex, and has been developed by Garner [11, 10, 12]. Given the degeneracy of quantale enriched categories, we anticipate a more elementary approach will be feasible, and aim to develop this in later work.

We have focused on models. It would be interesting to develop corresponding syntactic aspects, in the form of a suitable metalanguage. Discussions in the related work of [16] and [8] suggest such a language will have a process algebraic feel, but we leave the details to later work.

Finally, a more speculative suggestion. Exciting recent categorical work on compositional game theory [13] has shown surprising applications of category theory in economic settings. Given the intrinsic interest of economists in both resources and costs, it would be interesting to explore applications of our approach in that setting.

References

- 1 Jon Beck. Distributive laws. In *Seminar on triples and categorical homology theory*, pages 119–140. Springer, 1969.
- 2 Gérard Boudol. The lambda-calculus with multiplicities. In *International Conference on Concurrency Theory*, pages 1–6. Springer, 1993.
- 3 Fernando GSL Brandao, Michał Horodecki, Jonathan Oppenheim, Joseph M Renes, and Robert W Spekkens. Resource theory of quantum states out of thermal equilibrium. *Physical review letters*, 111(25):250404, 2013.
- 4 Bob Coecke, Tobias Fritz, and Robert W Spekkens. A mathematical theory of resources. *Information and Computation*, 250:59–86, 2016.
- 5 Brian Day. On closed categories of functors. In *Reports of the Midwest Category Seminar IV*, pages 1–38. Springer, 1970.
- 6 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1-3):1–41, 2003.
- 7 Marcelo Fiore. Generalised species of structures: Cartesian closed and differential structures, 2004. Talk slides.
- 8 Marcelo Fiore. Mathematical models of computational and combinatorial structures. In *International Conference on Foundations of Software Science and Computation Structures*, pages 25–46. Springer, 2005.
- 9 Marcelo Fiore, Nicola Gambino, Martin Hyland, and Glynn Winskel. Relative pseudomonads, Kleisli bicategories, and substitution monoidal structures. *Selecta Mathematica*, pages 1–40, 2016.
- 10 Richard Garner. Double clubs. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 47(4):261–317, 2006.
- 11 Richard Garner. *Polycategories*. PhD thesis, University of Cambridge, 2006.
- 12 Richard Garner. Polycategories via pseudo-distributive laws. *Advances in Mathematics*, 218(3):781–827, 2008.
- 13 Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 472–481, 2018. doi:10.1145/3209108.3209165.
- 14 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.
- 15 Gilad Gour, Markus P Müller, Varun Narasimhachar, Robert W Spekkens, and Nicole Yunger Halpern. The resource theory of informational nonequilibrium in thermodynamics. *Physics Reports*, 583:1–58, 2015.
- 16 Martin Hyland. Some reasons for generalising domain theory. *Mathematical Structures in Computer Science*, 20(2):239–265, 2010.
- 17 Martin Hyland. Elements of a theory of algebraic theories. *Theoretical Computer Science*, 546:132–144, 2014.
- 18 Peter T Johnstone. *Stone spaces*, volume 3. Cambridge University Press, 1986.
- 19 Max Kelly. *Basic concepts of enriched category theory*, volume 64. CUP Archive, 1982. Available as a TAC reprint.
- 20 Max Kelly and Miguel L Laplaza. Coherence for compact closed categories. *Journal of Pure and Applied Algebra*, 19:193–213, 1980.
- 21 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.
- 22 F William Lawvere. Metric spaces, generalized logic, and closed categories. *Rendiconti del seminario matematico e fisico di Milano*, 43(1):135–166, 1973.


- 23 Tom Leinster. *Higher operads, higher categories*, volume 298. Cambridge University Press, 2004.
- 24 Iman Marvian and Robert W Spekkens. The theory of manipulations of pure state asymmetry: I. Basic tools, equivalence classes and single copy transformations. *New Journal of Physics*, 15(3):033001, 2013.
- 25 J Peter May. *The Geometry of Iterated Loop Spaces*. Springer, 1972.
- 26 Lambert Meertens. Algorithmics-towards programming as a mathematical activity. *Mathematics and Computer Science*, 1, 1986. *CWI Monographs (JW de Bakker, M. Hazewinkel, JK Lenstra, eds.) North Holland, Puhl. Co*, 1986.
- 27 Dusko Pavlovic. Quantitative concept analysis. In *International Conference on Formal Concept Analysis*, pages 260–277. Springer, 2012.
- 28 ME Szabo. Polycategories. *Communications in Algebra*, 3(8):663–689, 1975.
- 29 Thomas Theurer, Nathan Killoran, Dario Egloff, and Martin B Plenio. Resource theory of superposition. *Physical review letters*, 119(23):230401, 2017.

On Compositionality of Dinatural Transformations

Guy McCusker¹

University of Bath, United Kingdom


G.A.McCusker@bath.ac.uk

 <https://orcid.org/0000-0002-0305-6398>

Alessio Santamaria²

University of Bath, United Kingdom

A.Santamaria@bath.ac.uk

 <https://orcid.org/0000-0001-7683-5221>

Abstract

Natural transformations are ubiquitous in mathematics, logic and computer science. For operations of mixed variance, such as currying and evaluation in the lambda-calculus, Eilenberg and Kelly's notion of extranatural transformation, and often the even more general dinatural transformation, is required. Unfortunately dinaturals are not closed under composition except in special circumstances. This paper presents a new sufficient condition for composability.

We propose a generalised notion of dinatural transformation in many variables, and extend the Eilenberg-Kelly account of composition for extranaturals to these transformations. Our main result is that a composition of dinatural transformations which creates no cyclic connections between arguments yields a dinatural transformation.

We also extend the classical notion of horizontal composition to our generalized dinaturals and demonstrate that it is associative and has identities.

2012 ACM Subject Classification Theory of computation → Categorical semantics, Theory of computation → Proof theory

Keywords and phrases Dinatural transformation, categorical logic, compositionality

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.33

1 Introduction

Natural transformations are a ubiquitous notion in mathematics, logic and computer science. They are used to interpret logical rules, program forming operations, adjointness conditions and free constructions. Naturality is an equational property that expresses the idea that the transformation operates on structure, independent of the underlying data. Given functors $F, G: \mathbb{C} \rightarrow \mathbb{D}$, a natural transformation $\varphi: F \rightarrow G$ comprises a family of morphisms $\varphi_A: F(A) \rightarrow G(A)$ in \mathbb{D} . The naturality condition is specified as a commutative diagram

$$\begin{array}{ccc} F(A) & \xrightarrow{F(f)} & F(B) \\ \varphi_A \downarrow & & \downarrow \varphi_B \\ G(A) & \xrightarrow{G(f)} & G(B) \end{array}$$

¹ The author acknowledges the support of EPSRC grant EP/K033042/1.

² The author acknowledges the support of an EPSRC Doctoral Training Partnership studentship at the University of Bath.



33:2 On Compositionality of Dinatural Transformations

which may alternatively be pictured as follows:

$$\begin{array}{ccc}
 F\left(\boxed{f}\right) & & F\left(\boxed{id}\right) \\
 \varphi \downarrow & = & \varphi \downarrow \\
 G\left(\boxed{id}\right) & & G\left(\boxed{f}\right)
 \end{array}$$

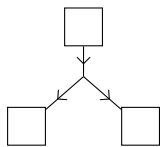
In this picture, each box represents an argument of a functor, while the vertical lines instances of the transformation. With this reading, the diagrammatic equation above is the same as the standard naturality square. Later we will give a precise meaning to pictures of this kind and make use of them in our proofs. For now, they will provide useful intuition.

When dealing with logic, type theory and programming languages, often we encounter transformations between functors of mixed variance, like the evaluation map $ev_{A,B}: A \times (A \Rightarrow B) \rightarrow B$. Eilenberg and Kelly [5] developed the notion of extranatural transformation to account for this. The equational property of ev is explained by means of graphs:

$$\begin{array}{ccc}
 A \times (A' \Rightarrow B) & \xrightarrow{f \times (id \Rightarrow g)} & A' \times (A' \Rightarrow B) \\
 \downarrow id \times (f \Rightarrow id) & & \downarrow ev_{A', B'} \\
 A \times (A \Rightarrow B') & \xrightarrow{ev_{A, B}} & B'
 \end{array}
 \quad \Leftarrow \quad
 \begin{array}{ccc}
 \boxed{f} \times (\boxed{id} \Rightarrow \boxed{g}) & & \boxed{id} \times (\boxed{f} \Rightarrow \boxed{id}) \\
 \downarrow A' & & \downarrow A \\
 \boxed{id} & & \boxed{g}
 \end{array}$$

(Grey boxes indicate contravariant arguments of the functors involved.) $ev = (ev_{A,B})$ is said to be extranatural in A and natural in B . Note that the connections show which arguments of the functors involved must be set equal in order for an equational property to hold. There is one connected component in the graph for each such collection of arguments. Eilenberg and Kelly show that a composite of extranatural transformations is again extranatural, provided the graph obtained by pasting the graphs together along the common interface is acyclic.

In the graphs of all the transformations we have seen so far, arguments are linked in pairs. For transformations such as the diagonal $\delta = (\delta_A: A \rightarrow A \times A)$, this is a limitation. Though its equational properties are adequately described using the naturality of a transformation from the identity functor $id_{\mathbb{C}}$ to the diagonal functor Δ , this account becomes clumsy if we attempt to discuss the associativity of the diagonal operation, for example. One would prefer to picture it as:



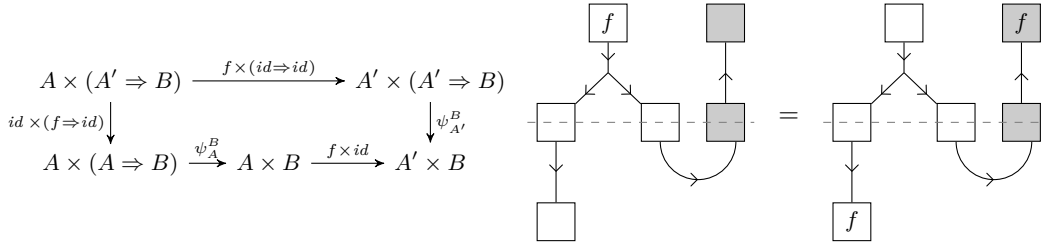
Kelly [13] points this out, and suggests that a more general notion of natural transformation, in which graphs have ramifications, may be available, but does not go on to develop it.

Such ramifications have ramifications. One source of difficulty is that composing these generalised natural and extranatural transformations quickly leads to *dinatural transformations* [4]. For example, working in a cartesian closed category, by composing the diagonal δ and evaluation $ev^B = (ev_A^B: A \times (A \Rightarrow B) \rightarrow B)_{A \in \mathbb{C}}$, we can construct morphisms

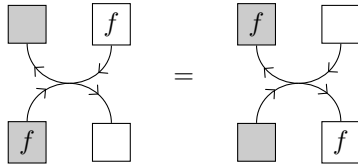
$$\psi_A^B = \delta_A \times id_{A \Rightarrow B}; id_A \times ev_A^B: A \times (A \Rightarrow B) \rightarrow A \times B.$$

By pasting together the graphs of the transformations $\delta_A \times id_{A \Rightarrow B}$ and $id_A \times ev_A^B$ we obtain the following depiction of the appropriate “naturality” in A for this family of morphisms.

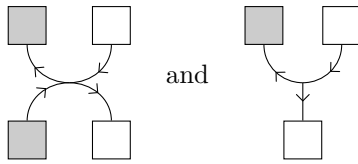
(From now on we drop the name of the functors involved and retain only the boxes and the lines, an empty box being the same as a box containing an identity.)



This is not a natural nor an extranatural transformation, but a dinatural transformation. Dinatural transformations are families of morphisms between functors of the form $\mathbb{C}^{\text{op}} \times \mathbb{C} \rightarrow \mathbb{C}$ where the dinaturality condition can be drawn as follows:



Dinatural transformations arise often in a computer science context. For instance the Church numerals $n = (n_A : (A \Rightarrow A) \rightarrow (A \Rightarrow A))$ and the fixed point combinator $Y = (Y_A : (A \Rightarrow A) \rightarrow A)$ are dinatural transformations, with graphs

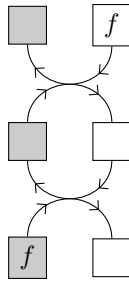


(We note Curry’s prescience in his naming of the Y combinator.) More generally, dinatural transformations have been proposed as a suitable way to understand parametric polymorphism [1] and as an interpretation of cut-free proofs, or equivalently typed lambda terms [8]. But dinatural transformations suffer from a troublesome shortcoming: they do not compose.

Our pictorial representation makes it clear that there is no reason to expect these to be closed under composition: starting from the situation as pictured on Figure 1 below, there is no way to reach a situation where the dinaturality of either transformation may be applied. Under special circumstances, such as when certain squares of morphisms are pullbacks or pushouts, the composite may turn out to be dinatural, but not as a direct consequence of the dinaturality of the two transformations.

In contrast to Eilenberg and Kelly’s treatment of extranatural transformations, the usual description of dinatural transformations concerns functors of one argument (strictly speaking two arguments, of different variance, that are required to be equal). A consequence of this is that any composition of dinaturals appears to have a cyclic dependency among arguments, as seen in Figure 1.

In this paper we introduce a generalised notion of dinatural transformation. As in [13], our transformations are equipped with a graph as part of their data whose composition does not always form a cycle. These transformations enjoy a similar compositionality property to the extranaturals: as long as no cycles are created, dinaturality is preserved by composition.



■ **Figure 1** Cycles and impossibility to apply dinaturality.

Thus, one is freed from the burden of conducting ad hoc verification of dinaturality conditions. For example, the dinaturality theorems of [8] can readily be proved by drawing the graphs of the transformations interpreting cut-free proofs and observing that they are acyclic. The proof of our result is significantly more demanding than Eilenberg and Kelly’s case, because of the ramifications in the dependency graphs. But to a computer scientist these graphs have a familiar appearance: they look and behave like Petri nets. Our argument proceeds by formalising a correspondence between morphisms built from functors and dinatural transformations and configurations of Petri nets. The desired dinaturality equation reduces to a question of reachability of one configuration from another, which is readily settled using the theory of Petri nets. In this way we not only discover a helpful sufficient condition for composability of dinaturals but also turn an intuitive diagrammatic reasoning method into a formal tool. Moreover, one can show that ours is also an “essentially necessary” condition: if the dinaturality of a composite transformation $\varphi; \psi$ may be derived using only the dinaturality of φ and ψ , then the composite graph is acyclic (cf. [12, §1.3]). For lack of space we do not present the proof of this fact here.

The above discussion concerns only the “vertical” composition of transformations. Natural transformations may also be composed horizontally; this operation is needed when one wishes to substitute functors for the arguments of other functors, and apply transformations between them. Kelly already noticed this in his generalisation of Godement calculus for functors and natural transformations in many variables [13]. To date we are not aware of any generalisation of this operation to dinatural transformations. Our second contribution is to develop a notion of horizontal composition for our dinatural transformations that extends the well known version for natural transformations and establish that it is associative and has identities. Unfortunately, we seem to have lost one of the fundamental properties of horizontal composition of natural transformations: compatibility with the vertical one. Indeed, an analogous version of interchange law for the natural case does not (*cannot*) hold with dinatural transformations, even when we restrict ourselves to simple cases. The problem stems from the “shape”, as it were, of the dinaturality condition, that prevents the vertical composability of the two horizontal compositions. A different kind of interchange law seems to be needed, but as yet we have not been able to find one that works, not even for Eilenberg and Kelly’s transformations with no ramifications. We shall dedicate the near future to investigate this matter and hopefully we shall solve this rather *natural* problem.

Related work. Our interest in this topic arose from a desire to understand better the algebraic properties of Guglielmi and Gundersen’s atomic flows [10, 9], which are an abstraction of information flow in classical logic proofs. The graphical structures we use extend so-called Kelly-Mac Lane graphs [14] which originated with Eilenberg and Kelly [5] and may be seen

as string diagrams for closed categories; the wide variety of string diagrams is surveyed in [19]. They are closely related to proof nets encountered in the proof theory of linear logic [7]. Blute [2] studies dinatural transformations corresponding to proofs of multiplicative linear logic and establishes a compositionality result for that case. Freyd, Robinson and Rosolini [6] studied dinaturality in the category of PERs. The close relationship between dinaturality and fixed point combinators was studied by Mulry [16] and Simpson [20].

► **Notation.** We denote by \mathbb{I} the category with one object and one morphism. Let $\alpha \in \text{List}\{+, -\}$, $|\alpha| = n$. We refer to the i -th element of α as α_i . We denote by $\bar{\alpha}$ the list obtained from α by swapping the signs. Given a category \mathbb{C} , if $n \geq 1$, then we define $\mathbb{C}^\alpha := \mathbb{C}^{\alpha_1} \times \dots \times \mathbb{C}^{\alpha_n}$, with $\mathbb{C}^+ = \mathbb{C}$ and $\mathbb{C}^- = \mathbb{C}^{\text{op}}$, otherwise $\mathbb{C}^\alpha := \mathbb{I}$. Composition of morphisms $f: A \rightarrow B$ and $g: B \rightarrow C$ will be denoted by $g \circ f$, gf or also $f; g$. \mathbb{N} is the set of natural numbers, including 0. Given $n \in \mathbb{N}$, we ambiguously denote n for both the number n and the set $\{1, \dots, n\}$.

2 Dinatural transformations, types and vertical composition

To make precise the graphical ideas introduced above, we employ a notion of *type* for our transformations, following Kelly [13]. The type indicates how the various arguments of the domain and codomain functors are related by naturality conditions.

The category $\mathbb{T}\text{ypes}$. Let $\mathbb{T}\text{ypes}$ be the category of cospans [3] of finite sets and functions; that is, $\mathbb{T}\text{ypes}$ has \mathbb{N} as its set of objects, and a morphism $f: n \rightarrow m$ is a cospan $f = (n \xrightarrow{\sigma} k \xleftarrow{\tau} m)$; different cospans counting as the same morphism if they differ only by an automorphism, that is a permutation, of k . Given $n \in \mathbb{N}$, the identity morphism on n is the cospan of id_n . Composition of f and $g = (m \xrightarrow{\sigma'} p \xleftarrow{\tau'} t)$ is the cospan $gf = (n \rightarrow q \leftarrow t)$ got by computing the pushout of τ against σ' as functions:

$$\begin{array}{ccc}
 & & t \\
 & & \downarrow \tau' \\
 m & \xrightarrow{\sigma'} & p \\
 \tau \downarrow & \lrcorner & \downarrow \xi \\
 n & \xrightarrow{\sigma} & k \xrightarrow{\zeta} q
 \end{array} \tag{1}$$

Transformations. Throughout this section, we fix a category \mathbb{C} .

► **Definition 1.** Let $\alpha, \beta \in \text{List}\{+, -\}$, $T: \mathbb{C}^\alpha \rightarrow \mathbb{C}$, $S: \mathbb{C}^\beta \rightarrow \mathbb{C}$ functors. A *transformation* $\varphi: T \rightarrow S$ of *type* $f = (|\alpha| \xrightarrow{\sigma} k \xleftarrow{\tau} |\beta|)$ (with k positive integer) is a family of morphisms

$$\left(\varphi_{A_1, \dots, A_k}: T(A_{\sigma_1}, \dots, A_{\sigma_{|\alpha|}}) \rightarrow S(A_{\tau_1}, \dots, A_{\tau_{|\beta|}}) \right)_{(A_1 \dots A_k) \in \mathbb{C}^k}.$$

Functions σ and τ tell us which of the $|\alpha|$ arguments of T and the $|\beta|$ arguments of S must be equated, and also which among A_1, \dots, A_k to use in each “slot”. Notice that σ and τ need not be surjective, so we can define transformations with “unused variables”.

► **Definition 2.** Let $\varphi: T \rightarrow S$ of type f be a transformation as in Definition 1, $R: \mathbb{C}^\gamma \rightarrow \mathbb{C}$ and $\psi: S \rightarrow R$ a transformation of type $g = (|\beta| \xrightarrow{\sigma'} p \xleftarrow{\tau'} |\gamma|)$, so that we have, for all B_1, \dots, B_p ,

$$\psi_{B_1, \dots, B_p}: S(B_{\sigma'_1}, \dots, B_{\sigma'_p}) \rightarrow R(B_{\tau'_1}, \dots, B_{\tau'_p}).$$

33:6 On Compositionality of Dinatural Transformations

The *vertical composition* $\psi \circ \varphi$ is defined as the transformation of type

$$gf = |\alpha| \xrightarrow{\zeta\sigma} q \xleftarrow{\xi\tau'} |\gamma|$$

where ζ and ξ are given by (1) and $(\psi \circ \varphi)_{C_1, \dots, C_q}$ is the composite:

$$\begin{aligned} T(C_{\zeta\sigma 1}, \dots, C_{\zeta\sigma|\alpha|}) &\xrightarrow{\varphi_{C_{\zeta\sigma 1}, \dots, C_{\zeta\sigma k}}} S(C_{\zeta\tau 1}, \dots, C_{\zeta\tau|\beta|}) \\ &\parallel \\ S(C_{\xi\sigma' 1}, \dots, C_{\xi\sigma'|\beta|}) &\xrightarrow{\psi_{C_{\xi\sigma' 1}, \dots, C_{\xi\sigma' p}}} R(C_{\xi\tau' 1}, \dots, C_{\xi\tau'|\gamma|}) \end{aligned}$$

(Notice that by definition $\varphi_{C_{\zeta\sigma 1}, \dots, C_{\zeta\sigma k}}$ requires that the i -th variable of T be the σi -th element of the list $(C_{\zeta\sigma 1}, \dots, C_{\zeta\sigma k})$, which is indeed $C_{\zeta\sigma i}$.)

► **Definition 3.** Consider $T: \mathbb{C}^\alpha \rightarrow \mathbb{C}$, $S: \mathbb{C}^\beta \rightarrow \mathbb{C}$, $\varphi: T \rightarrow S$ a transformation of type $|\alpha| \xrightarrow{\sigma} k \xleftarrow{\tau} |\beta|$ as in Definition 1. For $i \in \{1, \dots, k\}$, we say that φ is *dinatural in A_i* (or, more precisely, *in its i -th variable*) if and only if for all $A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_k$ objects of \mathbb{C} and for all $f: A \rightarrow B$ in \mathbb{C} the following diagram commutes:

$$\begin{array}{ccc} & \xrightarrow{\varphi_{A_1, \dots, A_{i-1}, B, A_{i+1}, \dots, A_k}} & \\ T(x_1, \dots, x_{|\alpha|}) & \nearrow & \cdot \xrightarrow{S(y_1, \dots, y_{|\beta|})} \cdot \\ & \searrow & \\ T(x'_1, \dots, x'_{|\alpha|}) & \searrow & \cdot \xrightarrow{S(y'_1, \dots, y'_{|\beta|})} \cdot \\ & \xrightarrow{\varphi_{A_1, \dots, A_{i-1}, A, A_{i+1}, \dots, A_k}} & \end{array}$$

where

$$x_j = \begin{cases} f & \sigma j = i \wedge \alpha_j = + \\ id_B & \sigma j = i \wedge \alpha_j = - \\ id_{A_{\sigma j}} & \sigma j \neq i \end{cases} \quad y_j = \begin{cases} id_B & \tau j = i \wedge \beta_j = + \\ f & \tau j = i \wedge \beta_j = - \\ id_{A_{\tau j}} & \tau j \neq i \end{cases}$$

$$x'_j = \begin{cases} id_A & \sigma j = i \wedge \alpha_j = + \\ f & \sigma j = i \wedge \alpha_j = - \\ id_{A_{\sigma j}} & \sigma j \neq i \end{cases} \quad y'_j = \begin{cases} f & \tau j = i \wedge \beta_j = + \\ id_A & \tau j = i \wedge \beta_j = - \\ id_{A_{\tau j}} & \tau j \neq i \end{cases}$$

► **Remark.** Definition 3 is a generalisation of the well known notion of dinatural transformation, which we can obtain when $\alpha = \beta = [-, +]$ and $k = 1$. Here we are allowing multiple variables at once and the possibility for T and S of having an arbitrary number of copies of \mathbb{C} and \mathbb{C}^{op} in their domain, for each variable $i \in \{1, \dots, k\}$.

It is known that dinatural transformations generalise natural and extranatural ones. Here we make this fact explicit by defining the latter as particular cases of dinatural transformations where the functors and the type have a special shape: essentially, a dinatural transformation $\varphi: T \rightarrow S$ is natural in A_i if T and S are both covariant or both contravariant in the variables involved by A_i ; φ is extranatural in A_i if one of the functors T and S does not involve the variable A_i while A_i appears both covariantly and contravariantly in the other.

► **Definition 4.** Let $\varphi: T \rightarrow S$ be a transformation as in Definition 1. $\varphi = (\varphi_{A_1, \dots, A_k})$ is said to be *natural in A_i* if and only if

- it is dinatural in A_i ;
- $\forall u \in \sigma^{-1}\{i\}. \forall v \in \tau^{-1}\{i\}. (\alpha_u = \beta_v = +) \vee (\alpha_u = \beta_v = -)$.

φ is called *extranatural* in A_i if and only if

- it is dinatural in A_i ;
- $(\sigma^{-1}\{i\} = \emptyset \wedge \exists j_1, j_2 \in \tau^{-1}\{i\}. \beta_{j_1} \neq \beta_{j_2}) \vee (\tau^{-1}\{i\} = \emptyset \wedge \exists i_1, i_2 \in \sigma^{-1}\{i\}. \alpha_{i_1} \neq \alpha_{i_2})$.

Notice that our notion of (extra)natural transformations is more general than the one given by Eilenberg and Kelly in [5], as we allow the arguments of T and S to be equated not just in pairs, but in an arbitrary number, according to σ and τ .

► **Example 5.** Suppose that \mathbb{C} is a cartesian category, with $\times: \mathbb{C} \times \mathbb{C} \rightarrow \mathbb{C}$ the product functor, and consider the diagonal transformation $\delta = (\delta_A: A \rightarrow A \times A)_{A \in \mathbb{C}}: id_{\mathbb{C}} \rightarrow \times$ of type $1 \rightarrow 1 \leftarrow 2$. We have that δ is natural in its only variable.

► **Example 6.** Suppose that \mathbb{C} is a cartesian closed category, fix an object R in \mathbb{C} , and consider the functor

$$\begin{array}{ccc} \mathbb{C} \times \mathbb{C}^{\text{op}} & \xrightarrow{T} & \mathbb{C} \\ (A, A') & \longmapsto & (A' \Rightarrow R) \times A \end{array}$$

The evaluation $ev^R = (ev_A^R: T(A, A) \rightarrow R)_{A \in \mathbb{C}}: T \rightarrow R$ is a transformation of type $2 \rightarrow 1 \leftarrow 0$ which is extranatural in its only variable.

We proceed now to study the composability problem for dinatural transformations. Let $\varphi: F_1 \rightarrow F_2$ and $\psi: F_2 \rightarrow F_3$ be transformations where

- $F_i: \mathbb{C}^{\alpha^i} \rightarrow \mathbb{C}$ is a functor for all $i \in \{1, 2, 3\}$,
- φ and ψ have type, respectively,

$$|\alpha^1| \xrightarrow{\sigma_1} k_1 \xleftarrow{\tau_1} |\alpha^2| \quad \text{and} \quad |\alpha^2| \xrightarrow{\sigma_2} k_2 \xleftarrow{\tau_2} |\alpha^3|.$$

We shall establish conditions under which $\psi \circ \varphi$ is dinatural in some of its variables. In order to do so, we associate to $\psi \circ \varphi$ a graph which somehow reflects the signature of φ and ψ .

The graph of $\psi \circ \varphi$. We assign to $\psi \circ \varphi$ a directed bipartite graph $\Gamma(\psi \circ \varphi)$ whose vertices are given by (distinct) finite sets P and T , while $\bullet-, -\bullet: T \rightarrow \mathcal{P}(P)$ are the input and output functions for elements in T (that is, there is an arc from p to t if and only if $p \in \bullet t$, and there is an arc from t to p if and only if $p \in -t$), as follows: $P = |\alpha^1| + |\alpha^2| + |\alpha^3|$, $T = k_1 + k_2$ and, indicating with $\iota_i: |\alpha^i| \rightarrow P$ and $\rho_i: k_i \rightarrow T$ the canonical injections,

$$\bullet(\rho_i(t)) = \{\iota_i(p) \mid \sigma_i(p) = t, \alpha_p^i = +\} \cup \{\iota_{i+1}(p) \mid \tau_i(p) = t, \alpha_p^{i+1} = -\}$$

$$(\rho_i(t))\bullet = \{\iota_i(p) \mid \sigma_i(p) = t, \alpha_p^i = -\} \cup \{\iota_{i+1}(p) \mid \tau_i(p) = t, \alpha_p^{i+1} = +\}$$

In other words, the inputs of a variable t of transformation φ are the covariant arguments of F_1 and the contravariant arguments of F_2 which are mapped by σ_1 and τ_1 , respectively, to t ; similarly for outputs of t (swapping ‘covariant’ and ‘contravariant’) and for variables of ψ . Graphically, we draw elements of P as white or grey boxes (if corresponding to a covariant or contravariant argument of an F_i , respectively), and elements of T as black squares, as in the following example.

► **Example 7.** Suppose that \mathbb{C} is cartesian closed, fix an object R in \mathbb{C} , consider functors

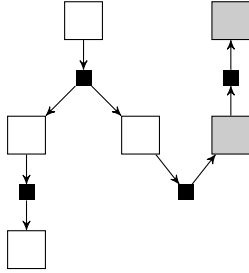
$$\begin{array}{ccccc} \mathbb{C} \times \mathbb{C}^{\text{op}} & \xrightarrow{F} & \mathbb{C} & \mathbb{C} \times \mathbb{C} \times \mathbb{C}^{\text{op}} & \xrightarrow{G} & \mathbb{C} & \mathbb{C} & \xrightarrow{H} & \mathbb{C} \\ (A, B) & \longmapsto & A \times (B \Rightarrow R) & (A, B, C) & \longmapsto & A \times B \times (C \Rightarrow R) & A & \longmapsto & A \times R \end{array}$$

33:8 On Compositionality of Dinatural Transformations

and transformations $\varphi = \delta \times id_{(-)\Rightarrow R}: F \rightarrow G$ and $\psi = id_C \times ev^R: G \rightarrow H$ of types, respectively,

$$\begin{array}{ccc}
 2 \xrightarrow{\sigma} 2 & \xleftarrow{\tau} & 3 \\
 1 \xrightarrow{\quad} 1 & \xleftarrow{\quad} & 1 \\
 2 \xrightarrow{\quad} 2 & \xleftarrow{\quad} & 3
 \end{array}
 \quad \text{and} \quad
 \begin{array}{ccc}
 3 \xrightarrow{\eta} 2 & \xleftarrow{\theta} & 1 \\
 1 \xrightarrow{\quad} 1 & \xleftarrow{\quad} & 1 \\
 2 \xrightarrow{\quad} 2 & & \\
 3 \xrightarrow{\quad} & &
 \end{array}
 .$$

Then $\psi \circ \varphi$ has type $2 \rightarrow 1 \leftarrow 1$ and its graph is:



► **Remark.** Each connected component of $\Gamma(\psi \circ \varphi)$ corresponds to a variable of $\psi \circ \varphi$. This is due to how the pushout of τ_1 against σ_2 is computed when we calculate the type of $\psi \circ \varphi$: if p is the result of the pushout, then p is isomorphic, in Set , to the quotient set of T modulo the least equivalence relation \sim such that for all $\rho_1(x)$ and $\rho_2(y)$, $\rho_1(x) \sim \rho_2(y)$ if and only if there exists $z \in |\alpha^2|$ such that $\tau_1(z) = x$ and $\sigma_2(z) = y$; in other words, if they are connected in $\Gamma(\psi \circ \varphi)$ (by means of an undirected path).

Since we want to discuss the dinaturality of $\psi \circ \varphi$ in each of its variables separately, we start by assuming that $\psi \circ \varphi$ is “connected”, that is has type $|\alpha^1| \rightarrow 1 \leftarrow |\alpha^3|$, and that φ and ψ are dinatural in all their variables. The result we want to prove is then the following.

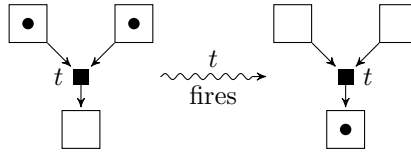
► **Theorem 8.** *Let φ and ψ be transformations which are dinatural in all their variables and such that $\psi \circ \varphi$ depends on only one variable. If $\Gamma(\psi \circ \varphi)$ is acyclic, then $\psi \circ \varphi$ is a dinatural transformation.*

We shall prove this theorem by interpreting $\Gamma(\psi \circ \varphi)$ as a Petri Net [18], whose set of places is P and of transitions is T . Places can host tokens, and recall that a marking for $\Gamma(\psi \circ \varphi)$ is a function $M: P \rightarrow \mathbb{N}$, that is, a distribution of tokens. A transition t is enabled in M if $M(p) > 0$ for all $p \in \bullet t$; an enabled transition t can fire, and the firing of t removes one token from each of its inputs and adds one token to each of its outputs, that is it generates a new marking M' defined as follows:

$$M'(p) = \begin{cases} M(p) - 1 & p \in \bullet t \\ M(p) + 1 & p \in t \bullet \\ M(p) & \text{otherwise} \end{cases}$$

Graphically, we draw tokens as black dots, see Figure 2.

The reason for which we use Petri Nets to prove Theorem 8 is that the firing of an enabled transition in $\Gamma(\psi \circ \varphi)$ corresponds to applying the dinaturality of φ or ψ in the corresponding variable, thus giving rise to an equation of morphisms in \mathbb{C} . It follows that a sequence of firings corresponds to a chain of equations. Since we are interested in proving that two certain morphisms, corresponding to the two legs of the hexagon that we want to show is



■ **Figure 2** The firing of an enabled transition t .

commutative (to prove that $\psi \circ \varphi$ is dinatural), are equal, we shall individuate two markings M_0 and M_d for $\Gamma(\psi \circ \varphi)$ that correspond to those morphisms, and prove that M_d is *reachable* from M_0 , that is that there is a sequence of firings of enabled transitions that transforms M_0 into M_d . This reduction to Petri nets not only provides an intuitive reasoning tool that corresponds directly to the diagrams we have been drawing, but also allows us to make use of the well-developed theory of Petri nets. Indeed our compositionality result will follow from a theorem about reachability in acyclic Petri nets.

► **Notation.** We extend the input and output notation for places too, where

$$\bullet p = \{t \in T \mid p \in t\bullet\}, \quad p\bullet = \{t \in T \mid p \in t\}$$

► **Remark.** Since σ_i and τ_i are functions, we have that $|\bullet p|, |p\bullet| \leq 1$ and also that $|\bullet p \cup p\bullet| \geq 1$. With a little abuse of notation then, if $\bullet p = \{t\}$ then we shall simply write $\bullet p = t$, and similarly for $p\bullet$.

Labelled markings. Not all markings for $\Gamma(\psi \circ \varphi)$ correspond to a morphism in \mathbb{C} . In this section we shall individuate a class of them for which it is possible to define an associated morphism in \mathbb{C} .

► **Definition 9.** Consider $f: A \rightarrow B$ a morphism in \mathbb{C} . A *labelled marking* is a triple (M, L, f) where functions $M: P \rightarrow \{0, 1\}$ and $L: T \rightarrow \{A, B\}$ are such that for all $p \in P$

$$M(p) = 1 \implies L(\bullet p) = A, L(p\bullet) = B$$

$$M(p) = 0 \implies \begin{cases} p\bullet = \emptyset \implies L(\bullet p) = B \\ \bullet p = \emptyset \implies L(p\bullet) = A \\ \bullet p \neq \emptyset \neq p\bullet \implies L(\bullet p) = L(p\bullet) \end{cases}$$

For each labelled marking (M, L, f) we define a morphism in \mathbb{C} obtained by composing the functors F_i with appropriate components of φ and ψ . Each argument of F_i corresponds to a place in the graph. For each marked place the corresponding F_i 's argument will be f ; for unmarked places it will be id . The definition of labelled marking puts constraints on the marking itself, ensuring that the result of this operation is a well-formed morphism in \mathbb{C} .

► **Definition 10.** Let $f: A \rightarrow B$ in \mathbb{C} , (M, L, f) a labelled marking. We define a morphism $\mu(M, L, f)$ in \mathbb{C} as follows:

$$\mu(M, L, f) = F_1(x_1^1, \dots, x_{|\alpha_1|}^1); \varphi_{X_1^1 \dots X_{k_1}^1}; F_2(x_1^2, \dots, x_{|\alpha_2|}^2); \psi_{X_1^2 \dots X_{k_2}^2}; F_3(x_1^3, \dots, x_{|\alpha_3|}^3)$$

where

$$x_j^i = \begin{cases} f & M(\iota_i(j)) = 1 \\ id_{L(\iota_i(j))} & M(\iota_i(j)) = 0 \wedge t \in \bullet p \cup p\bullet \end{cases} \quad X_j^i = L(\rho_i(j)).$$

33:10 On Compositionality of Dinatural Transformations

We proceed now to show that the firing of an enabled, B -labelled transition in a labelled marking yields an equation between the associated morphisms. Consider then (M, L, f) a labelled marking, t in T such that $L(t) = B$ and $M(p) = 1$ for all $p \in \bullet t$. Notice that necessarily $M(p) = 0$ for all $p \in t \bullet$ (otherwise we would have $L(t) = A$ by definition of labelled marking). Define functions $M': P \rightarrow \{0, 1\}$ and $L': T \rightarrow \{A, B\}$ as follows, for all $p \in P$ and $s \in T$:

$$M'(p) = \begin{cases} 0 & p \in \bullet t \\ 1 & p \in t \bullet \\ M(p) & \text{otherwise} \end{cases} \quad L'(s) = \begin{cases} A & s = t \\ L(s) & \text{otherwise} \end{cases}$$

(M' is the marking obtained from M by firing t .) It is an immediate consequence of the definition that (M', L', f) is still a labelled marking.

► **Proposition 11.** *In the notations above, $\mu(M, L, f) = \mu(M', L', f)$.*

Proof. Since $t \in T$, we have $t = \rho_u(i)$ for some $u \in \{1, 2\}$ and $i \in \{1, \dots, k_u\}$. The fact that t is enabled ensures that, in the notations of Definition 10,

$$\begin{aligned} \sigma_u(j) = i \wedge \alpha_j^u = + &\implies x_j^u = f \\ \sigma_u(j) = i \wedge \alpha_j^u = - &\implies x_j^u = id_B \\ \tau_u(j) = i \wedge \alpha_j^{u+1} = + &\implies x_j^{u+1} = id_B \\ \tau_u(j) = i \wedge \alpha_j^{u+1} = - &\implies x_j^{u+1} = f \end{aligned}$$

hence we can apply the dinaturality of φ or ψ (if, respectively, $u = 1$ or $u = 2$) in its i -th variable and obtain therefore a new morphism, which a simple check can show is equal to $\mu(M', L', f)$. ◀

It immediately follows that a sequence of firings of B -labelled transitions gives rise to a labelled marking whose associated morphism is still equal to the original one, as the following Proposition states.

► **Proposition 12.** *Let (M, L, f) be a labelled marking, M_d a marking reachable from M by firing only B -labelled transitions t_1, \dots, t_m , $L_d: T \rightarrow \{A, B\}$ defined as:*

$$L_d(s) = \begin{cases} A & s = t_i \text{ for some } i \in \{1, \dots, m\} \\ L(s) & \text{otherwise} \end{cases}$$

Then (M_d, L_d, f) is a labelled marking and $\mu(M, L, f) = \mu(M_d, L_d, f)$.

We have now to individuate the two markings M_0 and M_d which correspond to the two morphisms we want to prove to be equal to show that $\psi \circ \varphi$ is dinatural, when $\Gamma(\psi \circ \varphi)$ is acyclic. Since we are assuming that $\psi \circ \varphi: F_1 \rightarrow F_3$ depends on only one variable, those morphisms are:

$$\begin{aligned} \delta_1 &= F_1(x_1, \dots, x_{|\alpha^1|}); [\psi \circ \varphi]_B; F_3(y_1, \dots, y_{|\alpha^3|}) \\ \delta_2 &= F_1(x'_1, \dots, x'_{|\alpha^1|}); [\psi \circ \varphi]_A; F_3(y'_1, \dots, y'_{|\alpha^3|}) \end{aligned}$$

where

$$x_i = \begin{cases} f & \alpha_i^1 = + \\ id_B & \alpha_i^1 = - \end{cases} \quad y_i = \begin{cases} id_B & \alpha_i^3 = + \\ f & \alpha_i^3 = - \end{cases}$$

$$x'_i = \begin{cases} id_B & \alpha_i^1 = + \\ f & \alpha_i^1 = - \end{cases} \quad y'_i = \begin{cases} f & \alpha_i^3 = + \\ id_B & \alpha_i^3 = - \end{cases}$$

Now, f appears in all the covariant arguments of F_1 and the contravariant ones of F_3 , in δ_1 , which correspond in $\Gamma(\psi \circ \varphi)$ to those places which have no inputs (in Petri nets terminology, *sources*), whereas f appears, in δ_2 , in those arguments corresponding to places with no outputs (*sinks*). The two markings we are interested into are, therefore,

$$M_0(p) = \begin{cases} 1 & \bullet p = \emptyset \\ 0 & \text{otherwise} \end{cases} \quad M_d(p) = \begin{cases} 1 & p \bullet = \emptyset \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

What about the labelling? We have that $[\psi \circ \varphi]_B = \varphi_{B\dots B}; \psi_{B\dots B}$, hence we shall consider $L: T \rightarrow \{A, B\}$ constantly equal to B : it is easy to see that (M_0, L, f) is a labelled marking. Now all we have to show is that M_d is reachable from M_0 by only firing B -labelled transitions: it is enough to make sure that each transition is fired at most once to satisfy this condition. (Notice that since $\Gamma(\varphi)$ is acyclic, if a transition fires once than it will remain disabled for ever, hence no transition can fire more than once anyway.) In order to do that, we recall some general properties of Petri nets, see [17].

Every Petri Net N with n transitions and m places defines a $m \times n$ matrix of integers $A = [a_{pt}]$, called *incidence matrix* of N . In the case of a net with at most one arc between any two vertices (like $\Gamma(\psi \circ \varphi)$), we have

$$a_{pt} = \begin{cases} 1 & p \in t \bullet \\ -1 & p \in \bullet t \\ 0 & \text{otherwise} \end{cases}$$

It is not difficult to see that a_{pt} represents the number of tokens changed in place p when transition t fires once. If we represent an arbitrary marking M as a $m \times 1$ vector, we can state the following theorem [11], which gives a necessary and sufficient condition for reachability of a marking M_d from another marking M_0 in case N is acyclic.

► **Theorem 13.** *Let N be an acyclic Petri Net with m places and n transitions, A its incidence matrix, M_0, M_d two markings for N . Then M_d is reachable from M_0 if and only if there is a $n \times 1$ vector x of non-negative integers such that*

$$M_d = M_0 + Ax. \quad (3)$$

The “only if” part is easy to show, as x can be the vector which tells how many times each transition fires to transform M_0 into M_d . The interesting part is the vice versa: if we can find a vector of non-negative integers x that solves equation (3), then the proof of Theorem 13 ensures the existence of a firing sequence that transforms M_0 into M_d by firing each transition t exactly x_t times. (A constructive proof for Theorem 13 can be found in [21].)

We use these considerations to prove that $\psi \circ \varphi$ is a dinatural transformation by finding a vector x that solves equation (3) for $N = \Gamma(\psi \circ \varphi)$ and M_0 and M_d as in (2). Since we want to move the tokens from the sources to the sinks and $\Gamma(\psi \circ \varphi)$ is connected (Remark 2), we ought to fire each transition at least once; on the other hand, as already observed, the acyclicity of $\Gamma(\psi \circ \varphi)$ ensures that any transition cannot fire more than once. Hence $x = [1, \dots, 1]$ is the solution we are seeking.

Proof of Theorem 8. Consider $x = [1, \dots, 1]$ of length $|T|$. A simple computation shows that, if A is the incidence matrix of $\Gamma(\psi \circ \varphi)$ and M_0 and M_d are as in (2), $M_d = M_0 + Ax$:

33:12 On Compositionality of Dinatural Transformations

it is enough to notice that A 's row corresponding to place p is made of all 0's except for exactly one 1 if p is a sink, exactly one -1 if p is a source, and exactly one 1 and one -1 if p is neither of them. Hence, by Theorem 13, M_d is reachable from M_0 , and by Proposition 12 with $M = M_0$ and $L: T \rightarrow \{A, B\}$ constantly equal to B , we obtain that $\mu(M_0, L, f) = \mu(M_0, L_d, f)$. By the arbitrariness of the morphism $f: A \rightarrow B$ we have chosen, we get the dinaturality of $\psi \circ \varphi$. \blacktriangleleft

It is not difficult to generalise Theorem 8 to the case in which $\psi \circ \varphi$ depends on more than one variable: it is enough to apply the same argument to one connected component of $\Gamma(\psi \circ \varphi)$ at a time.

► **Theorem 14.** *Let $\varphi: T \rightarrow S$ and $\psi: S \rightarrow R$ as in Definition 2, $i \in \{1, \dots, q\}$. If φ and ψ are dinatural in all their variables in, respectively, $\zeta^{-1}\{i\}$ and $\xi^{-1}\{i\}$ (with ζ and ξ given by the pushout (1)), and if the i -th connected component of $\Gamma(\psi \circ \varphi)$ is acyclic, then $\psi \circ \varphi$ is dinatural in its i -th variable.*

We conclude this section with a straightforward corollary:

► **Corollary 15.** *Let $\varphi: T \rightarrow S$ and $\psi: S \rightarrow R$ be transformations which are dinatural in all their variables. If $\Gamma(\psi \circ \varphi)$ is acyclic, then $\psi \circ \varphi$ is dinatural in all its variables.*

3 Horizontal composition

Horizontal composition of natural transformations [15] is a well known operation which is rich in interesting properties: it is associative, unitary, compatible with vertical composition. Also, it plays a crucial role in the calculus of substitution of functors and natural transformations developed by Kelly in [13]. An appropriate generalisation of this notion for dinatural transformations seems to be absent in the literature; here we propose a possible definition and prove some of its properties. First, we briefly recall the definition for the natural case.

► **Definition 16.** Consider (classical) natural transformations

$$\begin{array}{ccc} \mathbb{A} & \begin{array}{c} \xrightarrow{F} \\ \Downarrow \varphi \\ \xrightarrow{G} \end{array} & \mathbb{B} \\ & & \begin{array}{c} \xrightarrow{H} \\ \Downarrow \psi \\ \xrightarrow{K} \end{array} \\ & & \mathbb{C} \end{array}$$

The horizontal composition $\psi * \varphi: HF \rightarrow KG$ is the natural transformation whose A -th component, for $A \in \mathbb{A}$, is either leg of the following commutative square:

$$\begin{array}{ccc} HF(A) & \xrightarrow{\psi_{F(A)}} & KF(A) \\ H(\varphi_A) \downarrow & & \downarrow K(\varphi_A) \\ HG(A) & \xrightarrow{\psi_{G(A)}} & KG(A) \end{array} \quad (4)$$

Now, the commutativity of (4) is due to the naturality of ψ ; the fact that $\psi * \varphi$ is in turn a natural transformation is due to the naturality of both φ and ψ . However, in order to *define* the family of morphisms $\psi * \varphi$, all we have to do is to apply the naturality condition of ψ to the components of φ , one by one. We apply the very same idea to dinatural transformations, leading to the following preliminary definition for classical dinatural transformations.

► **Definition 17.** Let $\varphi: F \rightarrow G$ and $\psi: H \rightarrow K$ dinatural transformations of type $2 \rightarrow 1 \leftarrow 2$, where $F, G: \mathbb{A}^{\text{op}} \times \mathbb{A} \rightarrow \mathbb{B}$ and $H, K: \mathbb{B}^{\text{op}} \times \mathbb{B} \rightarrow \mathbb{C}$. The *horizontal composition* $\psi * \varphi$ is the family of morphisms

$$([\psi * \varphi]_A: H(G(A, A), F(A, A)) \rightarrow K(F(A, A), G(A, A)))_{A \in \mathbb{A}}$$

where the general component $[\psi * \varphi]_A$ is given, for any object $A \in \mathbb{A}$, by either leg of the following commutative hexagon:

$$\begin{array}{ccccc} & & \xrightarrow{\psi_{F(A,A)}} & & \\ H(\varphi_{A,1}) \nearrow & & & & \searrow K(1, \varphi_A) \\ & \cdot & & & \cdot \\ & \searrow H(1, \varphi_A) & & & \nearrow K(\varphi_{A,1}) \\ & & \xrightarrow{\psi_{G(A,A)}} & & \end{array}$$

► **Remark.** If functors F, G, H and K all factor through the projection $\mathbb{A}^{\text{op}} \times \mathbb{A} \rightarrow \mathbb{A}$ or $\mathbb{B}^{\text{op}} \times \mathbb{B} \rightarrow \mathbb{B}$, then φ and ψ are natural transformations and $\psi * \varphi$ coincides with the classical definition of horizontal composition of natural transformations.

It turns out that, as happens with classical natural transformations, the dinaturality of φ and ψ implies the dinaturality of their horizontal composition.

► **Theorem 18.** Let φ and ψ be dinatural transformations as in Definition 17. Then $\psi * \varphi$ is a dinatural transformation

$$\psi * \varphi: H(G^{\text{op}}, F) \rightarrow K(F^{\text{op}}, G)$$

of type $4 \rightarrow 1 \leftarrow 4$, where $H(G^{\text{op}}, F), K(F^{\text{op}}, G): \mathbb{A}^{[+, -, \cdot, +]} \rightarrow \mathbb{C}$ are defined on objects as

$$H(G^{\text{op}}, F)(A, B, C, D) = H(G^{\text{op}}(A, B), F(C, D))$$

$$K(F^{\text{op}}, G)(A, B, C, D) = K(F^{\text{op}}(A, B), G(C, D))$$

and similarly on morphisms.

Proof. The proof consists in showing that the diagram that asserts the dinaturality of $\psi * \varphi$ commutes: this is done in Figure 3, in the Appendix. ◀

We can now proceed with the general definition, which involves transformations of arbitrary type. As the idea behind Definition 17 is to apply the dinaturality of ψ on the general component of φ in order to define $\psi * \varphi$, if ψ is a transformation with many variables, then we have many dinaturality conditions we can apply to φ , namely one for each variable of ψ in which ψ is dinatural. Hence, the general definition will depend on the variable of ψ we want to use. For the sake of simplicity, we shall consider only the one-category case, that is when all functors in the definition involve one category \mathbb{C} , in line with our approach in Section 2; the general case follows with no substantial complications except for a much heavier notation. Indeed, when $\mathbb{A} = \mathbb{B} = \mathbb{C}$, Definition 17 is a special case of the following.

► **Definition 19.** Let $F: \mathbb{C}^\alpha \rightarrow \mathbb{C}$, $G: \mathbb{C}^\beta \rightarrow \mathbb{C}$, $H: \mathbb{C}^\gamma \rightarrow \mathbb{C}$, $K: \mathbb{C}^\delta \rightarrow \mathbb{C}$ be functors, $\varphi = (\varphi_{A_1, \dots, A_n}): F \rightarrow G$ be a transformation of type $|\alpha| \xrightarrow{\sigma} n \xleftarrow{\tau} |\beta|$ and $\psi = (\psi_{B_1, \dots, B_m}): H \rightarrow K$ of type $|\gamma| \xrightarrow{\eta} m \xleftarrow{\theta} |\delta|$ a transformation which is dinatural in its i -th variable. Denoting with $++$ the concatenation of a family of lists, let

$$H(X_1 \dots X_{|\gamma|}): \mathbb{C}^{\overset{|\gamma|}{++} \lambda^u} \rightarrow \mathbb{C}, \quad K(Y_1 \dots Y_{|\delta|}): \mathbb{C}^{\overset{|\delta|}{++} \mu^v} \rightarrow \mathbb{C}$$

33:14 On Compositionality of Dinatural Transformations

be functors, defined similarly to $H(G^{\text{op}}, F)$ and $K(F^{\text{op}}, G)$ in Theorem 18, where for all $u \in \{1, \dots, |\gamma|\}$:

$$X_u = \begin{cases} F & \eta u = i \wedge \gamma_u = + \\ G^{\text{op}} & \eta u = i \wedge \gamma_u = - \\ id_{\mathbb{C}^{\gamma_u}} & \eta u \neq i \end{cases} \quad \lambda^u = \begin{cases} \alpha & \eta u = i \wedge \gamma_u = + \\ \bar{\beta}^3 & \eta u = i \wedge \gamma_u = - \\ [\gamma_u] & \eta u \neq i \end{cases}$$

$$a_u = \begin{cases} \iota_n \sigma & \eta u = i \wedge \gamma_u = + \\ \iota_n \tau & \eta u = i \wedge \gamma_u = - \\ \iota_m \eta_{\{u\}} & \eta u \neq i \end{cases}$$

with $\iota_n: n \rightarrow (i-1) + n + (m-i)$ and $\iota_m: m \rightarrow (i-1) + n + (m-i)$ fixed injections, and for all $v \in \{1, \dots, |\delta|\}$:

$$Y_v = \begin{cases} G & \theta v = i \wedge \delta_v = + \\ F^{\text{op}} & \theta v = i \wedge \delta_v = - \\ id_{\mathbb{C}^{\delta_v}} & \theta v \neq i \end{cases} \quad \mu^v = \begin{cases} \beta & \theta v = i \wedge \delta_v = + \\ \bar{\alpha} & \theta v = i \wedge \delta_v = - \\ [\delta_v] & \theta v \neq i \end{cases}$$

$$b_v = \begin{cases} \iota_n \tau & \theta v = i \wedge \delta_v = + \\ \iota_n \sigma & \theta v = i \wedge \delta_v = - \\ \iota_m \theta_{\{v\}} & \theta v \neq i \end{cases}$$

The i -th horizontal composition $\psi \stackrel{i}{*} \varphi$ is a transformation

$$\psi \stackrel{i}{*} \varphi: H(X_1 \dots X_{|\gamma|}) \rightarrow K(Y_1 \dots Y_{|\delta|})$$

of type

$$\sum_{u=1}^{|\gamma|} |\lambda^u| \xrightarrow{[a_1 \dots a_{|\gamma|}]} (i-1) + n + (m-i) \xleftarrow{[b_1 \dots b_{|\delta|}]} \sum_{v=1}^{|\delta|} |\mu^v|$$

whose general component, $[\psi \stackrel{i}{*} \varphi]_{B_1 \dots B_{i-1}, A_1 \dots A_n, B_{i+1} \dots B_m}$, is either leg of the commutative hexagon obtained by applying the dinaturality of ψ in its i -th variable to $\varphi_{A_1, \dots, A_n}$, that is the morphism

$$H(x_1, \dots, x_{|\gamma|}); \psi_{B_1 \dots B_{i-1}, G(A_{\tau_1} \dots A_{\tau_{|\alpha|}}, B_{i+1} \dots B_m); K(y_1, \dots, y_{|\delta|})$$

where

$$x_u = \begin{cases} \varphi_{A_1, \dots, A_n} & \eta u = i \wedge \gamma_u = + \\ id_{G(A_{\tau_1} \dots A_{\tau_{|\alpha|}})} & \eta u = i \wedge \gamma_u = - \\ id_{B_{\eta u}} & \eta u \neq i \end{cases} \quad y_v = \begin{cases} id_{G(A_{\tau_1} \dots A_{\tau_{|\alpha|}})} & \theta v = i \wedge \delta_v = + \\ \varphi_{A_1, \dots, A_n} & \theta v = i \wedge \delta_v = - \\ id_{B_{\theta v}} & \theta v \neq i \end{cases}$$

► **Notation.** For the rest of this paper we shall denote the m variables of ψ as B_1, \dots, B_m and the n variables of φ as A_1, \dots, A_n , as in Definition 19. In this spirit, we shall sometimes write $\psi \stackrel{B_i}{*} \varphi$ instead of $\psi \stackrel{i}{*} \varphi$.

³ Remember that for any $\beta \in \text{List}\{+, -\}$ we denote $\bar{\beta}$ the list obtained from β by swapping the $+$'s with the $-$'s.

► Remark. $\psi \overset{i}{*} \varphi$ depends on all the variables of $\psi = (\psi_{B_1, \dots, B_m})$ where B_i has been substituted by the variables of $\varphi = (\varphi_{A_1, \dots, A_n})$.

As for the classical natural case, only the dinaturality of ψ in its i -th variable is needed to define the i -th horizontal composition of φ and ψ . It is immediate from the definitions that $\psi \overset{i}{*} \varphi$ is dinatural in all the “ B variables” (that is, those variables inherited from ψ) where also ψ is. Theorem 18 generalises to the following one, which states that if φ is dinatural in A_j , then $\psi \overset{i}{*} \varphi$ is also dinatural in A_j ; in other words, $\psi \overset{i}{*} \varphi$ is dinatural in all the “ A variables” where φ is dinatural.

► **Theorem 20.** *In the same notation as in Definition 19, if φ is dinatural in its j -th variable and ψ in its i -th one, then $\psi \overset{i}{*} \varphi$ is dinatural in its $(i - 1 + j)$ -th variable. In other words, if φ is dinatural in A_j and ψ in B_i , then $\psi \overset{B_i}{*} \varphi$ is dinatural in A_j .*

Unitarity. It is straightforward to see that horizontal composition has a unit, namely the identity (di)natural transformation of the identity functor.

► **Theorem 21.** *Let $T: \mathbb{C}^\alpha \rightarrow \mathbb{C}$ and $S: \mathbb{C}^\beta \rightarrow \mathbb{C}$ be functor, $\varphi: T \rightarrow S$ be a transformation of type $|\alpha| \xrightarrow{\sigma} k \xleftarrow{\tau} |\beta|$. Then $id_{id_{\mathbb{C}}} * \varphi = \varphi$. If φ is dinatural in its i -th variable, then also $\varphi \overset{i}{*} id_{id_{\mathbb{C}}} = \varphi$.*

Associativity. Throughout this section fix transformations $\varphi: F \rightarrow G$, $\psi: H \rightarrow K$ and $\chi: U \rightarrow V$. For sake of simplicity, denote with A_1, \dots, A_n , B_1, \dots, B_m and C_1, \dots, C_l the variables of, respectively, φ , ψ and χ . The theorem asserting associativity of horizontal composition, which we aim to prove here, is the following.

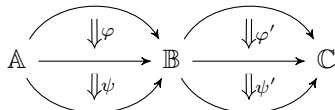
► **Theorem 22.** *Suppose ψ is dinatural in B_i and χ is dinatural in C_j . Then*

$$\chi \overset{j}{*} (\psi \overset{i}{*} \varphi) = (\chi \overset{j}{*} \psi) \overset{j-1+i}{*} \varphi \text{ or, in alternative notation, } \chi \overset{C_j}{*} (\psi \overset{B_i}{*} \varphi) = (\chi \overset{C_j}{*} \psi) \overset{B_i}{*} \varphi.$$

Proof. The proof that the two sides have the same signature is in the Appendix (Proposition 25). Regarding the single components, it is enough to consider the case in which φ , ψ and χ are all of type $2 \rightarrow 1 \leftarrow 2$, the general case follows as a consequence.

Fix then an object A in \mathbb{C} . Figure 4, in the Appendix, shows how to pass from $(\chi * \psi) * \varphi$ to $\chi * (\psi * \varphi)$ by pasting three commutative diagrams. In order to save space, we simply wrote “ $H(G, F)$ ” instead of the proper “ $H(G^{\text{op}}(A, A), F(A, A))$ ” and similarly for all the other instances of functors in the nodes of the diagram in Figure 4; we also dropped the subscript for components of φ , ψ and χ when they appear as arrows, that is we simply wrote φ instead of φ_A , since there is only one object involved and there is no risk of confusion. ◀

Incompatibility with vertical composition. It is well known that horizontal composition is compatible with the vertical one for classical natural transformations: in the following situation,



with φ, φ', ψ and ψ' natural transformations, we have:

$$(\psi' \circ \varphi') * (\psi \circ \varphi) = (\psi' * \psi) \circ (\varphi' * \varphi) \tag{†}$$

It is also well known that dinatural transformations do not vertically compose, in general; on the other hand, we have defined a notion of horizontal composition which is always possible. Are these two operations compatible, at least when vertical composition is defined?

The answer, unfortunately, is *No*, at least if by “compatible” we mean “compatible as in the natural case (\dagger)”. Indeed, consider dinatural transformations

$$\begin{array}{ccc}
 & F & \\
 \curvearrowright & \Downarrow \varphi & \curvearrowleft \\
 \mathbb{A}^{\text{op}} \times \mathbb{A} & \xrightarrow{G} & \mathbb{B} \\
 \curvearrowleft & \Downarrow \psi & \curvearrowright \\
 & H &
 \end{array}
 \qquad
 \begin{array}{ccc}
 & J & \\
 \curvearrowright & \Downarrow \varphi' & \curvearrowleft \\
 \mathbb{B}^{\text{op}} \times \mathbb{B} & \xrightarrow{K} & \mathbb{C} \\
 \curvearrowleft & \Downarrow \psi' & \curvearrowright \\
 & L &
 \end{array}$$

such that $\varphi; \psi$ and $\varphi'; \psi'$ are dinatural. Then

$$\varphi' * \varphi: J(G, F) \rightarrow K(F, G) \qquad \psi' * \psi: K(H, G) \rightarrow L(G, H)$$

which means that $\varphi' * \varphi$ and $\psi' * \psi$ are not even composable as families of morphisms, as the codomain of the former is not the domain of the latter. The problem stems from the fact that the codomain of the horizontal composition $\varphi' * \varphi$ depends on the codomain of φ' and also the domain *and* codomain of φ , which are not the same as the domain and codomain of ψ : indeed, in order to be composable, φ and ψ must share only one functor, and not both. This does not happen in the natural case, and ultimately this is due to the difference between the naturality and the dinaturality conditions for a transformation.

References

- 1 E. S. Bainbridge, P. J. Freyd, A. Scedrov, and P. J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70(1):35–64, 1990. doi:10.1016/0304-3975(90)90151-7.
- 2 R. Blute. Linear logic, coherence and dinaturality. *Theoretical Computer Science*, 115(1):3–41, 1993. doi:10.1016/0304-3975(93)90053-V.
- 3 J. Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, volume 47 of *Lecture Notes in Mathematics*, pages 1–77. Springer, Berlin, Heidelberg, 1967. doi:10.1007/BFb0074299.
- 4 E. Dubuc and Ross Street. Dinatural transformations. In S. Mac Lane, H. Applegate, M. Barr, B. Day, E. Dubuc, A. P. Phreilambud, R. Street, M. Tierney, and S. Swierczkowski, editors, *Reports of the Midwest Category Seminar IV*, volume 137 of *Lecture Notes in Mathematics*, pages 126–137. Springer, Berlin, Heidelberg, 1970. doi:10.1007/BFb0060443.
- 5 S. Eilenberg and G. M. Kelly. A generalization of the functorial calculus. *Journal of Algebra*, 3(3):366–375, 1966. doi:10.1016/0021-8693(66)90006-8.
- 6 P. J. Freyd, E. P. Robinson, and G. Rosolini. Dinaturality for free. In A. M. Pitts, M. P. Fourman, and P. T. Johnstone, editors, *Applications of Categories in Computer Science: Proceedings of the London Mathematical Society Symposium, Durham 1991*, London Mathematical Society Lecture Note Series, pages 107–118. Cambridge University Press, Cambridge, 1992. doi:10.1017/CB09780511525902.007.
- 7 J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, jan 1987. doi:10.1016/0304-3975(87)90045-4.
- 8 J.-Y. Girard, A. Scedrov, and P. J. Scott. Normal Forms and Cut-Free Proofs as Natural Transformations. In N. M. Yiannis, editor, *Logic from Computer Science*, volume 21 of *Mathematical Sciences Research Institute Publications*, pages 217–241. Springer, New York, NY, 1992. doi:10.1007/978-1-4612-2822-6_8.

- 9 A. Guglielmi and T. Gundersen. Normalisation Control in Deep Inference via Atomic Flows. *Logical Methods in Computer Science*, 4(1), 2008. doi:10.2168/LMCS-4(1:9)2008.
- 10 A. Guglielmi, T. Gundersen, and L. Straßburger. Breaking Paths in Atomic Flows for Classical Logic. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 284–293, jul 2010. doi:10.1109/LICS.2010.12.
- 11 K. Hiraishi and A. Ichikawa. A Class of Petri Nets That a Necessary and Sufficient Condition for Reachability is Obtainable. *Transactions of the Society of Instrument and Control Engineers*, 24(6):635–640, 1988. doi:10.9746/sicetr1965.24.635.
- 12 G. M. Kelly. An abstract approach to coherence. In G. M. Kelly, M. Laplaza, G. Lewis, and S. Mac Lane, editors, *Coherence in Categories*, volume 281 of *Lecture Notes in Mathematics*, pages 106–147. Springer, Berlin, Heidelberg, 1972. doi:10.1007/BFb0059557.
- 13 G. M. Kelly. Many-variable functorial calculus. I. In G. M. Kelly, M. Laplaza, G. Lewis, and S. Mac Lane, editors, *Coherence in Categories*, volume 281 of *Lecture Notes in Mathematics*, pages 66–105. Springer, Berlin, Heidelberg, 1972. doi:10.1007/BFb0059556.
- 14 G. M. Kelly and S. MacLane. Coherence in closed categories. *Journal of Pure and Applied Algebra*, 1(1):97–140, jan 1971. doi:10.1016/0022-4049(71)90013-2.
- 15 S. MacLane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2 edition, 1978. URL: //www.springer.com/gb/book/9780387984032.
- 16 P. S. Mulry. Categorical fixed point semantics. *Theoretical Computer Science*, 70(1):85–97, jan 1990. doi:10.1016/0304-3975(90)90154-A.
- 17 T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989. doi:10.1109/5.24143.
- 18 C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Mathematisches Institut der Universität Bonn, Bonn, 1962. OCLC: 258511501.
- 19 P. Selinger. A Survey of Graphical Languages for Monoidal Categories. In B. Coecke, editor, *New Structures for Physics*, volume 813 of *Lecture Notes in Physics*, pages 289–355. Springer, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-12821-9_4.
- 20 A. K. Simpson. A characterisation of the least-fixed-point operator by dinaturality. *Theoretical Computer Science*, 118(2):301–314, 1993. doi:10.1016/0304-3975(93)90112-7.
- 21 G. Stremersch and R. K. Boel. Structuring Acyclic Petri Nets for Reachability Analysis and Control. *Discrete Event Dynamic Systems*, 12(1):7–41, jan 2002. doi:10.1023/A:1013331703036.

A Appendix

Regarding Theorem 20

The proof of this theorem relies on the fact that we can reduce ourselves, without loss of generality, to Theorem 18. In order to prove that, we introduce the notion of *focalisation* of a transformation on one of its variables.

► **Definition 23.** Let $\phi = (\phi_{A_1, \dots, A_k}): T \rightarrow S$ be a transformation of type $|\alpha| \xrightarrow{\sigma} k \xleftarrow{\tau} |\beta|$ with $T: \mathbb{C}^\alpha \rightarrow \mathbb{C}$ and $S: \mathbb{C}^\beta \rightarrow \mathbb{C}$. Fix $j \in \{1, \dots, k\}$ and objects $A_1, \dots, A_{j-1}, A_{j+1}, \dots, A_k$ in \mathbb{C} . Consider functors $\bar{T}^j, \bar{S}^j: \mathbb{C}^{\text{op}} \times \mathbb{C} \rightarrow \mathbb{C}$ defined by

$$\bar{T}^j(A, B) = T(C_1, \dots, C_{|\alpha|}), \quad \bar{S}^j(A, B) = S(D_1, \dots, D_{|\beta|})$$

where

$$C_u = \begin{cases} B & \sigma u = j \wedge \alpha_u = + \\ A & \sigma u = j \wedge \alpha_u = - \\ A_{\sigma u} & \sigma u \neq j \end{cases} \quad D_v = \begin{cases} B & \tau v = j \wedge \beta_v = + \\ A & \tau v = j \wedge \beta_v = - \\ A_{\tau v} & \tau v \neq j \end{cases}$$

The focalisation of ϕ on its j -th variable is the transformation $\bar{\phi}^j: \bar{T}^j \rightarrow \bar{S}^j$ of type $2 \rightarrow 1 \leftarrow 2$ where

$$\bar{\phi}_X^j = \varphi_{A_1 \dots A_{j-1}, X, A_{j+1} \dots A_k}.$$

Sometimes we may write $\bar{\phi}^{A_j}: \bar{T}^{A_j} \rightarrow \bar{S}^{A_j}$ too, when we fix as A_1, \dots, A_k the name of the variables of ϕ .

► **Remark.** ϕ is dinatural in its j -th variable if and only if $\bar{\phi}^j$ is dinatural in its only variable for all objects $A_1, \dots, A_{j-1}, A_{j+1}, \dots, A_k$ in \mathbb{C} fixed by the focalisation of ϕ .

The $(-)^j$ construction depends on the $k-1$ objects we fix, but not to make the notation too heavy, we shall always call those (arbitrary) objects $A_1, \dots, A_{j-1}, A_{j+1}, \dots, A_n$ for $\bar{\varphi}^j$ and $B_1, \dots, B_{i-1}, B_{i+1}, \dots, B_m$ for $\bar{\psi}^i$.

► **Lemma 24.** *Let φ and ψ be transformations as in Definition 19, with ψ dinatural in its i -th variable. It is the case that $\bar{\psi}^i * \varphi$ is dinatural in its $(i-1+j)$ -th variable if and only if $\bar{\psi}^i * \bar{\varphi}^j$ is dinatural in its only variable for all objects $B_1, \dots, B_{i-1}, A_1, \dots, A_{j-1}, A_{j+1}, \dots, A_n, B_{i+1}, \dots, B_m$ in \mathbb{C} .*

Proof. Direct check that the equations between morphisms demanded by unpacking the two definitions are the same. ◀

Proof of Theorem 20. Consider transformations $\bar{\varphi}^j$ and $\bar{\psi}^i$. By Remark A, they are both dinatural in their only variable. Hence, by Theorem 18, $\bar{\psi}^i * \bar{\varphi}^j$ is dinatural and by Lemma 24 we conclude. ◀

Regarding the signature of $\chi * \psi * \varphi$

Suppose that $\varphi: F \rightarrow G$ has type $|\alpha| \xrightarrow{\sigma} n \xleftarrow{\tau} |\beta|$, $\psi: H \rightarrow K$ has type $|\gamma| \xrightarrow{\eta} m \xleftarrow{\theta} |\delta|$ and $\chi: U \rightarrow V$ has type $|\varepsilon| \xrightarrow{\pi} l \xleftarrow{\omega} |\zeta|$. First of all, notice how both $\chi^{C_j} * (\psi^{B_i} * \varphi)$ and $(\chi^{C_j} * \psi)^{B_i} * \varphi$ are families of morphisms depending on variables

$$C_1, \dots, C_{j-1}, B_1, \dots, B_{i-1}, A_1, \dots, A_n, B_{i+1}, \dots, B_m, C_{j+1}, \dots, C_l.$$

Next, we compute their domain and codomain functors. We have $\psi^{B_i} * \varphi: H(X_1, \dots, X_{|\gamma|}) \rightarrow K(Y_1, \dots, Y_{|\delta|})$ where we are using the same notations as in Definition 19. Hence

$$\chi^{C_j} * (\psi^{B_i} * \varphi): U(W_1, \dots, W_{|\varepsilon|}) \rightarrow V(Z_1, \dots, Z_{|\zeta|})$$

with $U(W_1, \dots, W_{|\varepsilon|}): \mathbb{C}_{u=1}^{|\varepsilon|} \nu^u \rightarrow \mathbb{C}$, $V(Z_1, \dots, Z_{|\zeta|}): \mathbb{C}_{u=1}^{|\zeta|} \xi^u \rightarrow \mathbb{C}$ where

$$W_u = \begin{cases} H(X_1, \dots, X_{|\gamma|}) & \pi u = j \wedge \varepsilon_u = + \\ K(Y_1, \dots, Y_{|\delta|})^{\text{op}} & \pi u = j \wedge \varepsilon_u = - \\ id_{\mathbb{C}^{\varepsilon_u}} & \pi u \neq j \end{cases} \quad \nu^u = \begin{cases} \frac{|\gamma|}{u=1} \lambda^u & \pi u = j \wedge \varepsilon_u = + \\ \frac{|\delta|}{u=1} \mu^u & \pi u = j \wedge \varepsilon_u = - \\ [\varepsilon_u] & \pi u \neq j \end{cases}$$

and similarly are defined Z_u and ξ^u (swapping $H(X_1, \dots, X_{|\gamma|})$ with $K(Y_1, \dots, Y_{|\delta|})$, ω with π , ε with ζ and so on).

On the other hand, we have

$$\chi \overset{C_j}{*} \psi: U(L_1, \dots, L_{|\varepsilon|}) \rightarrow V(M_1, \dots, M_{|\zeta|})$$

with $U(L_1, \dots, L_{|\varepsilon|}): \mathbb{C}_{u=1}^{|\varepsilon|} \rho^u \rightarrow \mathbb{C}$, $V(M_1, \dots, M_{|\zeta|}): \mathbb{C}_{u=1}^{|\zeta|} \vartheta^u \rightarrow \mathbb{C}$ where

$$L_u = \begin{cases} H & \pi u = j \wedge \varepsilon_u = + \\ K^{\text{op}} & \pi u = j \wedge \varepsilon_u = - \\ id_{\mathbb{C}^{\varepsilon_u}} & \pi u \neq j \end{cases} \quad \rho^u = \begin{cases} \gamma & \pi u = j \wedge \varepsilon_u = + \\ \bar{\delta} & \pi u = j \wedge \varepsilon_u = - \\ [\varepsilon_u] & \pi u \neq j \end{cases}$$

$$M_u = \begin{cases} K & \omega u = j \wedge \zeta_u = + \\ H^{\text{op}} & \omega u = j \wedge \zeta_u = - \\ id_{\mathbb{C}^{\zeta_u}} & \omega u \neq j \end{cases} \quad \vartheta^u = \begin{cases} \delta & \omega u = j \wedge \zeta_u = + \\ \bar{\gamma} & \omega u = j \wedge \zeta_u = - \\ [\zeta_u] & \omega u \neq j \end{cases}$$

$\chi \overset{C_j}{*} \psi$ has type $\sum_{u=1}^{|\varepsilon|} |\rho^u| \xrightarrow{[c_1, \dots, c_{|\varepsilon|}]} (j-1) + m + (l-j) \xleftarrow{[d_1, \dots, d_{|\zeta|}]} \sum_{u=1}^{|\zeta|} |\zeta^u|$ with

$$c_u = \begin{cases} \iota_m \eta & \pi u = j \wedge \varepsilon_u = + \\ \iota_m \theta & \pi u = j \wedge \varepsilon_u = - \\ \iota_l \pi_{\{i\}} & \pi u \neq j \end{cases} \quad d_u = \begin{cases} \iota_m \theta & \omega u = j \wedge \zeta_u = + \\ \iota_m \eta & \omega u = j \wedge \zeta_u = - \\ \iota_l \omega_{\{i\}} & \omega u \neq j \end{cases}$$

and $\iota_m: m \rightarrow (j-1) + m + (l-j)$, $\iota_l: l \rightarrow (j-1) + m + (l-j)$ defined as

$$\iota_m(x) = x + j - 1 \quad \iota_l(x) = \begin{cases} x & x \leq j \\ x + m - 1 & x > j \end{cases}$$

Therefore, the domain of $(\chi \overset{C_j}{*} \psi) \overset{B_i}{*} \varphi$ is $U(L_1, \dots, L_{|\varepsilon|})(P_1^1, \dots, P_{|\rho^1|}^1, \dots, P_1^{|\varepsilon|}, \dots, P_{|\rho^{|\varepsilon|}|}^{|\varepsilon|})$ while the codomain is $V(M_1, \dots, M_{|\zeta|})(Q_1^1, \dots, Q_{|\vartheta^1|}^1, \dots, Q_1^{|\zeta|}, \dots, Q_{|\vartheta^{|\zeta|}|}^{|\zeta|})$ where

$$P_v^u = \begin{cases} F & c_u(v) = j - 1 + i \wedge \rho_v^u = + \\ G^{\text{op}} & c_u(v) = j - 1 + i \wedge \rho_v^u = - \\ id_{\mathbb{C}^{\rho_v^u}} & c_u(v) \neq j - 1 + i \end{cases}$$

and similarly Q_v^u . Denoting the domain of $(\chi \overset{C_j}{*} \psi) \overset{B_i}{*} \varphi$ as $U(L(P))$, we have

$$U(L(P)): \mathbb{C}_{u=1}^{|\varepsilon|} \left(\prod_{v=1}^{|\rho^u|} w_v^u \right) \rightarrow \mathbb{C}$$

where

$$w_v^u = \begin{cases} \alpha & c_u(v) = j - 1 + i \wedge \rho_v^u = + \\ \bar{\beta} & c_u(v) = j - 1 + i \wedge \rho_v^u = - \\ [\rho_v^u] & c_u(v) \neq j - 1 + i \end{cases}$$

33:20 On Compositionality of Dinatural Transformations

► **Proposition 25.** *Transformations $\chi \overset{C_j}{*} (\psi \overset{B_i}{*} \varphi)$ and $(\chi \overset{C_j}{*} \psi) \overset{B_i}{*} \varphi$ have the same domain, codomain, and type.*

Proof. One can prove that $\prod_{u=1}^{|\varepsilon|} \left(\prod_{v=1}^{|\rho^u|} w_v^u \right) = \prod_{u=1}^{|\varepsilon|} \nu^u$ by showing that $\prod_{v=1}^{|\rho^u|} w_v^u = \nu^u$ for all $u \in \{1, \dots, |\varepsilon|\}$, analysing each of the three cases for ηu that define ν^u .

Next, we have that

$$U(L(P)) = U\left(L_1(P_1^1, \dots, P_{|\rho^1|}^1), \dots, L_{|\varepsilon|}(P_1^{|\varepsilon|}, \dots, P_{|\rho^{|\varepsilon|}|}^{|\varepsilon|})\right)$$

and by showing that $W_u = L_u(P_1^u, \dots, P_{|\rho^u|}^u)$ for all $u \in \{1, \dots, |\varepsilon|\}$, one proves that $\chi \overset{C_j}{*} (\psi \overset{B_i}{*} \varphi)$ and $(\chi \overset{C_j}{*} \psi) \overset{B_i}{*} \varphi$ have the same domain; an analogous procedure shows that they also share the same codomain.

Finally, we briefly analyse only the left hand sides of the types of $\chi \overset{C_j}{*} (\psi \overset{B_i}{*} \varphi)$ and $(\chi \overset{C_j}{*} \psi) \overset{B_i}{*} \varphi$; the right hand sides are handled analogously. For $\chi \overset{C_j}{*} (\psi \overset{B_i}{*} \varphi)$ we have

$$\sum_{u=1}^{|\varepsilon|} |\nu^u| \xrightarrow{[r_1, \dots, r_{|\varepsilon|}]} (j-1) + [(i-1) + k + (l-i)] + (m-j)$$

with

$$r_u = \begin{cases} ((\cdot) + j - 1) \circ [a_1, \dots, a_{|\gamma|}] & \eta u = j \wedge \varepsilon_u = + \\ ((\cdot) + j - 1) \circ [b_1, \dots, b_{|\delta|}] & \eta u = j \wedge \varepsilon_u = - \\ \iota_m \text{id}_{\mathbb{C}^{\eta u}} & \eta u \neq j \end{cases}$$

where function $((\cdot) + j - 1)$ merges $(i-1) + k + (l-i)$ into $N = (j-1) + [(i-1) + k + (l-i)] + (m-j)$, by adding $j - 1$ to its argument, and ι_m into N . For $(\chi \overset{C_j}{*} \psi) \overset{B_i}{*} \varphi$, which is the same as $(\chi \overset{j}{*} \psi) \overset{-1+i}{*} \varphi$, we have

$$\sum_{u=1}^{|\varepsilon|} \sum_{v=1}^{|\rho^u|} |w_v^u| \xrightarrow{[s_1^1, \dots, s_{|\rho^1|}^1, \dots, s_1^{|\varepsilon|}, \dots, s_{|\rho^{|\varepsilon|}|}^{|\varepsilon|}]} M$$

where $M = (j-1 + i-1) + k + [(j-1 + m + l - j) - (j-1 + i)] = N$ and

$$s_v^u = \begin{cases} ((\cdot) + j - 1 + i - 1) \circ \sigma & c_u(v) = j - 1 + i \wedge \rho_v^u = + \\ ((\cdot) + j - 1 + i - 1) \circ \tau & c_u(v) = j - 1 + i \wedge \rho_v^u = - \\ \iota_m d_{u \setminus \{v\}} & c_u(v) \neq j - 1 + i \end{cases}$$

Notice that here we are asserting an equality between natural numbers; in other words, we are just writing, in two different ways, the same set. Checking that $r_u = [s_1^u, \dots, s_{|\rho^u|}^u]$ and noticing that functions $[\dots r_u \dots]$ and $[\dots s_v^u \dots]$ coincide on every elements of their domain, we conclude. ◀

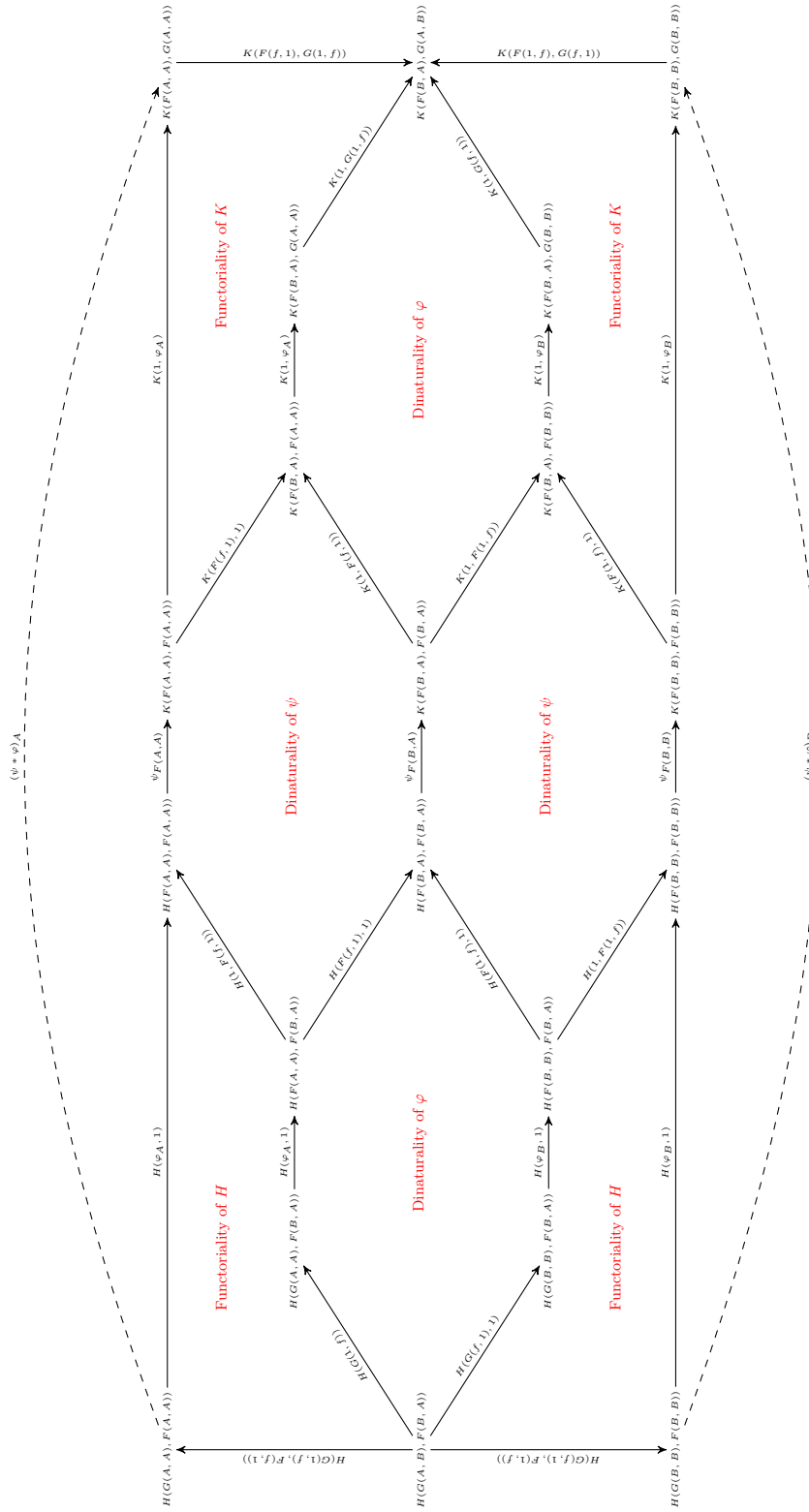


Figure 3 Proof of Theorem 18: dinaturality of horizontal composition in the classical case. Here $f : A \rightarrow B$.

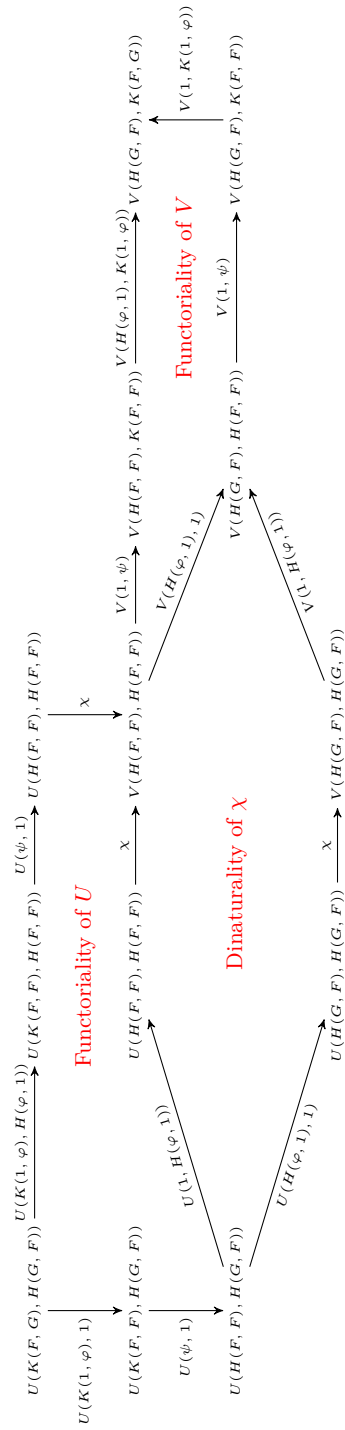


Figure 4 Associativity of horizontal composition in the classical case. The upper leg is $(\chi * \psi) * \varphi$, whereas the lower one is $\chi * (\psi * \varphi)$.

Synthesizing Optimally Resilient Controllers

Daniel Neider

Max Planck Institute for Software Systems, 67663 Kaiserslautern, Germany
neider@mpi-sws.org

Alexander Weinert¹

Reactive Systems Group, Saarland University, 66123 Saarbrücken, Germany
weinert@react.uni-saarland.de

Martin Zimmermann²

Reactive Systems Group, Saarland University, 66123 Saarbrücken, Germany
zimmermann@react.uni-saarland.de

Abstract

Recently, Dallal, Neider, and Tabuada studied a generalization of the classical game-theoretic model used in program synthesis, which additionally accounts for unmodeled intermittent disturbances. In this extended framework, one is interested in computing optimally resilient strategies, i.e., strategies that are resilient against as many disturbances as possible. Dallal, Neider, and Tabuada showed how to compute such strategies for safety specifications.

In this work, we compute optimally resilient strategies for a much wider range of winning conditions and show that they do not require more memory than winning strategies in the classical model. Our algorithms only have a polynomial overhead in comparison to the ones computing winning strategies. In particular, for parity conditions optimally resilient strategies are positional and can be computed in quasipolynomial time.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Controller Synthesis, Infinite Games, Resilient Strategies, Disturbances

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.34

Related Version Full version available online [18], <https://arxiv.org/abs/1709.04854>.

1 Introduction

Reactive synthesis is an exciting and promising approach to solving a crucial problem, whose importance is ever-increasing due to ubiquitous deployment of embedded systems: obtaining correct and verified controllers for safety-critical systems. Instead of an engineer programming a controller by hand and then verifying it against a formal specification, synthesis automatically constructs a correct-by-construction controller from the given specification (or reports that no such controller exists).

Typically, reactive synthesis is modeled as a two-player zero-sum game on a finite graph that is played between the system, which seeks to satisfy the specification, and its environment, which seeks to violate it. Although this model is well understood, there are still multiple obstacles to overcome before synthesis can be realistically applied in practice. These obstacles include not only the high computational complexity of the problem, but also more fundamental ones. Among the most prohibitive issues in this regard is the need

¹ Supported by the Saarbrücken Graduate School of Computer Science.

² Supported by the project “TriCS” (ZI 1516/1-1) of the German Research Foundation (DFG).



for a complete model of the interaction between the system and its environment, including an accurate model of the environment, the actions available to both players, as well as the effects of these actions.

This modeling task often places an insurmountable burden on engineers as the environments in which real-life controllers are intended to operate tend to be highly complex or not fully known at design time. Also, when a controller is deployed in the real world, a common source of errors is a mismatch between the controller's intended result of an action and the actual result. Such situations arise, e.g., in the presence of disturbances, when the effect of an action is not precisely known, or when the intended control action of the controller cannot be executed, e.g., when an actuator malfunctions. By a slight abuse of notation from control theory, such errors are subsumed under the generic term *disturbance* (cf. [10]).

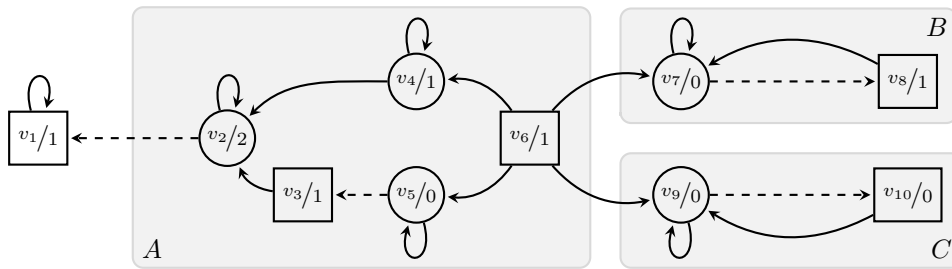
To obtain controllers that can handle disturbances, one has to yield control over their occurrence to the environment. However, due to the antagonistic setting of the two-player zero-sum game, this would allow the environment to violate the specification by causing disturbances at will. Overcoming this requires the engineer to develop a realistic disturbance model, which is a highly complex task, as such disturbances are assumed to be rare events. Also, incorporating such a model into the game leads to a severe blowup in the size of the game, which can lead to intractability due to the high computational complexity of synthesis.

To overcome these fundamental difficulties, Dallal, Neider, and Tabuada [10] proposed a conceptually simple, yet powerful extension of infinite games termed “games with unmodeled intermittent disturbances”. Such games are played similarly to classical infinite games: two players, called Player 0 and Player 1, move a token through a finite graph, whose vertices are partitioned into vertices under the control of Player 0 and Player 1, respectively; the winner is declared based on a condition on the resulting play. In contrast to classical games, however, the graph is augmented with additional *disturbance edges* that originate in vertices of Player 0 and may lead to any other vertex. Moreover, the mechanics of how Player 0 moves is modified: whenever she moves the token, her move might be overridden, and the token instead moves along a disturbance edge. This change in outcome implicitly models the occurrence of a disturbance – the intended result of the controller and the actual result differ – but it is not considered to be antagonistic. Instead, the occurrence of a disturbance is treated as a rare event without any assumptions on frequency, distribution, etc. This approach very naturally models the kind of disturbances typically occurring in control engineering [10].

As a non-technical example, consider a scenario with three siblings, Alice, Bob, and Charlie, and their father, Donald. He repeatedly asks Alice to fetch water from a well using a jug made of clay. Alice has three ways to fulfill that task: she may get the water herself or she may delegate it to either Bob or Charlie. In a simple model, the outcome of these strategies is identical: Donald's request for water is fulfilled. This is, however, unrealistic, as this model ignores the various ways that the execution of the strategies may go wrong. By modeling the situation as a game with disturbances, we obtain a more realistic model.

If Alice gets the jug herself, no disturbance can occur: she controls the outcome completely. If she delegates the task to Bob, the older of her brothers, Donald may get angry with her for not fulfilling her duties herself, which should not happen infinitely often. Finally, if she delegates the task to her younger brother Charlie, he might drop and break the jug, which would be disastrous for Alice.

These strategies can withstand different numbers of disturbances: the first strategy does not offer any possibility for disturbances, while infinitely many (a single) disturbance cause Alice to lose when using the second (the third) strategy. This model captures the intuition about Donald's and Charlie's behavior: both events occur non-antagonistically and their frequency is unknown.



■ **Figure 1** A (max-) parity game with disturbances. Disturbance edges are drawn as dashed arrows. Vertices are labeled with both a name and a color. Vertices under control of Player 0 are drawn as circles, while vertices under control of Player 1 are drawn as rectangles.

This non-antagonistic nature of disturbances is different from existing approaches in the literature and causes many interesting phenomena that do not occur in the classical theory of infinite graph-based games. Some of these already manifest themselves in the parity game shown in Figure 1, in which vertices are labeled with non-negative integers, so-called colors, and Player 0 wins if the highest color seen infinitely often is even. Consider, for instance, vertex v_2 . In the classical setting without disturbances, Player 0 wins every play reaching v_2 by simply looping in this vertex forever (since the highest color seen infinitely often is even). However, this is no longer true in the presence of disturbances: a disturbance in v_2 causes a play to proceed to vertex v_1 , from which Player 0 can no longer win. In vertex v_7 , Player 0 is in a similar, yet less severe situation: she wins every play with finitely many disturbances but loses if infinitely many disturbances occur. Finally, vertex v_9 falls into a third category: from this vertex, Player 0 wins every play even if infinitely many disturbances occur. In fact, disturbances partition the set of vertices from which Player 0 can guarantee to win into three disjoint regions (indicated as shaded boxes in Figure 1): (A) vertices from which she can win if at most a fixed finite number of disturbances occur, (B) vertices from which she can win if any finite number of disturbances occurs but not if infinitely many occur, and (C) vertices from which she can win even if infinitely many disturbances occur.

The observation above gives rise to a question that is both theoretically interesting and practically important: if Player 0 can tolerate different numbers of disturbances from different vertices, how should she play to be *resilient*³ to as many disturbances as possible, i.e., to tolerate as many disturbances as possible but still win? Put slightly differently, disturbances induce an order on the space of winning strategies (“a winning strategy is better if it is more resilient”), and the natural problem is to compute optimally resilient winning strategies, yielding optimally resilient controllers. Note that this is in contrast to the classical theory of infinite games, where the space of winning strategies is unstructured.

Dallal, Neider, and Tabuada [10] have solved the problem of computing optimally resilient winning strategies for safety games. Their approach exploits the existence of maximally permissive winning strategies in safety games [2], which allows Player 0 to avoid “harmful” disturbance edges during a play. In games with more expressive winning conditions, however, this is no longer possible, as witnessed by vertex v_4 in the example of Figure 1: although Player 0 can avoid a disturbance edge by looping in v_4 forever, she needs to move to v_2 eventually in order to see an even color (otherwise she loses), thereby risking to lose if a

³ We have deliberately chosen the term *resilience* so as to avoid confusion with the already highly ambiguous notions of *robustness* and *fault tolerance*.

disturbance occurs. In fact, the problem of constructing optimally resilient winning strategies for games other than safety games is still open. In this work, we solve this problem for a large class of infinite games, including parity games. In detail, our contributions are as follows.

We study the concept of *resilience*, which captures for each vertex how many disturbances need to occur for Player 0 to lose. This generalizes the notion of determinacy and allows us to derive optimally resilient winning strategies.

Our main result is an algorithm for computing the resilience of vertices and optimally resilient winning strategies. This algorithm requires the game to have a prefix-independent winning condition, to be determined, and all its subgames to be (classically) solvable. The latter two conditions are necessary, as resilience generalizes determinacy and computing optimally resilient strategies generalizes solving games. The algorithm uses solvers for the underlying game without disturbances as a subroutine, which it invokes a linear number of times on various subgames. For many winning conditions, the time complexity of our algorithm thus falls into the same complexity class as solving the original game without disturbances, e.g., we obtain a quasipolynomial algorithm for parity games with disturbances, which matches the currently best known upper bound for classical parity games.

Stated differently, if the three assumptions above are satisfied by a winning condition, then computing the resilience and optimally resilient strategies is not harder than determining winning regions and winning strategies (ignoring a polynomial overhead).

Our algorithm requires the winning condition of the game to be prefix-independent. We also show how to overcome this restriction by generalizing the classical notion of game reductions to the setting of games with disturbances. As a consequence, via reductions our algorithm can be applied to prefix-dependent winning conditions. Hence, we have generalized the original result of Dallal, Neider, and Tabuada from safety games to all games which are algorithmically solvable, in particular all ω -regular games.

Finally, we discuss further phenomena that arise in the presence of disturbances. Amongst others, we illustrate how the additional goal of avoiding disturbances whenever possible affects the memory requirements of strategies. Moreover, we raise the question of how benevolent disturbances can be leveraged to recover from losing a play. However, an in-depth investigation of these phenomena is outside the scope of this paper and left for future work.

Proofs omitted due to space restrictions are in the full version [18].

Related Work. The notion of *unmodeled intermittent disturbances* in infinite games has recently been formulated by Dallal, Neider, and Tabuada [10]. In that work, the authors also present an algorithm for computing optimally resilient strategies for safety games with disturbances, which is an extension of the classical attractor computation [14]. Due to the relatively simple nature of such games, however, this algorithm cannot easily be extended to handle more expressive winning conditions, and the approach presented in this work relies on fundamentally different ideas.

For the special case of parity games, we can also characterize vertices of finite resilience (presented in Subsection 3.1) by a reduction to finding optimal strategies in energy parity games [9], which yields the same complexity as our algorithm (though such a reduction would not distinguish between vertices of type B and type C). Also, it is unclear if and how this reduction can be extended to other winning conditions and if custom-made solutions would be required for each new class of game. By contrast, our refinement-based approach works for any class of infinite games that satisfies the mild assumptions discussed in Section 4.

Resilience is not a novel concept in the context of reactive systems synthesis. It appears, for instance, in the work by Topcu et al. [21] as well as Ehlers and Topcu [12]. A notion of resilience that is very similar to the one considered here has been proposed by Huang

et al. [15], where the game graph is augmented with so-called “error edges”. However, this setting differs from the one studied in this work in various aspects. Firstly, Huang et al. work in the framework of concurrent games and model errors as being under the control of Player 1. This contrasts to the setting considered here, in which the players play in alternation and disturbances are seen as rare events rather than antagonistic to Player 0. Secondly, Huang et al. restrict themselves to safety games, whereas we consider a much broader class of infinite games. Finally, Huang et al. compute resilient strategies with respect to a fixed parameter k , thus requiring to repeat the computation for various values of k to find optimally resilient strategies. In contrast, our approach computes an optimal strategy in a single run. Hence, they consider a more general model of interaction, but only a simple winning condition, while the notion of disturbances considered here is incomparable to theirs.

Related to resilience are various notions of *fault tolerance* [1, 7, 11, 13] and *robustness* [3, 4, 5, 6, 16, 19, 20]. For instance, Brihaye et al. [7] consider quantitative games under failures, which are a generalization of sabotage games [22]. The main difference to our setting is that Brihaye et al. consider failures – embodied by a saboteur player – as antagonistic, whereas we consider disturbances as a non-antagonistic events. Moreover, solving a parity game while maintaining a cost associated with the sabotage semantics below a given threshold is EXPTIME-complete, whereas our approach computes optimally resilient controllers for parity conditions in quasipolynomial time.

Besides fault tolerance, robustness in the area of reactive controller synthesis has also attracted considerable interest in the recent years, typically in settings with specifications of the form $\varphi \Rightarrow \psi$ stating that the controller needs to fulfill the guarantee ψ if the environment satisfies the assumption φ . A prominent example of such work is that of Bloem et al. [3], in which the authors understand robustness as the property that “if assumptions are violated temporarily, the system is required to recover to normal operation with as few errors as possible” and consider the synthesis of robust controllers for the GR(1) fragment of Linear Temporal Logic [6]. Other examples include quantitative synthesis [4], where robustness is defined in terms of payoffs, and the synthesis of robust controllers for cyber-physical systems [16, 19]. For a more in-depth discussion of related notions of resilience and robustness in reactive synthesis, we refer the interested reader to Dallal, Neider, and Tabuada’s section on related work [10, Section I]. Moreover, a survey of a large body of work dealing with robustness in reactive synthesis has been presented by Bloem et al. [5].

2 Preliminaries

For notational convenience, we employ some ordinal notation à la von Neumann: the non-negative integers are defined inductively as $0 = \emptyset$ and $n + 1 = n \cup \{n\}$. Now, the first limit ordinal is $\omega = \{0, 1, 2, \dots\}$, the set of the non-negative integers. The next two successor ordinals are $\omega + 1 = \omega \cup \{\omega\}$ and $\omega + 2 = \omega + 1 \cup \{\omega + 1\}$. These ordinals are ordered by set inclusion, i.e., we have $0 < 1 < 2 < \dots < \omega < \omega + 1 < \omega + 2$. For convenience of notation, we also denote the cardinality of ω by ω .

Infinite Games with Disturbances. An arena (with unmodeled disturbances) $\mathcal{A} = (V, V_0, V_1, E, D)$ consists of a finite directed graph (V, E) , a partition $\{V_0, V_1\}$ of V into the set of vertices V_0 of Player 0 (denoted by circles) and the set of vertices of Player 1 (denoted by squares), and a set $D \subseteq V_0 \times V$ of *disturbance edges* (denoted by dashed arrows). Note that only vertices of Player 0 have outgoing disturbance edges. We require that every vertex $v \in V$ has a successor v' with $(v, v') \in E$ to avoid finite plays.

A *play* in \mathcal{A} is an infinite sequence $\rho = (v_0, b_0)(v_1, b_1)(v_2, b_2) \cdots \in (V \times \{0, 1\})^\omega$ such that $b_0 = 0$ and for all $j > 0$: $b_j = 0$ implies $(v_{j-1}, v_j) \in E$, and $b_j = 1$ implies $(v_{j-1}, v_j) \in D$. Hence, the additional bits b_j for $j > 0$ denote whether a standard or a disturbance edge has been taken to move from v_{j-1} to v_j . We say ρ starts in v_0 . A play prefix $(v_0, b_0) \cdots (v_j, b_j)$ is defined similarly and ends in v_j . The number of disturbances in a play $\rho = (v_0, b_0)(v_1, b_1)(v_2, b_2) \cdots$ is $\#_D(\rho) = |\{j \in \omega \mid b_j = 1\}|$, which is either some $k \in \omega$ (if there are finitely many disturbances, namely k) or it is equal to ω (if there are infinitely many). A play ρ is *disturbance-free*, if $\#_D(\rho) = 0$.

A *game (with unmodeled disturbances)*, denoted by $\mathcal{G} = (\mathcal{A}, \text{Win})$, consists of an arena $\mathcal{A} = (V, V_0, V_1, E, D)$ and a winning condition $\text{Win} \subseteq V^\omega$. A play $\rho = (v_0, b_0)(v_1, b_1)(v_2, b_2) \cdots$ is winning for Player 0, if $v_0 v_1 v_2 \cdots \in \text{Win}$, otherwise it is winning for Player 1. Hence, winning is oblivious to occurrences of disturbances. A winning condition Win is prefix-independent if for all $\rho \in V^\omega$ and all $w \in V^*$ we have $\rho \in \text{Win}$ if and only if $w\rho \in \text{Win}$.

In examples, we often use the parity condition, the canonical ω -regular winning condition. Let $\Omega: V \rightarrow \omega$ be a coloring of a set V of vertices. The (max-) parity condition $\text{Parity}(\Omega) = \{v_0 v_1 v_2 \cdots \in V^\omega \mid \limsup \Omega(v_0)\Omega(v_1)\Omega(v_2) \cdots \text{ is even}\}$ requires the maximal color occurring infinitely often during a play to be even. A game $(\mathcal{A}, \text{Win})$ is a parity game, if $\text{Win} = \text{Parity}(\Omega)$ for some coloring Ω of the vertices of \mathcal{A} . In figures, we label a vertex v with color c by v/c .

In our proofs we make use of the safety condition $\text{Safety}(U) = \{v_0 v_1 v_2 \cdots \in V^\omega \mid v_j \notin U \text{ for every } j \in \omega\}$ for a given set $U \subseteq V$ of unsafe vertices. It requires Player 0 to only visit safe vertices, i.e., Player 1 wins a play if it visits at least one unsafe vertex.

A strategy for Player $i \in \{0, 1\}$ is a function $\sigma: V^*V_i \rightarrow V$ such that $(v_j, \sigma(v_0 \cdots v_j)) \in E$ holds for every $v_0 \cdots v_j \in V^*V_i$. A play $(v_0, b_0)(v_1, b_1)(v_2, b_2) \cdots$ is consistent with σ , if $v_{j+1} = \sigma(v_0 \cdots v_j)$ for every j with $v_j \in V_i$ and $b_{j+1} = 0$, i.e., if the next vertex is the one prescribed by the strategy unless a disturbance edge is used. A strategy σ is positional, if $\sigma(v_0 \cdots v_j) = \sigma(v_j)$ for all $v_0 \cdots v_j \in V^*V_i$.

► **Remark.** Note that a strategy σ does not have access to the bits indicating whether a disturbance occurred or not. However, this is not a restriction: let $(v_0, b_0)(v_1, b_1)(v_2, b_2) \cdots$ be a play with $b_j = 1$ for some $j > 0$. We say that this disturbance is consequential (w.r.t. σ), if $v_j \neq \sigma(v_0 \cdots v_{j-1})$, i.e., if the disturbance transition (v_{j-1}, v_j) traversed by the play did not lead to the vertex the strategy prescribed. Such consequential disturbances can be detected by comparing the actual vertex v_j to σ 's output $\sigma(v_0 \cdots v_{j-1})$. On the other hand, inconsequential disturbances will just be ignored. In particular, the number of consequential disturbances is always at most the number of disturbances.

Infinite Games without Disturbances. We characterize the classical notion of infinite games, i.e., those without disturbances, (see, e.g., [14]) as a special case of games with disturbances. Let \mathcal{G} be a game with vertex set V . A strategy σ for Player i in \mathcal{G} is a winning strategy for her from $v \in V$, if every disturbance-free play that starts in v and that is consistent with σ is winning for Player i .

The winning region $\mathcal{W}_i(\mathcal{G})$ of Player i in \mathcal{G} contains those vertices $v \in V$ from which Player i has a winning strategy. Thus, the winning regions of \mathcal{G} are independent of the disturbance edges, i.e., we obtain the classical notion of infinite games. We say that Player i wins \mathcal{G} from v , if $v \in \mathcal{W}_i(\mathcal{G})$. Solving a game amounts to determining its winning regions. Note that every game has disjoint winning regions. In contrast, a game is determined, if every vertex is in either winning region.

Resilient Strategies. Let \mathcal{G} be a game with vertex set V and let $\alpha \in \omega + 2$. A strategy σ for Player 0 in \mathcal{G} is α -resilient from $v \in V$ if every play ρ that starts in v , that is consistent with σ , and with $\#_D(\rho) < \alpha$, is winning for Player 0. Thus, a k -resilient strategy with $k \in \omega$ is winning even under at most $k - 1$ disturbances, an ω -resilient strategy is winning even under any finite number of disturbances, and an $(\omega + 1)$ -resilient strategy is winning even under infinitely many disturbances. Note that *every* strategy is 0-resilient, as no play has less than zero disturbances. Also, a strategy is 1-resilient from v if and only if it is winning for Player 0 from v . We define the resilience of a vertex v of \mathcal{G} as

$$r_{\mathcal{G}}(v) = \sup\{\alpha \in \omega + 2 \mid \text{Player 0 has an } \alpha\text{-resilient strategy for } \mathcal{G} \text{ from } v\}.$$

Note that the definition is not antagonistic, i.e., it is not defined via strategies of Player 1. Nevertheless, due to the remarks above, resilient strategies generalize winning strategies.

► **Remark.** Let \mathcal{G} be a determined game. Then, $r_{\mathcal{G}}(v) > 0$ if and only if $v \in \mathcal{W}_0(\mathcal{G})$.

A strategy σ is *optimally resilient*, if it is $r_{\mathcal{G}}(v)$ -resilient from every vertex v . Every such strategy is a *uniform* winning strategy for Player 0, i.e., a strategy that is winning from every vertex in her winning region. Hence, positional optimally resilient strategies can only exist in games which have uniform positional winning strategies for Player 0. Our goal is to determine the mapping $r_{\mathcal{G}}$ and to compute an optimally resilient strategy.

3 Computing Optimally Resilient Strategies

To compute optimally resilient strategies, we first characterize the vertices of finite resilience in Subsection 3.1. All other vertices either have resilience ω or $\omega + 1$. To distinguish between these possibilities, we show how to determine the vertices with resilience $\omega + 1$ in Subsection 3.2. In Subsection 3.3, we show how to compute optimally resilient strategies using the results of the first two subsections.

3.1 Characterizing Vertices of Finite Resilience

Our goal in this subsection is to characterize vertices with finite resilience in a game with prefix-independent winning condition, i.e., those vertices from which Player 0 can win even under $k - 1$ disturbances, but not under k disturbances, for some $k \in \omega$.

To illustrate our approach, consider the parity game in Figure 1 (on Page 3). The winning region of Player 1 only contains the vertex v_1 . Thus, by Remark 2, v_1 is the only vertex with resilience zero, every other vertex has a larger resilience.

Now, consider the vertex v_2 , which has a disturbance edge leading into the winning region of Player 1. Due to this edge, v_2 has resilience one. The unique disturbance-free play starting in v_1 is consistent with every strategy for Player 0 and violates the winning condition. Due to prefix-independence, prepending the disturbance edge does not change the winner and consistency with every strategy for Player 0. Hence, this play witnesses that v_2 has resilience at most one, while v_2 being in Player 0's winning region yields the matching lower bound. However, v_2 is the only vertex to which this reasoning applies. Now, consider v_3 : from here, Player 1 can force a play to visit v_2 using a standard edge. Thus, v_3 has resilience one as well. Again, this is the only vertex to which this reasoning is applicable.

In particular, from v_4 Player 0 can avoid reaching the vertices for which we have determined the resilience by using the self loop. However, this comes at a steep price for her: doing so results in a losing play, as the color of v_4 is odd. Thus, if she wants to have a chance at winning, she has to take a risk by moving to v_2 , from which she has a 1-resilient strategy,

i.e., one that is winning if no more disturbances occur. For this reason, v_4 has resilience one as well. The same reasoning applies to v_6 : Player 1 can force the play to v_4 and from there Player 0 has to take a risk by moving to v_2 .

The vertices v_3 , v_4 , and v_6 share the property that Player 1 can either enforce a play violating the winning condition or reach a vertex with already determined finite resilience. These three vertices are the only ones currently satisfying this property. They all have resilience one since Player 1 can enforce to reach a vertex of resilience one, but he cannot enforce reaching a vertex of resilience zero. Now, we can also determine the resilience of v_5 : The disturbance edge from v_5 to v_3 witnesses it being two.

Afterwards, these two arguments no longer apply to new vertices: no disturbance edge leads from a $v \in \{v_7, \dots, v_{10}\}$ to some vertex whose resilience is already determined and Player 0 has a winning strategy from each v that additionally avoids vertices whose resilience is already determined. Thus, our reasoning cannot determine their resilience. This is consistent with our goal, as all four vertices have non-finite resilience: v_7 and v_8 have resilience ω and v_9 and v_{10} have resilience $\omega + 1$. Our reasoning here cannot distinguish these two values. We solve this problem in Subsection 3.2.

We now formalize the reasoning sketched above: starting from the vertices in Player 1's winning region having resilience zero, we use a so-called disturbance update and risk update to determine all vertices of finite resilience. A disturbance update computes the resilience of vertices having a disturbance edge to a vertex whose resilience is already known (such as vertices v_2 and v_5 in the example of Figure 1). A risk update, on the other hand, determines the resilience of vertices from which either Player 1 can force a visit to a vertex with known resilience (such as vertices v_3 and v_6) or Player 0 needs to move to such a vertex in order to avoid losing (e.g., vertex v_4). To simplify our proofs, we describe both as monotone operators updating partial rankings mapping vertices to ω , which might update already defined values. We show that applying these updates in alternation eventually yields a stable ranking that indeed characterizes the vertices of finite resilience.

Throughout this section, we fix a game $\mathcal{G} = (\mathcal{A}, \text{Win})$ with $\mathcal{A} = (V, V_0, V_1, E, D)$ and prefix-independent $\text{Win} \subseteq V^\omega$ satisfying the following condition: the game $(\mathcal{A}, \text{Win} \cap \text{Safety}(U))$ is determined for every $U \subseteq V$. We discuss this requirement in Section 4.

A ranking for \mathcal{G} is a partial mapping $r: V \dashrightarrow \omega$. The domain of r is denoted by $\text{dom}(r)$, its image by $\text{im}(r)$. Let r and r' be two rankings. We say that r' refines r if $\text{dom}(r') \supseteq \text{dom}(r)$ and if $r'(v) \leq r(v)$ for all $v \in \text{dom}(r)$. A ranking r is sound, if we have $r(v) = 0$ if and only if $v \in \mathcal{W}_1(\mathcal{G})$ (cf. Remark 2).

Let r be a ranking for \mathcal{G} . We define the ranking r' as

$$r'(v) = \min(\{r(v)\} \cup \{r(v') + 1 \mid v' \in \text{dom}(r) \text{ and } (v, v') \in D\}),$$

where $\{r(v)\} = \emptyset$ if $v \notin \text{dom}(r)$, and $\min \emptyset$ is undefined (causing $r'(v)$ to be undefined). We call r' the *disturbance update* of r .

► **Lemma 1.** *The disturbance update r' of a sound ranking r is sound and refines r .*

Again, let r be a ranking for \mathcal{G} . For every $k \in \text{im}(r)$ let $A_k = \mathcal{W}_1(\mathcal{A}, \text{Win} \cap \text{Safety}(\{v \in \text{dom}(r) \mid r(v) \leq k\}))$ the winning region of Player 1 in the game where he either wins by reaching a vertex v with $r(v) \leq k$ or by violating the winning condition. Now, define $r'(v) = \min\{k \mid v \in A_k\}$, where $\min \emptyset$ is again undefined. We call r' the *risk update* of r .

► **Lemma 2.** *The risk update r' of a sound ranking r is sound and refines r .*

Let r_0 be the unique sound ranking with domain $\mathcal{W}_1(\mathcal{G})$, i.e., r_0 maps exactly the vertices in Player 1's winning region to zero. Starting with r_0 , we inductively define a sequence of rankings $(r_j)_{j \in \omega}$ such that r_j for an odd (even) $j > 0$ is the disturbance (risk) update of r_{j-1} , i.e., we alternate between disturbance and risk updates.

Due to refinement, the r_j eventually stabilize, i.e., there is some j_0 such that $r_j = r_{j_0}$ for all $j \geq j_0$. Define $r^* = r_{j_0}$. Due to r_0 being sound and by Lemma 1 and Lemma 2, each r_j , and r^* in particular, is sound. If $v \in \text{dom}(r^*)$, let j_v be the minimal j with $v \in \text{dom}(r_j)$; otherwise, j_v is undefined.

► **Lemma 3.** *If $v \in \text{dom}(r^*)$, then $r_{j_v}(v) = r_j(v)$ for all $j \geq j_v$.*

Lemma 3 implies that an algorithm computing the r_j does not need to implement the definition of the two updates as presented above, but can be optimized by taking into account that a rank is never updated once set. However, for the proofs below, the definition presented above is more expedient, as it gives stronger preconditions to rely on, e.g., Lemma 1 and 2 only hold for the definition presented above.

Also, from the proof of Lemma 3, we obtain an upper bound on the maximal rank of r^* . This in turn implies that the r_j stabilize quickly, as $r_j = r_{j+1} = r_{j+2}$ implies $r_j = r^*$.

► **Corollary 4.** *We have $\text{im}(r^*) = \{0, 1, \dots, n\}$ for some $n < |V|$ and $r^* = r_{2|V|}$.*

The main result of this section shows that r^* characterizes the resilience of vertices of finite resilience.

► **Lemma 5.** *Let r^* be defined for \mathcal{G} as above, and let $v \in V$.*

1. *If $v \in \text{dom}(r^*)$, then $r_{\mathcal{G}}(v) = r^*(v)$.*
2. *If $v \notin \text{dom}(r^*)$, then $r_{\mathcal{G}}(v) \in \{\omega, \omega + 1\}$.*

Combining Corollary 4 and Lemma 5, we obtain an upper bound on the resilience of vertices with finite resilience.

► **Corollary 6.** *We have $r_{\mathcal{G}}(V) \cap \omega = \{0, 1, \dots, n\}$ for some $n < |V|$.*

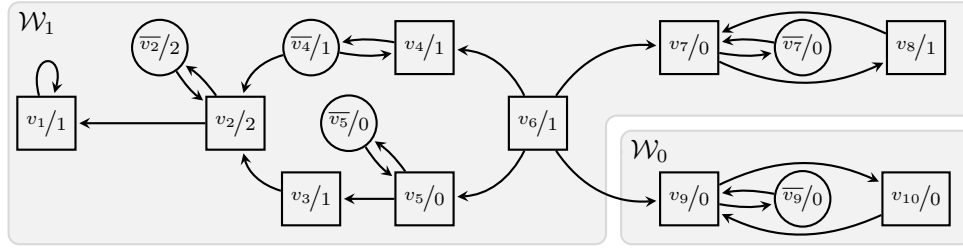
3.2 Characterizing Vertices of Resilience $\omega + 1$

Our goal in this subsection is to determine the vertices of resilience $\omega + 1$, i.e., those from which Player 0 can win even under an infinite number of disturbances. Intuitively, in this setting, we give Player 1 control over the disturbance edges, as he cannot execute more than infinitely many disturbances during a play.

In the following, we prove this intuition to be correct. To this end, we transform the arena of the game so that at a Player 0 vertex, first Player 1 gets to choose whether he wants to take one of the disturbance edges and, if not, gives control to Player 0, who is then able to use a standard edge.

Given a game $\mathcal{G} = (\mathcal{A}, \text{Win})$ with $\mathcal{A} = (V, V_0, V_1, E, D)$, we define the *rigged game* $\mathcal{G}_{\text{rig}} = (\mathcal{A}', \text{Win}')$ with $\mathcal{A}' = (V', V'_0, V'_1, E', D')$ such that $V' = V'_0 \cup V'_1$ with $V'_0 = \{\bar{v} \mid v \in V_0\}$ and $V'_1 = V$ and $D' = \emptyset$. The set E' of edges is the union of the following sets:

- D : Player 1 uses a disturbance edge.
- $\{(v, \bar{v}) \mid v \in V_0\}$: Player 1 does not use a disturbance edge and yields control to Player 0.
- $\{(\bar{v}, v') \mid (v, v') \in E \text{ and } v \in V_0\}$: Player 0 has control and picks a standard edge.
- $\{(v, v') \mid (v, v') \in E \text{ and } v \in V_1\}$: Player 1 takes a standard edge.



■ **Figure 2** The rigged game obtained for the game of Figure 1.

Further, $\text{Win}' = \{\rho \in (V')^\omega \mid h(\rho) \in \text{Win}\}$ where h is the homomorphism induced by $h(v) = v$ and $h(\bar{v}) = \varepsilon$ for every $v \in V$.

Figure 2 illustrates the construction of a rigged game for the example game of Figure 1 (note that the rigged game is also a parity game in this example). Note that the winning region of Player 0 corresponds to the vertices of resilience $\omega + 1$ in the game of Figure 1.

The following lemma formalizes the observation that $\mathcal{W}_0(\mathcal{G}_{\text{rig}})$ characterizes the vertices of resilience $\omega + 1$ in \mathcal{G} . Note that we have no assumptions on \mathcal{G} here.

► **Lemma 7.** *Let v be a vertex of game \mathcal{G} . Then, $v \in \mathcal{W}_0(\mathcal{G}_{\text{rig}})$ if and only if $r_{\mathcal{G}}(v) = \omega + 1$.*

Note that a slight extension of the rigged game also allows to characterize the vertices of resilience ω . To this end, one uses the same arena as for the rigged game, but adds to the winning condition of the rigged game all those plays during which Player 1 takes infinitely many disturbance edges. Then, Player 0 has to satisfy the original winning condition if only finitely many disturbance edges are taken by Player 1, but wins vacuously if Player 1 takes infinitely many disturbance edges. This is possible from exactly those vertices that have resilience ω . However, for our purposes, we do not need to investigate this modified rigged game. We have shown how to determine the vertices of finite resilience and those of resilience $\omega + 1$. Thus, all other vertices have resilience ω .

Furthermore, the proof of Lemma 7 also yields the preservation of positional strategies.

► **Corollary 8.** *Assume Player 0 has a positional winning strategy for \mathcal{G}_{rig} from v . Then, Player 0 has an $(\omega + 1)$ -resilient positional strategy from v .*

3.3 Computing Optimally Resilient Strategies

This subsection is concerned with computing the resilience and optimally resilient strategies. Here, we focus on positional and finite-state strategies, which are sufficient for the majority of winning conditions in the literature. Nevertheless, it is easy to see that our framework is also applicable to infinite-state strategies.

In the proof of Lemma 5, we construct strategies σ_f and σ_ω such that σ_f is $r_{\mathcal{G}}(v)$ -resilient from every v with $r_{\mathcal{G}}(v) \in \omega$ and such that σ_ω is ω -resilient from every v with $r_{\mathcal{G}}(v) \geq \omega$. Both strategies are obtained by combining winning strategies for some game $(\mathcal{A}, \text{Win} \cap \text{Safety}(U))$. However, even if these winning strategies are positional, the strategies σ_f and σ_ω are in general not positional. Nonetheless, we show in the proof of Theorem 9 that such positional winning strategies and a positional one for \mathcal{G}_{rig} can be combined into a single positional optimally resilient strategy.

Recall the requirements from Subsection 3.1 for a game $(\mathcal{A}, \text{Win})$: Win is prefix-independent and the game \mathcal{G}_U is determined for every $U \subseteq V$, where we write \mathcal{G}_U for the game $(\mathcal{A}, \text{Win} \cap \text{Safety}(U))$ for some $U \subseteq V$. To prove the results of this subsection, we

need to impose some additional effectiveness requirements: we require that each game \mathcal{G}_U and the rigged game \mathcal{G}_{rig} can be effectively solved. Also, we first assume that Player 0 has positional winning strategies for each of these games, which have to be effectively computable as well. We discuss the severity of these requirements in Section 4.

► **Theorem 9.** *Let \mathcal{G} satisfy all the above requirements. Then, the resilience of \mathcal{G} 's vertices and a positional optimally resilient strategy can be effectively computed.*

To prove this result, we refine the following standard technique that combines positional winning strategies for games with prefix-independent winning conditions.

Assume we have a positional strategy σ_v for every vertex v in some set $W \subseteq V$ such that σ_v is winning from v . Furthermore, let R_v be the set of vertices visited by plays that start in v and are consistent with σ_v . Also, let $m(v) = \min_{\prec} \{v' \in V \mid v \in R_{v'}\}$ for some strict total ordering \prec of W . Then, the positional strategy σ defined by $\sigma(v) = \sigma_{m(v)}(v)$ is winning from each $v \in W$, as along every play that starts in some $v \in W$ and is consistent with σ , the value of the function m only decreases. Thus, after it has stabilized, the remaining suffix is consistent with some strategy $\sigma_{v'}$. Hence, the suffix is winning for Player 0 and prefix-independence implies that the whole play is winning for her as well.

Here, we have to adapt this reasoning to respect the resilience of the vertices and to handle disturbance edges. Also, we have to pay attention to vertices of resilience $\omega + 1$, as plays starting in such vertices have to be winning under infinitely many disturbances.

Proof of Theorem 9. The effective computability of the resilience follows from the effectiveness requirements on \mathcal{G} : to compute the ranking r^* , it suffices to compute the disturbance and risk updates. The former are trivially effective while the effectiveness of the latter ones follows from our assumption. Lemma 5 shows that r^* correctly determines the resilience of all vertices with finite resilience. Finally by solving the rigged game, we also determine the resilience of the remaining vertices (Lemma 7). Again, this game can be solved by our assumption. Thus, it remains to show how to compute a positional optimally resilient strategy. To this end, we compute a positional strategy σ_v for every v satisfying the following:

- For every $v \in V$ with $r_{\mathcal{G}}(v) \in \omega \setminus \{0\}$, the strategy σ_v is winning for Player 0 from v for the game $(\mathcal{A}, \text{Win} \cap \text{Safety}(\{v' \in V \mid r_{\mathcal{G}}(v') < r_{\mathcal{G}}(v)\}))$. The existence of such a strategy has been shown in the proof of Item 1 of Lemma 5.
- For every $v \in V$ with $r_{\mathcal{G}}(v) = \omega$, the strategy σ_v is winning for Player 0 from v for the game $(\mathcal{A}, \text{Win} \cap \text{Safety}(\{v' \in V \mid r_{\mathcal{G}}(v') \in \omega\}))$. The existence of such a strategy has been shown in the proof of Item 2 of Lemma 5.
- For every $v \in V$ with $r_{\mathcal{G}}(v) = \omega + 1$, the strategy σ_v is $(\omega + 1)$ -resilient from v . The existence of such a strategy follows from Corollary 8, as we assume Player 0 to win \mathcal{G}_{rig} with positional strategies.
- For every $v \in V$ with $r_{\mathcal{G}}(v) = 0$, we fix an arbitrary positional strategy σ_v for Player 0.

Furthermore, we fix a strict linear order \prec on V such that $v \prec v'$ implies $r_{\mathcal{G}}(v) \leq r_{\mathcal{G}}(v')$, i.e., we order the vertices by ascending resilience. For $v \in V$ with $r_{\mathcal{G}}(v) \neq \omega + 1$, let R_v be the vertices reachable via disturbance-free plays that start in v and are consistent with σ_v . On the other hand, for $v \in V$ with $r_{\mathcal{G}}(v) = \omega + 1$, let R_v be the set of vertices reachable via plays with arbitrarily many disturbances that start in v and are consistent with σ_v .

We claim $R_v \subseteq \{v' \in V \mid r_{\mathcal{G}}(v') \geq r_{\mathcal{G}}(v)\}$ for every $v \in V$ (*). For v with $r_{\mathcal{G}}(v) \neq \omega + 1$ this follows immediately from the choice of σ_v . Thus, let v with $r_{\mathcal{G}}(v) = \omega + 1$. Assume σ_v reaches a vertex v' of resilience $r_{\mathcal{G}}(v') \neq \omega + 1$. Then, there exists a play ρ' starting in v' that is consistent with $\sigma_{v'}$, has less than $\omega + 1$ many disturbances and is losing for Player 0.

Thus the play obtained by first taking the play prefix to v' and then appending ρ' without its first vertex yields a play starting in v , consistent with σ_v , but losing for Player 0. This play witnesses that σ_v is not $(\omega + 1)$ -resilient from v , which yields the desired contradiction.

Let $m: V \rightarrow V$ be given as $m(v) = \min_{\prec} \{v' \in V \mid v \in R_{v'}\}$ and define the positional strategy σ as $\sigma(v) = \sigma_{m(v)}(v)$. By our assumptions, σ can be effectively computed. It remains to show that it is optimally resilient.

To this end, we apply the following two properties of edges (v, v') that may appear during a play that is consistent with σ , i.e., we either have $v \in V_0$ and $\sigma(v) = v'$ (which implies $(v, v') \in E$), or $v \in V_1$ and $(v, v') \in E$, or $v \in V_0$ and $(v, v') \in D$:

1. If $(v, v') \in E$, then we have $r_{\mathcal{G}}(v) \leq r_{\mathcal{G}}(v')$ and $m(v) \geq m(v')$. The first property follows from minimality of $m(v)$ and $(*)$ while the second follows from the definition of R_v .
2. If $(v, v') \in D$, then we distinguish several subcases, which all follow immediately from the definition of resilience:
 - If $r_{\mathcal{G}}(v) \in \omega$, then $r_{\mathcal{G}}(v') \geq r_{\mathcal{G}}(v) - 1$.
 - If $r_{\mathcal{G}}(v) = \omega$, then $r_{\mathcal{G}}(v') = \omega$, and
 - If $r_{\mathcal{G}}(v) = \omega + 1$, then $r_{\mathcal{G}}(v') = \omega + 1$ and $m(v) \geq m(v')$ (here, the second property follows from the definition of R_v for v with $r_{\mathcal{G}}(v) = \omega + 1$, which takes disturbance edges into account).

Now, consider a play $\rho = (v_0, b_0)(v_1, b_1)(v_2, b_2) \cdots$ that is consistent with σ . If $r_{\mathcal{G}}(v_0) = 0$ then we have nothing to show, as every strategy is 0-resilient from v .

Now, assume $r_{\mathcal{G}}(v_0) \in \omega \setminus \{0\}$. We have to show that if ρ has less than $r_{\mathcal{G}}(v_0)$ disturbances, then it is winning for Player 0. An inductive application of the above properties shows that in that case the last disturbance edge leads to a vertex of non-zero resilience. Furthermore, as the values $m(v_j)$ are only decreasing afterwards, they have to stabilize at some later point. Hence, there is some suffix of ρ that starts in some v' with non-zero resilience and that is consistent with the strategy $\sigma_{v'}$. Thus, the suffix is winning for Player 0 by the choice of $\sigma_{v'}$ and prefix-independence implies that ρ is winning for her as well.

Next, assume $r_{\mathcal{G}}(v_0) = \omega$. We have to show that if ρ has a finite number of disturbances, then it is winning for Player 0. Again, an inductive application of the above properties shows that in that case the last disturbance edge leads to a vertex of resilience ω or $\omega + 1$. Afterwards, the values $m(v_j)$ stabilize again. Hence, there is some suffix of ρ that starts in some v' with non-zero resilience and that is consistent with the strategy $\sigma_{v'}$. Thus, the suffix is winning for Player 0 by the choice of $\sigma_{v'}$ and prefix-independence implies that ρ is winning for her as well.

Finally, assume $r_{\mathcal{G}}(v_0) = \omega + 1$. Then, the above properties imply that ρ only visits vertices with resilience $\omega + 1$ and that the values $m(v_j)$ eventually stabilize. Hence, there is a suffix of ρ that is consistent with some $(\omega + 1)$ -resilient strategy $\sigma_{v'}$, where v' is the first vertex of the suffix. Hence, the suffix is winning for Player 0, no matter how many disturbances occurred. This again implies that ρ is winning for her as well. ◀

The algorithm determining the vertices' resilience and a positional optimally resilient strategy first computes r^* and the winner of the rigged game. This yields the resilience of \mathcal{G} 's vertices. Furthermore, the strategy is obtained by combining winning strategies for the games \mathcal{G}_U and for the rigged game as explained above.

Next, we analyze the complexity of the algorithm sketched above in some more detail. The inductive definition of the r_j can be turned into an algorithm computing r^* (using the results of Lemma 3 to optimize the naive implementation), which has to solve $\mathcal{O}(|V|)$ many games (and compute winning strategies for some of them) with winning condition $\text{Win} \cap \text{Safety}(U)$.

Furthermore, the rigged game, which is of size $\mathcal{O}(|V|)$, has to be solved and winning strategies have to be determined. Thus, the overall complexity is in general dominated by the complexity of solving these tasks.

We explicitly state one complexity result for the important case of parity games, using the fact that each of these games is then a parity game as well. Also, we use a quasipolynomial time algorithms for solving parity games [8] to solve the games \mathcal{G}_U and \mathcal{G}_{rig} .

► **Theorem 10.** *Optimally resilient strategies in parity games are positional and can be computed in quasipolynomial time.*

Using similar arguments, one can also analyze games where positional strategies do not suffice. As above, assume \mathcal{G} satisfies the same assumptions on determinacy and effectiveness, but only require that Player 0 has finite-state winning strategies⁴ for each game with winning condition $(\mathcal{A}, \text{Win} \cap \text{Safety}(U))$ and for the rigged game \mathcal{G}_{rig} . Then, one can show that she has a finite-state optimally resilient strategy. In fact, by reusing memory states, one can construct an optimally resilient strategy that it is not larger than any constituent strategy.

4 Discussion

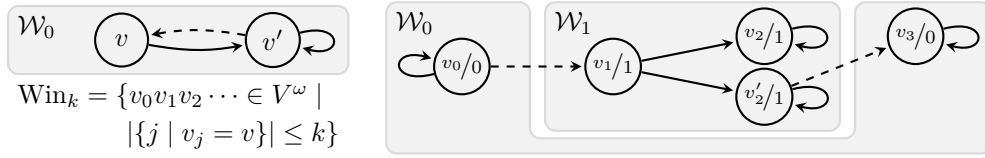
In this section, we discuss the assumptions required to be able to compute positional (finite-state) optimally resilient strategies with the algorithm presented in Section 3. To this end, fix a game $\mathcal{G} = (\mathcal{A}, \text{Win})$ with vertex set V and recall that \mathcal{G}_{rig} is the corresponding rigged game and that we defined $\mathcal{G}_U = (\mathcal{A}, \text{Win} \cap \text{Safety}(U))$ for $U \subseteq V$. Now, the assumptions on \mathcal{G} for Theorem 9 to hold are as follows: (1) Every game \mathcal{G}_U is determined. (2) Player 0 has a positional winning strategy from every vertex in her winning regions in the \mathcal{G}_U and in the game \mathcal{G}_{rig} . (3) Each \mathcal{G}_U and the game \mathcal{G}_{rig} can be effectively solved and positional winning strategies can be effectively computed for each such game. (4) Win is prefix-independent.

First, consider the determinacy assumption. It is straightforward to show $\mathcal{W}_0(\mathcal{G}_U) = \mathcal{W}_0(\mathcal{A} \setminus W, \text{Win} \cap (V \setminus W)^\omega)$ with $W = \mathcal{W}_1(\mathcal{A}, \text{Safety}(U))$. Thus, one can first determine and then remove the winning region of Player 1 in the safety game and then solve the subgame of \mathcal{G} played in Player 0's winning region of the safety game. Thus, all subgames of \mathcal{G} being determined suffices for our determinacy requirement being satisfied. The winning conditions one typically studies, e.g., parity and in fact all Borel ones [17], satisfy this property.

The next requirement concerns the existence of positional (finite-state) winning strategies for the games \mathcal{G}_U and \mathcal{G}_{rig} . For the \mathcal{G}_U , this requirement is satisfied if Player 0 has positional (finite-state) winning strategies for all subgames of \mathcal{G} . As every positional (finite-state) optimally resilient strategy is also a winning strategy in a certain subgame, this condition is necessary. Now, consider \mathcal{G}_{rig} , whose winning condition can be written as $h^{-1}(\text{Win})$ for the homomorphism h from Subsection 3.2. The winning conditions one typically studies, e.g., the Borel ones, are closed w.r.t. such supersequences. If \mathcal{G} is from a class of winning conditions that allows for positional (finite-state) winning strategies for Player 0, then this class typically also contains \mathcal{G}_{rig} . Also, the assumption on the effective solvability and computability of positional (finite-state) strategies is obviously necessary, as we solve a more general problem when determining optimally resilient strategies.

Finally, let us consider prefix-independence. If the winning condition is not prefix-independent, the algorithm presented in Section 3 does not compute the resilience of vertices correctly anymore. As an example, consider the family $\mathcal{G}_k = (\mathcal{A}, \text{Win}_k)$ of games shown

⁴ A finite state strategy is implemented by a finite automaton that processes play prefixes and outputs vertices to move to. See the full version [18] for a formal definition.



■ **Figure 3** Left: Counterexample to the correctness of the computation of resilience for games with prefix-dependent winning conditions. Right: Intuitively, moving from v_1 to v'_2 is preferable for Player 0, as it allows her to possibly “recover” from a first fault with the “help” of a second one.

on the left-hand side of Figure 3. In \mathcal{G}_k , it is Player 0’s goal to avoid more than k visits to v . Such a visit only occurs via a disturbance or if the initial vertex is v . Hence, we have $r_{\mathcal{G}_k}(v) = k$ and $r_{\mathcal{G}_k}(v') = k + 1$. Applying the algorithm from Section 3, however, the initial ranking function r_0 has an empty domain, since we have $\mathcal{W}_1(\mathcal{G}_k) = \emptyset$. Thus, the computation of the r_j immediately stabilizes, yielding r^* with empty domain. This is a counterexample to the generalization of Lemma 5 to prefix-dependent winning conditions.

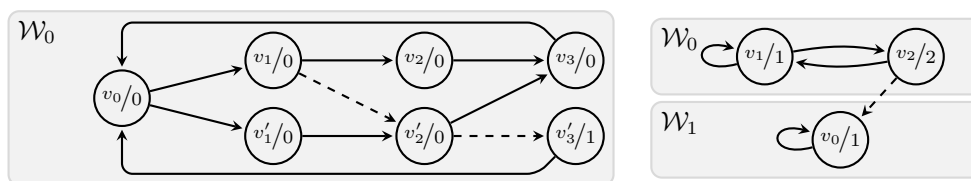
Nevertheless, one can still leverage the algorithm from Section 3 in order to compute the resilience of a wide range of games with prefix-dependent winning conditions. To this end, we extend the framework of game reductions to games with disturbances, in such a way that the existence of α -resilient strategies is preserved. Using this framework shows that Player 0 has a finite-state optimally resilient strategy in every game with ω -regular winning condition. Due to space restrictions, the details are spelled out in the full version [18]. Here, we just state the main result.

► **Theorem 11.** *Let a game \mathcal{G} be reducible to a game \mathcal{G}' with prefix-independent winning condition, which can be effectively computed from \mathcal{G} , and satisfies the assumptions from Section 3 (with finite-state strategies). Then, the resilience of \mathcal{G} ’s vertices and an optimally resilient finite-state strategy can be effectively computed.*

5 Outlook

We have developed a fine-grained view on the quality of strategies: instead of evaluating whether or not a strategy is winning, we compute its resilience against intermittent disturbances. While this measure of quality allows constructing “better” strategies than the distinction between winning and losing strategies, there remain aspects of optimality that are not captured in our notion of resilience. In this section we discuss these aspects and give examples of games in which there are crucial differences between optimally resilient strategies. In further research, we aim to synthesize optimal strategies with respect to these criteria.

As a first example, consider the parity game shown on the right-hand side of Figure 3. Vertices v_0 and v_3 have resilience 1 and $\omega + 1$, respectively, while vertices v_1 , v_2 , and v'_2 have resilience 0. Player 0’s only choice consists of moving to v_2 or to v'_2 from v_1 . Let σ and σ' be strategies for Player 0 that always move to v_2 and v'_2 from v_1 , respectively. Both strategies are optimally resilient. Hence, the algorithm from Section 3 may yield either one, depending on the underlying parity game solver used. Intuitively, however, σ' is preferable for Player 0, as a play prefix ending in v'_2 may proceed to her winning region if a single disturbance occurs. All plays encountering v_2 at some point, however, are losing for her. Hence, another interesting avenue for further research is to study **how to recover from losing**, i.e., how to construct strategies that leverage disturbances in order to leave Player 1’s winning region. For safety games, this has been addressed by Dallal, Neider, and Tabuada [10].



■ **Figure 4** Left: Moving to v_1 from v_0 allows Player 0 to minimize visits to odd colors, while moving to v_1' allows her to minimize the occurrence of disturbances. Right: Additional memory allows Player 0 to remain in v_1 longer and longer, thus decreasing the potential for disturbances.

The previous example shows that Player 0 can still make “meaningful” choices even if the play has moved outside her winning region. The game \mathcal{G} shown in the left-hand side of Figure 4 demonstrates that she can do so as well when remaining in vertices of resilience ω . Every vertex in \mathcal{G} has resilience ω , since every play with finitely many disturbances eventually remains in vertices of color 0. Moreover, the only choice to be made by Player 0 is whether to move to vertex v_1 or to vertex v_1' from vertex v_0 . Let σ and σ' be positional strategies that implement the former and the latter choice, respectively.

First consider a scenario in which visiting an odd color models the occurrence of some undesirable event, e.g., that a request has not been answered. In this case, Player 0 should aim to prevent visits to v_3' in \mathcal{G} , the only vertex of odd color. Hence, the strategy σ should be more desirable for her, as it requires two disturbances in direct succession in order to visit to v_3' . When playing consistently with σ' , however, a single disturbance suffices to visit v_3' .

On the other hand, consider a setting in which Player 0’s goal is to avoid the occurrence of disturbances. In that case, σ' is preferable over σ , as it allows for fewer situations in which disturbances may occur, since no disturbances are possible from vertices v_2 and v_3 .

Note that the goals of minimizing visits to vertices of odd color and minimizing the occurrence of disturbances are not contradictory: if both events are undesirable, it may be optimal for Player 0 to combine the strategies σ and σ' . In general, it is interesting to study how to **how to best brace for a finite number of disturbances**.

Recall that, due to Theorem 10, optimally resilient strategies for parity games do not require memory. In contrast, the game shown on the right-hand side of Figure 4 demonstrates that additional memory can serve to further improve such strategies. Any strategy for Player 0 that does not stay in v_1 from some point onwards is optimally resilient. However, every visit to v_2 risks a disturbance occurring, which would lead the play into a losing sink for Player 0. Hence, it is in her best interest to remain in vertex v_1 for as long as possible, thus minimizing the possibility for disturbances to occur. This behavior does, however, require memory to implement, as Player 0 needs to count the visits to v_1 in order to not remain in that state ad infinitum. Thus, for each optimally resilient strategy σ with finite memory there exists another optimally resilient strategy that uses more memory, but visits v_2 more rarely than σ , reducing the possibilities for disturbances to occur. Hence, it is interesting to study **how to balance avoiding disturbances with satisfying the winning condition**. This is particularly interesting if there is some cost assigned to disturbances.

Finally, another important and interesting aspect, which falls outside the scope of this paper, is to provide general guidelines and best practices on how to model synthesis problems by games with disturbances. We will address these problems in future research.

6 Conclusion

We presented an algorithm for computing optimally resilient strategies in games with disturbances to any game that satisfies some mild (and necessary) assumptions. Thereby, we have vastly generalized the work of Dallal, Neider, and Tabuada, who only considered safety games. Furthermore, we showed that optimally resilient strategies are typically of the same size as classical winning strategies. Finally, we have illustrated numerous novel phenomena that appear in the setting with disturbances but not in the classical one. Studying these phenomena is a very promising direction of future work.

References

- 1 Paul C. Attie, Anish Arora, and E. Allen Emerson. Synthesis of fault-tolerant concurrent programs. *ACM Trans. Program. Lang. Syst.*, 26(1):125–185, 2004. doi:10.1145/963778.963782.
- 2 Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: from parity games to safety games. *ITA*, 36(3):261–275, 2002. doi:10.1051/ita:2002013.
- 3 Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, Georg Hofferek, Barbara Jobstmann, Bettina Könighofer, and Robert Könighofer. Synthesizing robust systems. *Acta Inf.*, 51(3-4):193–220, 2014. doi:10.1007/s00236-013-0191-5.
- 4 Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In Ahmed Bouajjani and Oded Maler, editors, *CAV 2009*, volume 5643 of *LNCS*, pages 140–156. Springer, 2009. doi:10.1007/978-3-642-02658-4_14.
- 5 Roderick Bloem, Rüdiger Ehlers, Swen Jacobs, and Robert Könighofer. How to handle assumptions in synthesis. In Krishnendu Chatterjee, Rüdiger Ehlers, and Susmit Jha, editors, *SYNT 2014*, volume 157 of *EPTCS*, pages 34–50, 2014. doi:10.4204/EPTCS.157.7.
- 6 Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa’ar. Synthesis of Reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012. doi:10.1016/j.jcss.2011.08.007.
- 7 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Benjamin Monmege, Guillermo A. Pérez, and Gabriel Renault. Quantitative games under failures. In *FSTTCS 2015*, volume 45 of *LIPICs*, pages 293–306. Schloss Dagstuhl - LZI, 2015. doi:10.4230/LIPICs.FSTTCS.2015.293.
- 8 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, *STOC 2017*, pages 252–263. ACM, 2017. doi:10.1145/3055399.3055409.
- 9 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012. doi:10.1016/j.tcs.2012.07.038.
- 10 Eric Dallal, Daniel Neider, and Paulo Tabuada. Synthesis of safety controllers robust to unmodeled intermittent disturbances. In *CDC 2016*, pages 7425–7430. IEEE, 2016. doi:10.1109/CDC.2016.7799416.
- 11 Ali Ebneenasir, Sandeep S. Kulkarni, and Anish Arora. FTSyn: a framework for automatic synthesis of fault-tolerance. *STTT*, 10(5):455–471, 2008. doi:10.1007/s10009-008-0083-0.
- 12 Rüdiger Ehlers and Ufuk Topcu. Resilience to intermittent assumption violations in reactive synthesis. In Martin Fränzle and John Lygeros, editors, *HSCC 2014*, pages 203–212. ACM, 2014. doi:10.1145/2562059.2562128.

- 13 Alain Girault and Éric Rutten. Automating the addition of fault tolerance with discrete controller synthesis. *Form. Meth. in Sys. Des.*, 35(2):190–225, 2009. doi:10.1007/s10703-009-0084-y.
- 14 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.
- 15 Chung-Hao Huang, Doron A. Peled, Sven Schewe, and Farn Wang. A game-theoretic foundation for the maximum software resilience against dense errors. *IEEE Trans. Software Eng.*, 42(7):605–622, 2016. doi:10.1109/TSE.2015.2510001.
- 16 Rupak Majumdar, Elaine Render, and Paulo Tabuada. A theory of robust omega-regular software synthesis. *ACM Trans. Embedded Comput. Syst.*, 13(3):48:1–48:27, 2013. doi:10.1145/2539036.2539044.
- 17 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–371, 1975.
- 18 Daniel Neider, Alexander Weinert, and Martin Zimmermann. Synthesizing optimally resilient controllers. *arXiv*, 1709.04854, 2017. URL: <https://arxiv.org/abs/1709.04854>.
- 19 Paulo Tabuada, Sina Yamac Caliskan, Matthias Rungger, and Rupak Majumdar. Towards robustness for cyber-physical systems. *IEEE Trans. Automat. Contr.*, 59(12):3151–3163, 2014. doi:10.1109/TAC.2014.2351632.
- 20 Paulo Tabuada and Daniel Neider. Robust linear temporal logic. In *CSL 2016*, volume 62 of *LIPICs*, pages 10:1–10:21. Schloss Dagstuhl - LZI, 2016. doi:10.4230/LIPICs.CSL.2016.10.
- 21 Ufuk Topcu, Necmiye Ozay, Jun Liu, and Richard M. Murray. On synthesizing robust discrete controllers under modeling uncertainty. In Thao Dang and Ian M. Mitchell, editors, *HSCC 2012*, pages 85–94. ACM, 2012. doi:10.1145/2185632.2185648.
- 22 Johan van Benthem. An essay on sabotage and obstruction. In *Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, volume 2605 of *LNCS*, pages 268–276. Springer, 2005. doi:10.1007/978-3-540-32254-2_16.

Local Validity for Circular Proofs in Linear Logic with Fixed Points

Rémi Nollet

IRIF, Université Paris Diderot and CNRS, Paris, France
remi.nollet@irif.fr

Alexis Saurin

IRIF, CNRS, Université Paris Diderot and INRIA πr^2 , Paris, France
alexis.saurin@irif.fr

Christine Tasson

IRIF, Université Paris Diderot and CNRS, Paris, France
christine.tasson@irif.fr

Abstract

Circular (ie. non-wellfounded but regular) proofs have received increasing interest in recent years with the simultaneous development of their applications and meta-theory: infinitary proof theory is now well-established in several proof-theoretical frameworks such as Martin L of’s inductive predicates, linear logic with fixed points, etc. In the setting of non-wellfounded proofs, a validity criterion is necessary to distinguish, among all infinite derivation trees (aka. pre-proofs), those which are logically valid proofs. A standard approach is to consider a pre-proof to be valid if every infinite branch is supported by an infinitely progressing thread.

The paper focuses on circular proofs for MALL with fixed points. Among all representations of valid circular proofs, a new fragment is described, based on a stronger validity criterion. This new criterion is based on a labelling of formulas and proofs, whose validity is purely local. This allows this fragment to be easily handled, while being expressive enough to still contain all circular embeddings of Baelde’s μ MALL finite proofs with (co)inductive invariants: in particular deciding validity and computing a certifying labelling can be done efficiently. Moreover the Brotherston-Simpson conjecture holds for this fragment: every labelled representation of a circular proof in the fragment is translated into a standard finitary proof. Finally we explore how to extend these results to a bigger fragment, by relaxing the labelling discipline while retaining (i) the ability to locally certify the validity and (ii) to some extent, the ability to finitize circular proofs.

2012 ACM Subject Classification Theory of computation \rightarrow Logic, Theory of computation \rightarrow Proof theory, Theory of computation \rightarrow Linear logic, Theory of computation \rightarrow Logic and verification

Keywords and phrases sequent calculus, non-wellfounded proofs, circular proofs, induction, coinduction, fixed points, proof-search, linear logic, muMALL, finitization, infinite descent

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.35

Related Version Full version available at <https://hal.archives-ouvertes.fr/hal-01825477>.

Funding Partially funded by ANR Project RAPIDO, ANR-14-CE25-0007.

Acknowledgements We want to thank the anonymous reviewers for their very detailed comments.



  R mi Nollet, Alexis Saurin, and Christine Tasson;
licensed under Creative Commons License CC-BY

27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 35; pp. 35:1–35:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum f r Informatik, Dagstuhl Publishing, Germany

$$\frac{\vdash \Gamma, S \quad \vdash S^\perp, F[S/X]}{\vdash \Gamma, \nu X.F} \text{ } (\nu_{\text{inv}})$$

■ **Figure 1** Coinduction rule *à la* Park.

$$\frac{\frac{\vdots}{\vdash \mu X.X} (\mu) \quad \frac{\vdots}{\vdash \nu X.X, \Gamma} (\nu)}{\vdash \mu X.X} (\mu) \quad \frac{\vdots}{\vdash \nu X.X, \Gamma} (\nu)}{\vdash \Gamma} \text{ } (\text{Cut})$$

■ **Figure 2**

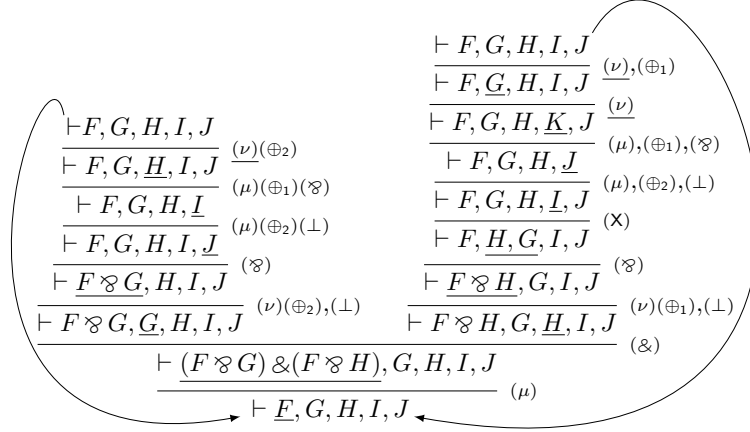
1 Introduction

Various logical settings have been introduced to reason about inductive and coinductive statements, both at the level of the logical languages modelling (co)induction (Martin L of's inductive predicates vs. fixed-point logics, that is μ -calculi) and at the level of the proof-theoretical framework considered (finite proofs with (co)induction *à la* Park [22] vs. infinite proofs with fixed-point/inductive predicate unfoldings) [8, 10, 11, 5, 2, 3]. Moreover, such proof systems have been considered over classical logic [8, 11], intuitionistic logic [12], linear-time or branching-time temporal logic [20, 19, 26, 27, 14, 15, 16] or linear logic [23, 17, 5, 4, 15].

In all those proof systems, the treatment of inductive and coinductive reasoning brings some highly complex proof figures. For instance, in proof systems using (co)induction rules *à la* Park, the rules allowing to derive a coinductive property (or dually to use an inductive hypothesis) have a complex inference of the form of fig. 1 (when presented in the setting of fixed-point logic – here we follow the one-sided sequent tradition of MALL that we will adopt in the rest of the paper). Not only is it difficult to figure out intuitively what is the meaning of this inference, but it is also problematic for at least two additional and more technical reasons: (i) it is hiding a cut rule that *cannot* be eliminated, which is problematic for extending the Curry-Howard correspondence to fixed-points logics, and (ii) it breaks the subformula property, which is problematic for proof search: at each coinduction rule, one has to guess an invariant (in the same way as one has to guess an appropriate induction hypothesis in usual mathematical proofs) which is problematic for automation of proof search.

Infinite (non-wellfounded) proofs have been proposed as an alternative in recent years [8, 10, 11]. By replacing the coinduction rule with simple fixed-point unfoldings and allowing for non-wellfounded branches, those proof systems address the problem of the subformula property for the cut-free systems. The cut-elimination dynamics for inductive-coinductive rules is also much simpler. Among those non-wellfounded proofs, circular, or cyclic proofs, that have infinite but regular derivations trees, have attracted a lot of attention for retaining the simplicity of the inferences of non-wellfounded proof systems but being amenable to a simple finite representation making it possible to have an algorithmic treatment of those proof objects.

However, in those proof systems when considering all possible infinite, non-wellfounded derivations (a. k. a. pre-proofs), it is straightforward to derive any sequent Γ (see fig. 2). Such pre-proofs are therefore unsound and one needs to impose a validity criterion to distinguish, among all pre-proofs, those which are logically valid proofs from the unsound ones. This condition will actually reflect the inductive and coinductive nature of our two fixed-point connectives: a standard approach [8, 10, 11, 23, 4] is to consider a pre-proof to be valid if



■ **Figure 3** Proof π_∞ .

every infinite branch is supported by an infinitely progressing thread. However, doing so, the logical correctness of circular proofs becomes a non-local property, much in the spirit of proof nets correctness criteria [18, 13].

Despite the need for a validity condition, circular proofs have recently received increasing interest with the simultaneous development of their applications and meta-theory: infinitary proof theory is now well-established in several proof-theoretical frameworks such as Martin Löf's inductive predicates, linear logic with fixed-points, *etc.*

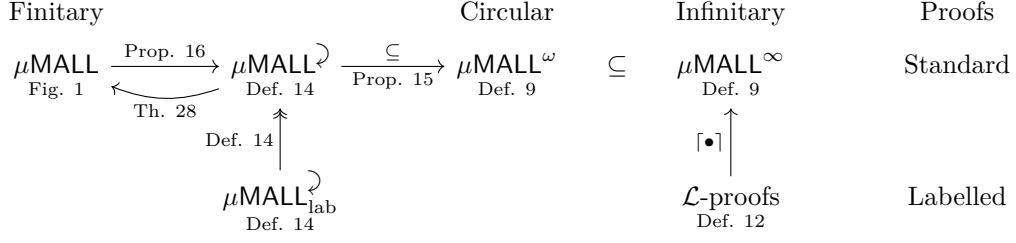
This paper is a contribution to two directions in the field of circular proofs:

1. the relationship between finite and circular proofs (at the level of provability and at the level of proofs themselves) and
2. the certification of circular proofs, that is the production of fast and/or small pieces of evidence to support validity of a circular pre-proof.

Comparing finite and infinite proofs is very natural. Informally, it amounts to considering the relative strength of inductive reasoning versus infinite descent: while infinite descent is a very old form of mathematical reasoning which appeared already in Euclid's *Elements* and was systematically investigated by Fermat, making precise its relationship with mathematical induction is still an open question for many proof formalisms. Their equivalence is known as the Brotherston–Simpson conjecture. While it is fairly straightforward to check that infinite descent (circular proofs) prove at least as many statements as inductive reasoning, the converse is complex and remains largely open. Last year, Simpson [24], on the one hand, and Berardi and Tatsuta [6, 7], on the other hand, made progress on this question but only in the framework of Martin Löf's inductive definitions, not in the setting of μ -calculi circular proofs in which invariant extraction is highly complex and known only for some fragments.

We conclude this introduction by considering a typical example of a circular proof with a complex validating thread structure: while this infinite proof has a regular derivation tree, its branches and threads have a complex geometry. The circular (pre-)proof of Figure 3 derives the sequent $\vdash F, G, H, I, J$ where $F = \mu X.(X \wp G) \& (X \wp H)$, $G = \nu X.X \oplus \perp$, $H = \nu X.\perp \oplus X$, $I = \mu Z.((Z \wp J) \oplus \perp)$, $J = \mu X.(K \wp X) \oplus \perp$ and $K = \nu Y.\mu Z.((Z \wp \mu X.(Y \wp X) \oplus \perp) \oplus \perp)$.

This example of a circular derivation happens to be valid (it is a μMALL^ω proof) but the description of its validating threads is quite complex. Indeed, each infinite branch β is validated by exactly one thread (see next section for detailed definitions) going through either G , H or K depending on the shape of the branch *at the limit* (infinite branches of this derivations can be described as ω -words on $A = \{l, r\}$ depending on whether the left or right back-edge is taken):



■ **Figure 4** Relations between the different systems used in the paper.

- (i) if β *ultimately* follows always the left cycle ($A^* \cdot l^\omega$), the unfolding of H validates β ;
- (ii) if β *ultimately* follows always the right cycle ($A^* \cdot r^\omega$), the unfolding of G validates β ;
- (iii) if β *endlessly switches* between left and right cycles ($A^* \cdot (r^+ \cdot l^+)^\omega$), K validates β .

The description of the thread validating this proof is thus complex. This is reflected in the difficulty to provide a local way to validate this proof and in the lack of a general method for finitizing this into a μMALL proof: to our knowledge, the usual finitization methods (working only for fragments of μMALL circular proofs) do not apply here.

Organization and contributions of the paper. In section 2, we provide the necessary background on infinitary and circular proof theory of multiplicative additive linear logic with least and greatest fixed points (respectively μMALL^∞ and μMALL^ω). Section 3 studies an approach to circular proofs based on labellings of greatest fixed points. We first motivate in section 3.1 such labellings as an alternative way to express the validating threads. Then, in section 3.2 we introduce finite representations of pre-proofs and use such labellings in order to locally certify their validity. Finally, in section 3.3, we turn to alternative characterizations of those circular proofs which can be labelled. The fragment of labellable proofs, while quite constrained (for instance, it does not include the example of Figure 3), is already enough to capture the circular proofs obtained by translation of μMALL proofs. In section 4, we address the converse: for any labelled derivation tree with back-edges, we provide a corresponding μMALL proof by generating a (co)inductive invariant based on an inspection of the labelling structure. Therefore, we answer the Brotherston–Simpson conjecture in a restricted fragment. In section 5, we introduce a more permissive labelling strategy that allows to label more proofs (in particular by allowing to loop not only on (ν) rules but on any rule) and that still ensures validity of the labellable derivations. For this relaxed labelling, we label the example of Figure 3 and show how to finitize it by adapting the method of section 4. Nevertheless, there is not yet a general method applicable to the complete extended labelling fragment. Relations between the various systems considered in the paper are summarized in Figure 4.

2 Background on circular proofs

We recall μMALL^∞ and μMALL^ω , which are non-wellfounded and circular proof systems, respectively, for an extension of MALL with least and greatest fixed points operators [4, 15].

► **Definition 1.** Given a set of fixed point operators $\mathcal{F} = \{\mu, \nu\}$ and an infinite set of propositional variables $\mathcal{V} = \{X, Y, \dots\}$, μMALL **pre-formulas** are inductively defined as: $A, B ::= \mathbf{0} \mid \top \mid A \oplus B \mid A \& B \mid \perp \mid \mathbf{1} \mid A \wp B \mid A \otimes B \mid X \mid \sigma X.A$ with $X \in \mathcal{V}$ and $\sigma \in \mathcal{F}$. $\sigma \in \mathcal{F}$ binds the variable X in A . From there, bound variables, free variables and capture-avoiding substitution are defined in a standard way. The subformula ordering is denoted \leq and $\text{fv}(\bullet)$ denotes free variables. When a pre-formula is closed, we simply call it a **formula**.

$$\begin{array}{c}
\frac{}{\vdash \mathbf{1}} \text{ (1)} \qquad \frac{}{\vdash F, F^\perp} \text{ (Ax)} \qquad \frac{\vdash \Gamma, F \quad \vdash F^\perp, \Delta}{\vdash \Gamma, \Delta} \text{ (Cut)} \qquad \frac{\vdash \Gamma, F, G, \Delta}{\vdash \Gamma, G, F, \Delta} \text{ (X)} \\
\frac{\vdash \Gamma}{\vdash \perp, \Gamma} \text{ (\perp)} \qquad \frac{\vdash F_i, \Gamma}{\vdash F_1 \oplus F_2, \Gamma} \text{ (\oplus_i)} \qquad \frac{\vdash F, \Gamma \quad \vdash G, \Delta}{\vdash F \otimes G, \Gamma, \Delta} \text{ (\otimes)} \qquad \frac{\vdash F[\mu X.F/X], \Gamma}{\vdash \mu X.F, \Gamma} \text{ (\mu)} \\
\frac{}{\vdash \top, \Gamma} \text{ (\top)} \qquad \frac{\vdash F, \Gamma \quad \vdash G, \Gamma}{\vdash F \& G, \Gamma} \text{ (\&)} \qquad \frac{\vdash F, G, \Gamma}{\vdash F \wp G, \Gamma} \text{ (\wp)} \qquad \frac{\vdash G[\nu X.G/X], \Gamma}{\vdash \nu X.G, \Gamma} \text{ (\nu)}
\end{array}$$

■ **Figure 5** μMALL^∞ inference rules.

Note that negation is not part of the syntax, so that we do not need any positivity condition on fixed-points expressions. We define negation, $(\bullet)^\perp$, as a meta-operation on pre-formulas and will use it on formulas.

► **Definition 2. Negation**, $(\bullet)^\perp$, is the involution on pre-formulas, satisfying: $\mathbf{0}^\perp = \top$, $(A \oplus B)^\perp = B^\perp \& A^\perp$, $\mathbf{1}^\perp = \perp$, $(A \otimes B)^\perp = B^\perp \wp A^\perp$, $X^\perp = X$, $(\mu X.A)^\perp = \nu X.A^\perp$.

► **Example 3.** The previous definition yields, *e.g.* $(\mu X.X)^\perp = (\nu X.X)$ and $(\mu X.\mathbf{1} \oplus X)^\perp = (\nu X.X \& \perp)$, as expected [3]. Note that we also have $(A[B/X])^\perp = A^\perp[B^\perp/X]$.

The reader may find it surprising to define $X^\perp = X$, but it is harmless since our proof system only deals with formulas (*i.e.* closed pre-formulas) as exemplified right above.

Fixed-points logics come with a notion of subformulas slightly different from usual:

► **Definition 4.** The **Fischer-Ladner closure** of a formula F , $\text{FL}(F)$, is the least set of formulas such that $F \in \text{FL}(F)$ and, whenever $G \in \text{FL}(F)$, (i) $G_1, G_2 \in \text{FL}(F)$ if $G = G_1 \star G_2$ for any $\star \in \{\oplus, \&, \wp, \otimes\}$; (ii) $B[G/X] \in \text{FL}(F)$ if G is $\mu X.B$ or $\nu X.B$. We say that G is a **FL-subformula** of F if $G \in \text{FL}(F)$.

In this work we choose to present sequents as lists of formulas together with an explicit exchange rule. Another usual choice is to present sequents as multisets of formulas. Yet, our approach takes the viewpoint of structural proof theory in which one is willing not to equate too many proofs. In particular, the sequents as (multi)sets are not relevant from the Curry-Howard perspective, *e.g.* it would equate the proofs denoting the two booleans. Moreover, most proof theoretical observations actually hold when one distinguishes between several *occurrences* of a formula in a sequent, giving the ability to *trace* the provenance of each occurrence. In [4], formula occurrences are localized formulas and the interested reader will check that all the following results hold also in this more explicit approach.

► **Definition 5.** A **pre-proof** of μMALL^∞ is a possibly infinite tree generated from the inference rules given in fig. 5.

Recall that μMALL [3], on the opposite, is obtained by forming only finite trees and by taking, instead of the (ν) rule of μMALL^∞ , the rule with explicit invariant of fig. 1.

When writing sequent proofs, we will often omit exchange rules, using the fact that every inference of def. 5 admits a derivable variant (preserving every correctness criterion considered in the paper) allowing the principal formula of the inference as well as the context (or auxiliary) formulas to be anywhere in the sequent, *e.g.* for the \wp introduction, the derived rule is $\frac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, A \wp B, \Delta}$ (\wp). We will use those derived rules when it is not ambiguous with respect to the formula occurrence relation. The following notion of threading function is folklore generally left implicit.

3 Labelling as validity

3.1 \mathcal{L} -proofs

In this subsection, we briefly mention an alternative approach to ensure validity of μMALL^∞ pre-proofs, aiming at motivating the tools used in the remainder of this paper (see details in the extended version). The idea is to witness thread progress by adding labels on some formulas.

► **Definition 11** (Labelled formulas). Let \mathcal{L} be an infinite countable set of **atoms** and call **labels** any finite list of atoms. Let $\mathcal{F}^\mathcal{L}$ be the set $\{\sigma^L \mid \sigma \in \{\mu, \nu\}, L \in \text{list}(\mathcal{L})\}$. **Labelled formulas**, or \mathcal{L} -formulas, are defined as μMALL formulas, by replacing \mathcal{F} with $\mathcal{F}^\mathcal{L}$ in the grammar of formulas (def. 1). Negation is lifted to labelled formulas, as $(\mu^L X.A)^\perp = \nu^L X.A^\perp$. We write $\sigma X.A$ for $\sigma^\emptyset X.A$ and standard, unlabelled formulas can thus be seen as labelled formulas where every label is empty. We define a **label-erasing** function $\lceil \bullet \rceil$ that associates to every \mathcal{L} -formula A the μMALL -formula $\lceil A \rceil$ obtained by erasing every label and satisfying $\lceil \sigma^L X.B \rceil = \sigma X. \lceil B \rceil$.

The standard μMALL^∞ proof system is adapted, to handle labels, by updating (Ax) and (ν) as $\frac{A \perp B}{\vdash A, B} (\text{Ax}') \quad \frac{\vdash A[\nu^{L,a} X.A], \Gamma}{\vdash \nu^L X.A, \Gamma} (\nu_b(a))$ where (i) A, B are said to be **orthogonal**, written $A \perp B$, when $\lceil A \rceil = \lceil B \rceil^\perp$ and (ii) in $(\nu_b(a))$, a must be a fresh label name, *i. e.* a does not appear free in the conclusion sequent of $(\nu_b(a))$ (in particular, $a \notin L$). Since we are in a one-sided framework, only labels on ν operators are relevant. Therefore, from now on, formulas have non-empty labels only on ν and require, for the cut inference, that all labels of cut formulas are empty. **\mathcal{L} -pre-proofs** are, as in def. 5, possibly infinite derivations using \mathcal{L} -formulas, and the validity condition is expressed in terms of labels:

► **Definition 12** (\mathcal{L} -proof). An **\mathcal{L} -proof** is an \mathcal{L} -pre-proof such that for every infinite branch $\gamma = (s_i)_{i \in \omega}$, there exists a sequence $(\nu^{L_i} X.G_i)_{i \in \omega}$ and a strictly increasing function ϵ on natural numbers such that for every $i \in \omega$, (i) the formula $\nu^{L_i} X.G_i$ is principal in $s_{\epsilon(i)}$ (ii) $\lceil \nu^{L_i} X.G_i \rceil = \lceil \nu^{L_{i+1}} X.G_{i+1} \rceil$ and (iii) $L_{i+1} = (L_i, a_i)$ for some $a_i \in \mathcal{L}$.

Note that the label-erasing function $\lceil \bullet \rceil$ is easily lifted to sequents and \mathcal{L} -pre-proofs. And if π is an \mathcal{L} -proof, then $\lceil \pi \rceil$ is a μMALL^∞ proof.

3.2 Finite representations of circular \mathcal{L} -proofs.

We now turn our attention to finite representations of (circular) \mathcal{L} -proofs. Immediately a difficulty occurs in comparison to non-labelled proofs: whereas an infinite non-labelled proof may happen to be regular, a valid \mathcal{L} -proof cannot be circular, for, along every infinite branch, the sets of labels will grow endlessly. To form circular proofs with labels, some atoms must be forgotten when going bottom-up.

We introduce two more rules: $(\zeta(a))$ and (lw) . The first one allows to forget one atom, just before recreating it by means of a back-edge to an already encountered ν -rule. The other one allows to forget any atom that will not be used to validate the proof. It is used to synchronise the different labels in a sequent before travelling through a back-edge.

■ labelled back-edge: $\frac{\vdash \nu^{L,a} X.A, \Gamma}{\vdash \nu^L X.A, \Gamma} (\zeta(a))$ with the constraint that it must be the source of a back-edge to the conclusion of a $\frac{\vdash A[\nu^{L,a} X.A], \Gamma}{\vdash \nu^L X.A, \Gamma} (\nu_b(a))$ below $(\zeta(a))$.

■ labelled weakening: $\frac{\vdash \Gamma, B[\nu^L X.A], \Delta}{\vdash \Gamma, B[\nu^{L,a} X.A], \Delta} (\text{LW})$

► **Definition 13** ($\mu\text{MALL}_{\text{lab}}^{\curvearrowright}$). $\mu\text{MALL}_{\text{lab}}^{\curvearrowright}$ denotes the finite derivations of \mathcal{L} -sequents built from the rules in fig. 5 by replacing (ν) by $(\nu_b(a))$, $(\varrho(a))$, (LW) , such that (i) the root sequent has empty labels and (ii) in every two $(\nu_b(a))$ and $(\nu_b(b))$ occurring in the proofs, $a \neq b$.

The label-erasing function $[\bullet]$ lifts to a translation from $\mu\text{MALL}_{\text{lab}}^{\curvearrowright}$ to the finite representations of μMALL^ω pre-proofs. Every rule of the labelled $\mu\text{MALL}_{\text{lab}}^{\curvearrowright}$ proof is sent by $[\bullet]$ to a valid rule of unlabelled μMALL^ω , except for the (LW) rule, which can safely be removed:

$$\frac{\vdash \Gamma, B[\nu^L X.A], \Delta}{\vdash \Gamma, B[\nu^{L,a} X.A], \Delta} (\text{LW}) \quad \text{becomes useless} \quad \frac{\vdash [\Gamma], [B][\nu X. [A]], [\Delta]}{\vdash [\Gamma], [B][\nu X. [A]], [\Delta]} \quad (1)$$

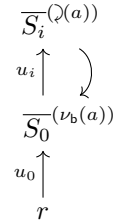
Since $\mu\text{MALL}_{\text{lab}}^{\curvearrowright}$ proofs are finite, label-erasing and unfolding give rise to μMALL^ω pre-proofs:

► **Definition 14** ($\mu\text{MALL}^{\curvearrowright}$). We denote as $\mu\text{MALL}^{\curvearrowright}$ the set of circular pre-proofs that are obtained from $\mu\text{MALL}_{\text{lab}}^{\curvearrowright}$ by label-erasing and total unfolding.

► **Proposition 15** ($\mu\text{MALL}^{\curvearrowright} \subseteq \mu\text{MALL}^\omega$). *Every pre-proof of μMALL^ω that is the image of a proof in $\mu\text{MALL}_{\text{lab}}^{\curvearrowright}$ by label-erasing and total unfolding satisfies thread validity.*

Proof sketch (details are in appendix A, p. 19). Consider a pre-proof $[\pi]$ in $\mu\text{MALL}^{\curvearrowright}$ which is the image of an \mathcal{L} -proof π in $\mu\text{MALL}_{\text{lab}}^{\curvearrowright}$. We want to prove that every infinite branch b in $[\pi]$ contains a valid thread (see def. 7). Let b_0 be the corresponding infinite \mathcal{L} -branch in π . Notice that there is a sequent S_0 which is the lowest back-edge target crossed infinitely often by b_0 . Besides, S_0 is the conclusion of a $(\nu_b(a))$ rule, which unfolds some $\nu^L X.A$.

We decompose b_0 , with root r ; S_0 conclusion of $(\nu_b(a))$ and $\nu^L X.A$ at position p_0 in S_0 ; for any $i \geq 1$, S_i conclusion of a back-edge $(\varrho(a))$ with $\nu^{L,a} X.A$ at position p_0 in S_i . Then we notice that $\mathfrak{T}(u_i)(p_0)$ is a thread $(S_0, p_0) \xrightarrow{*} (S_i, p_0)$ which is progressing, as its source is the principal conclusion of the rule $(\nu_b(a))$. By gluing the $\mathfrak{T}(u_i)(p_0)$ and then erasing labels, we get a valid thread of b in $[\pi]$. ◀



► **Proposition 16.** *μMALL proofs can be translated to $\mu\text{MALL}^{\curvearrowright}$.*

Proof. The target of the usual translation [15] $\mu\text{MALL} \rightarrow \mu\text{MALL}^\omega$ is included in $\mu\text{MALL}^{\curvearrowright}$. The key case of this translation is shown in appendix A. ◀

Observe that a proof in $\mu\text{MALL}^{\curvearrowright}$ is not, in general, the translation of a μMALL proof.

3.3 Two alternative characterizations of $\mu\text{MALL}^{\curvearrowright}$

In the two following sections, we give two characterizations of $\mu\text{MALL}^{\curvearrowright}$ through validating sets (def. 20) and through a threading criterion over back-edges (def. 24).

► **Definition 17.** Given a directed graph $G = (V, E)$ and a set $S \subseteq V$, the set of vertices from which S is accessible is denoted as $S\uparrow := \{v \in V \text{ s.t. } \exists s \in S, v \rightarrow^* s\}$. Similarly $S\downarrow$ is the set of vertices accessible from S .

► **Definition 18** (G_π). For a finite representation π of a μMALL^ω pre-proof, the graph G_π is s. t. (i) its **vertices** are all positions of ν -formulas in all occurrences of sequents in π , plus

$$\text{the vertex } \perp: V_\pi := \left\{ \begin{array}{l} (i) \text{ } \nu \text{ position of a sequent } \Gamma \text{ in } \pi \\ (v, i, p) \text{ such that } (ii) \text{ } i \text{ position of a formula } A \text{ in } \Gamma \\ (iii) \text{ } p \text{ position of a } \nu\text{-subformula in } A \end{array} \right\} \uplus \{\perp\};$$

(ii) its **edges** go from a position in a formula to the position that comes from it in the sequent just below, as induced by the threading function of def. 6, or to the extra vertex \perp if it is a cut formula. In case this is a conclusion formula, there is no outgoing edge.

► **Definition 19** (G_r, S_r, T_r). Let π be a finite representation of a μMALL^ω pre-proof and (r) an occurrence of a (ν) -rule. We define the subgraph $G_r = (V_r, E_r)$ of G_π and $S_r, T_r \subseteq V_r$ st:

- **vertices** V_r are the extra vertex \perp plus all positions that are in the conclusion of this rule and in all above sequents, that is all sequents from which the conclusion of (r) can be reached, in the sense of def. 17;
- **edges** E_r are all edges of G_π between those vertices minus the edges of G_π that are induced by the back-edges of π targetting the conclusion of (r) , if there are some.
- $S_r \subseteq V_r$ is the set of all positions of the principal formulas of the sources sequents of the back-edges targetting the conclusion of (r) ;
- $T_r \subseteq V_r$ is the set of all positions of all subformulas of the conclusion of (r) except for the very position of its principal formula, plus the extra vertex \perp .

► **Definition 20.** Let (r) be an occurrence of a (ν) -rule in a pre-proof π of μMALL^ω . A **validating set** for (r) is a set $L \subseteq V_\pi$ such that $L = L\downarrow$ and $S_r \subseteq L \subseteq (V_r \setminus T_r)$.

► **Proposition 21.** Let (r) be an occurrence of a (ν) -rule of a pre-proof π of μMALL^ω . There exists a validating set for (r) iff T_r is not accessible from S_r in G_r iff $S_r\downarrow \subseteq V_r \setminus (T_r\uparrow)$. In this case, $S_r\downarrow$ is the smallest validating set of (r) and $V_r \setminus (T_r\uparrow)$ is the biggest one.

Proof. It is based on the fact that the complement of a downward-closed set is upward-closed. We then get the inclusions : $S_r \subseteq S_r\downarrow \subseteq L\downarrow = L \subseteq V_r \setminus (T_r\uparrow) \subseteq V_r \setminus T_r$. ◀

The following proposition gives an alternative criterion for μMALL^ω (see app. A, p. 19):

► **Proposition 22.** A finite representation π of a μMALL^ω pre-proof is a representation of a $\mu\text{MALL}_{\text{lab}}^\omega$ proof iff every occurrence of a ν -rule of π has a validating set.

► **Proposition 23.** Checking validity of a $\mu\text{MALL}_{\text{lab}}^\omega$ pre-proof is decidable. Membership in μMALL^ω can be decided in a time quadratic in the size of the (circular) pre-proof.

Proof. The former is immediate. The latter reduces to checking accessibility in a graph for each back-edge target, which can be done in quadratic time. ◀

► **Definition 24.** A finite representation of a μMALL^ω pre-proof finite representation is **strongly valid** when:

- (i) every back-edge targets the conclusion of a (ν) rule and
- (ii) if an occurrence (r') of $\frac{\vdash A[\nu X.A], \Gamma}{\vdash \nu X.A, \Gamma} (\nu)$ is the target of a back-edge, coming from an occurrence (r) of $\frac{\vdash \nu X.A, \Gamma}{\vdash \nu X.A, \Gamma}$ then every path t starting from the principal formula $\nu X.A$ of the conclusion of (r) , following the thread function (potentially through several back-edges, but never on or below the occurrence (r') of (ν)), ends on the principal formula $\nu X.A$ of the conclusion of (r') .

► **Proposition 25.** *A finite representation π of a μMALL^ω pre-proof is strongly valid iff every ν -rule of π has a validating set iff it is the representation of a $\mu\text{MALL}_{\text{lab}}^\curvearrowright$ proof.*

Proof. See proof in appendix B, p. 21. ◀

4 On Brotherston-Simpson's conjecture: finitizing circular proofs

The aim of this section is to prove a converse of prop. 16: *Every provable sequent of $\mu\text{MALL}^\curvearrowright$ is provable in μMALL .*

Let us consider a $\mu\text{MALL}^\curvearrowright$ proof π . Up to renaming of bound variables, we can assume that all (ν_b) rules are labelled by distinct labels. For every two labels a and b occurring in π , we say that $a \leq b$ whenever $(\nu_b(a))$ is under $(\nu_b(b))$. This order is well-founded because finite.

► **Definition 26.** For every rule $\frac{\vdash A[\nu^{V,a}X.A], \Gamma}{\vdash \nu^V X.A, \Gamma} (\nu_b(a))$ we define $\Gamma_{(a)}$ to be Γ .

We now define (i) for each atom a a sequent Γ_a formed of non-labelled formulas; (ii) for each formula A (with labels) occurring in the proof, a formula $\llbracket A \rrbracket$ without labels:

- **Definition 27.** We define by mutual induction: (1) $\Gamma_a := \llbracket \Gamma_{(a)} \rrbracket$.
(2) $H_\emptyset[F] := F$ and $H_{V,a}[F] := \otimes \Gamma_a^\perp \oplus H_V[F]$. (*i. e.* $H_V[F]$ is isomorphic to $(\bigoplus_{a \in V} \otimes \Gamma_a^\perp) \oplus F$.)
(3) By induction on formula A $\llbracket A \rrbracket$ is: (i) $\llbracket \nu^V X.A \rrbracket := \nu X.H_V[\llbracket A \rrbracket]$ (ii) it is homomorphic on other connectives: $\llbracket X \rrbracket := X$, $\llbracket \mathbf{1} \rrbracket := \mathbf{1}$, $\llbracket \mu X.A \rrbracket := \mu X.\llbracket A \rrbracket$, $\llbracket A \otimes B \rrbracket := \llbracket A \rrbracket \otimes \llbracket B \rrbracket$, *etc.*
(3) $\llbracket \cdot \rrbracket$ is lifted from formulas to sequences of formulas, pointwise.

This is well-founded because since any two distinct ν_b rules wear distinct variables the only Γ_b that are needed in the computation of Γ_a are those with $b < a$. Note that $\llbracket A \rrbracket = A$ as soon as A has no label variable. We can now state and prove the finitization theorem:

► **Theorem 28.** *Every provable sequent of $\mu\text{MALL}^\curvearrowright$ is provable in μMALL .*

Proof. Let π be a $\mu\text{MALL}_{\text{lab}}^\curvearrowright$ proof and replace, everywhere, each formula A by $\llbracket A \rrbracket$. All rules in this (almost) new derivation are now valid instances of μMALL rules, except for (ν_b) , (LW) and (\curvearrowright) rules. Actually, images of these rules by sequent translation $\llbracket \cdot \rrbracket$ are derivable in μMALL as shown in fig. 7 (a), (b) and (c) for (\curvearrowright) , (LW) and (ν_b) , respectively.

Replacing each instance of a (ν_b) , (LW) or (\curvearrowright) rule in π by its derived version, we get a fully valid proof of μMALL . If the conclusion of the original $\mu\text{MALL}^\curvearrowright$ proof was $\vdash \Gamma$ then what we get is a proof in μMALL of $\vdash \llbracket \Gamma \rrbracket$, *i. e.* the conclusion of the original $\mu\text{MALL}^\curvearrowright$ proof, if Γ contains no label variable. ◀

5 Relaxing the labelling of proofs

In this section, we discuss a possible extension of the labelling defined in section 3, in order to capture more proofs retaining (i) the ability to locally certify the validity and (ii) to some extent, the ability to finitize circular proofs. In order to motivate this extension, we shall consider a simpler example than the one in fig. 3 (π_∞).

Let D be an arbitrary formula. Lists of D can be represented as proofs of $L_0 := \mu X.\mathbf{1} \oplus (D \otimes X)$ and it is possible to encode in μMALL^ω the function taking two lists and

unfolding, the labelling must account for the progression of a thread. That is why every atomic label is now given in one of two modes: a passive mode ($a-$) and an active one ($a+$). Only an unfolding by a ν can turn a $-$ into a $+$.

Let us now turn back to our introductory example: π_∞ . For that example, simply separating the introduction of back-edges and the coinductive progress is not enough. Indeed, since targets of back-edges do not require to unfold a ν , there is *a priori* no reason to require that the sequents contains some ν -formula. While this is slightly hidden in the merge example, π_∞ gives a clear example of that and suggests that the (Rec) inference should have the ability to add labels *deeply* in the sequent, *i. e.* not only on the topmost ν fixed-points, but also to greatest fixed points occurring under some other connectives. The same remark applies to the back-edge rule since its conclusion sequents have the same structure as those of (Rec).

Driven by these observations, we now define a new labelling of circular preproofs and prove its correctness with respect to thread-validity.

► **Definition 29** (Extended labelling). Labelled formulas are built on the same grammar as previously, except that labels are lists of **signed variables**, that is of pairs of a variable and a symbol in $\{+, -\}$. Derivations are built with μ MALL inferences plus the following rules:

$$\frac{\vdash \nu^L X.A, \Gamma}{\vdash \nu^{L, a-} X.A, \Gamma} (\text{LW}(a-)) \quad \frac{\vdash \nu^{L, a-, L'} X.A, \Gamma}{\vdash \nu^{L, a+, L'} X.A, \Gamma} (\text{LW}(a+)) \quad \frac{\vdash A[\nu^{a_1+, \dots, a_n+} X.A], \Gamma}{\vdash \nu^{a_1-, \dots, a_n-} X.A, \Gamma} (\nu) \quad \frac{\vdash \Gamma[\nu^{L, a-} X.A]}{\vdash \Gamma[\nu^L X.A]} (\text{Rec}(a)) \quad \frac{}{\vdash \Gamma[\nu^{L, a+} X.A]} (\text{Rec}(a))$$

and the constraints that:

- a cut-formula cannot contain a non-empty label;
- all (Rec) rules must wear distinct variables;
- every (Rec(a)) rule must have at least one occurrence of “ $a-$ ” in its premise;
- each $\frac{}{\vdash \Gamma[\nu^{L, a+} X.A]} (\text{Rec}(a))$ rule is connected to the premise of a $\frac{\vdash \Gamma[\nu^{L, a-} X.A]}{\vdash B[\nu^L X.A], \Gamma} (\text{Rec}(a))$ via a back-edge. This implies in particular that this $(\text{Rec}(a))$ must be above this $(\text{Rec}(a))$ and that the premise of this (Rec(a)) must be the same sequent as the conclusion of this $(\text{Rec}(a))$ except for the change of sign of a , at every of its occurrences in the sequent.

► **Proposition 30** (Soundness of labelling). *If π is an extended labelled circular representation then $[\pi]$ is a circular representation of a valid μ MALL $^\omega$ proof.*

Proof. See proof in appendix C, p. 21. ◀

We now label our two examples with this new system. We will show that, while it is quite straightforward for the interleaving, it requires to unfold one back-edge of π_∞ .

π_∞ is presented labelled according to the extended labelling of fig. 9a. We make K apparent as a subformula of I and J respectively by decomposing:

$$I = I'[K] \quad J = J'[K] \quad J'[Y] := \mu X.((Y \wp X) \oplus \perp) \quad I'[Y] := \mu Z.((Z \wp J'[Y]) \oplus \perp).$$

Then we first did one step of unfolding on the right back-edge, and we took advantage of the two new facilities of the extended labelling:

1. we added three (Rec) rules, corresponding to the three ways for a branch of π_∞ to be valid, as summarized in the following array.

Shape of the branch	$A^* \cdot l^\omega$	$A^* \cdot r^\omega$	$l^* \cdot (r^+ \cdot l^+)^\omega$
Lowest (Rec) visited ∞^{ly}	b	a	c
Validating ν -formula	H	G	K

2. and so, we labelled the three formulas H , G and K at each corresponding (Rec), using for K the ability to label several occurrences at a time, and to label deeply ν -subformulas.

This indeed forms a correct labelling of π_∞ according to the extended labelling, hence ensuring their thread-validity.

The analysis leading to this choice of formulas is detailed in appendix D, p. 22. It allows to make finitary the derivation of fig. 9b, by expanding every formula as explained above, and by replacing every rule dealing with labels with an appropriate derivation, while leaving untouched the structure of rules not dealing with labels.

6 Conclusion

Summary of the contributions. In this paper, we contributed to the theory of circular proofs for μ MALL in two directions: (i) identifying fragments of circular proofs for which local conditions account for the validity of circular proof objects (in contrast to the global nature of thread conditions) and (ii) designing methods for translating circular proofs to finitary proofs (with explicit (co)induction rules). To do so, we introduced and studied several labelling systems, for circular proofs, or, more precisely, finite representation thereof, and made the following contributions:

- (i) First, we investigated how such labellings ensure validity of a labellable proof, turning a global and complex problem into a local and simpler one. Indeed, validity-checking is far from trivial in circular proof-theory for fixed-point logics, the best known bound for this problem being PSPACE. We provide two labellings, a simple and fairly restricted labelling discipline which forces back-edges to target (ν)-inferences and a more liberal one for which we only know that it ensures thread-validity.
- (ii) Second, we provided evidence on the usability of such labellings as a helpful guide in the generation of (co)inductive invariants which are necessary to translate a circular proof in a finitary proof system with (co)induction rules *à la* Park. We provided a full finitization method in a fairly restricted labelling system which contains at least all the translations of μ MALL proofs. However, this fragment is too constrained to treat standard examples that we discuss in the paper, and which contain most of the difficulties in finitizing circular proofs, namely: (i) interleaving of fixed-points and (ii) interleaving of back-edges resulting in various choices of a valid thread to support a branch.

Related and future works. We discuss related works as well as perspectives for pursuing this work along the above-mentioned directions:

Labelling and local certification is the basis of our approach. The idea of labelling μ -formulas to gather information on fixed-points unfoldings is naturally not new, already to be found in fixed-point approximation methods (see [14] for instance). The closest work in this direction is Stirling’s annotated proofs [25] and the application Afshari and Leigh [1] made of such proofs in obtaining completeness for the modal μ -calculus. Our labelling system works quite differently since only fixed-point operators are labelled while, in Stirling’s annotated proofs, every formula is labelled and labels are transmitted to immediate subformulas with a label extension on greatest fixed-points. Despite their difference, the relationships of those systems should be investigated further (in particular the role of the annotation restriction rule of Stirling’s system, def. 4 of [25]).

A less immediately connected topic is the connection between size-change termination (SCT) [21] and thread validity in μ -calculi: connections between those fields are not yet well understood despite early investigations by Dax *et al.*[14] for instance. More than a connection, this looks like an interplay: size-change termination is originally shown decidable by using Büchi automata and size-change graphs can be used to show validity of circular proofs [14]. There seems to be connections with our labelling system too.

In addition to investigating more closely those connections, we have several directions for improving our labelled proof system. The first task is to lift the results of section 3 to the extended labelling system. Indeed, for the more restricted fragment and given a circular proof presented as a graph with back-edges, we provided a method to effectively check that one can assign labels. It is therefore natural to expect extending these results to the relaxed framework. Another point we plan to investigate is whether every circular μ MALL proof can be labelled. Even though this can look paradoxical given the complexity of checking validity of circular proofs, one should keep in mind that it might well be the case that, in order to label a circular proof presented as a tree with back-edges, one has to unfold some of the back-edges, or possibly pick a different finite representation of the proof which may result in a space blow up. Related to this question is the connection of our labelling methods with size-change termination methods. Indeed, in designing the extended labelling, one gets closer to the kind of constructions one finds in SCT-based approaches: this should be investigated further since it may also be a key for our finitization objective. Note that the previous two directions would lead to a solution to the Brotherston-Simpson conjecture.

Finitization of circular proofs has been recently a very active topic with much research effort on solving Brotherston-Simpson's conjecture. The following recent contributions were made in the setting of Martin-Löf's inductive definitions: firstly, Berardi and Tatsuta proved [6] that, in general, the equivalence is false by providing a counter-example inspired by the *Hydra* paradox. Secondly, Simpson [24] on the one hand and Berardi and Tatsuta [7] on the other hand provided a positive answer in the restricted frameworks when the proof system contains arithmetics. While Simpson used tools from reverse mathematics and internalized circular proofs in ACA_0 , a fragment of second-order arithmetic with a comprehension axiom on arithmetical statements, Tatsuta and Berardi proved an equivalent result by a direct proof translation relying on an arithmetical version of the Ramsey and Podelsky-Rybalchenko theorems. A very natural question for future work is to extend the still *ad hoc* finitization method presented in the last section to the whole fragment of relaxed labelled proofs.

Circular proof search triggered interest compared to proof system with explicit inductive invariants (lacking subformula property). This has actually been turned to practice by Brotherston and collaborators [9]. We wish to investigate the potential use of labellings in circular proof-search. Indeed, there are several different labellings for a given finite derivation with back-edges where the labels are weakened. Prop. 21 characterizes least and greatest validating sets: those extremal validating sets correspond to different strategies in placing the labels, which have different properties with respect to the ability to form back-edges or to validate the proof that one may exploit in proof-search.

References

- 1 Bahareh Afshari and Graham E. Leigh. Cut-free completeness for modal mu-calculus. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005088.
- 2 David Baelde. On the proof theory of regular fixed points. In Martin Giese and Arild Waaler, editors, *Automated Reasoning with Analytic Tableaux and Related Methods, 18th International Conference, TABLEUX 2009, Oslo, Norway, July 6-10, 2009. Proceedings*, volume 5607 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2009. doi:10.1007/978-3-642-02716-1_8.
- 3 David Baelde. Least and greatest fixed points in linear logic. *ACM Transactions on Computational Logic (TOCL)*, 13(1):2, 2012.

- 4 David Baelde, Amina Doumane, and Alexis Saurin. Infinitary proof theory: the multiplicative additive case. 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 42:1–42:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.CSL.2016.42.
- 5 David Baelde and Dale Miller. Least and greatest fixed points in linear logic. In Nachum Dershowitz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference, LPAR 2007, Yerevan, Armenia, October 15-19, 2007, Proceedings*, volume 4790 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2007. doi:10.1007/978-3-540-75560-9_9.
- 6 Stefano Berardi and Makoto Tatsuta. Classical system of martin-löf’s inductive definitions is not equivalent to cyclic proof system. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 301–317, 2017. doi:10.1007/978-3-662-54458-7_18.
- 7 Stefano Berardi and Makoto Tatsuta. Equivalence of inductive definitions and cyclic proofs under arithmetic. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005114.
- 8 James Brotherston. *Sequent Calculus Proof Systems for Inductive Definitions*. PhD thesis, University of Edinburgh, 2006.
- 9 James Brotherston, Nikos Gorogiannis, and Rasmus Lerchedahl Petersen. A generic cyclic theorem prover. In *Programming Languages and Systems - 10th Asian Symposium, APLAS 2012, Kyoto, Japan, December 11-13, 2012. Proceedings*, volume 7705 of *Lecture Notes in Computer Science*, pages 350–367. Springer, 2012. doi:10.1007/978-3-642-35182-2_25.
- 10 James Brotherston and Alex Simpson. Complete sequent calculi for induction and infinite descent. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 51–62. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.16.
- 11 James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *J. Log. Comput.*, 21(6):1177–1216, 2011. doi:10.1093/logcom/exq052.
- 12 Pierre Clairambault. Least and greatest fixpoints in game semantics. In *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2009.
- 13 Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Arch. Math. Log.*, 28(3):181–203, 1989. doi:10.1007/BF01622878.
- 14 Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ -calculus. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2006. doi:10.1007/11944836_26.
- 15 Amina Doumane. *On the infinitary proof theory of logics with fixed points. (Théorie de la démonstration infinitaire pour les logiques à points fixes)*. PhD thesis, Paris Diderot University, France, 2017. URL: <https://tel.archives-ouvertes.fr/tel-01676953>.
- 16 Amina Doumane, David Baelde, Lucca Hirschi, and Alexis Saurin. Towards Completeness via Proof Search in the Linear Time μ -Calculus. Accepted for publication at LICS, 2016. URL: <https://hal.archives-ouvertes.fr/hal-01275289>.
- 17 Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL*

- 2013), *CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPICs*, pages 248–262. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. URL: <http://drops.dagstuhl.de/opus/portals/extern/index.php?semnr=13009>.
- 18 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.
 - 19 Roope Kaivola. A simple decision method for the linear time mu-calculus. In Jörg Desel, editor, *Structures in Concurrency Theory, Workshops in Computing*, pages 190–204. Springer London, 1995. doi:10.1007/978-1-4471-3078-9_13.
 - 20 Dexter Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.
 - 21 Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 81–92. ACM, 2001. doi:10.1145/360204.360210.
 - 22 David Park. Fixpoint induction and proofs of program properties. *Machine intelligence*, 5(59-78):5–3, 1969.
 - 23 Luigi Santocanale. A calculus of circular proofs and its categorical semantics. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2002. doi:10.1007/3-540-45931-6_25.
 - 24 Alex Simpson. Cyclic arithmetic is equivalent to peano arithmetic. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 283–300, 2017. doi:10.1007/978-3-662-54458-7_17.
 - 25 Colin Stirling. A tableau proof system with names for modal mu-calculus. In *HOWARD-60: A Festschrift on the Occasion of Howard Barringer's 60th Birthday*, volume 42 of *EPiC Series in Computing*, pages 306–318. EasyChair, 2014. URL: <http://www.easychair.org/publications/?page=1932281032>.
 - 26 Igor Walukiewicz. On completeness of the mu-calculus. In *LICS*, pages 136–146. IEEE Computer Society, 1993.
 - 27 Igor Walukiewicz. Completeness of Kozen's axiomatisation of the propositional mu-calculus. In *Proceedings, 10th Annual IEEE Symposium on Logic in Computer Science, San Diego, California, USA, June 26-29, 1995*, pages 14–24. IEEE Computer Society, 1995. doi:10.1109/LICS.1995.523240.

A Proofs of section 3

► **Lemma 31.** *Let b be an infinite branch in a finite, circular representation, i.e. an infinite ascending path from the root of a tree with back-edges. There is a vertex s in the tree, i.e. an occurrence of sequent in the representation, which is the lowest one infinitely appearing on b . Moreover, this vertex / occurrence of sequent is the target of a back-edge.*

Proof. This comes only for the tree-with-back-edges structure and does not rely on the proof structure. The crucial fact to notice is that in a tree, if S is a non empty, finite set of vertices that is connected for the relation of comparability, i.e. if $\forall v, v' \in S, v \leq v'$ or $v' \leq v$, then S has a minimum. This is proved by induction on the cardinal of S . Take then for S the set of vertices appearing infinitely on the branch b , and you get a vertex v , which is the desired

vertex. In particular, when v is accessed in b from another infinitely appearing vertex, it has to be via a back-edge. ◀

► **Lemma 32** (Follow-up of labels). *If u is a path in a labelled circular representation, if u does not cross the rule $(\nu_b(a))$, and if p is a position in the target sequent of u (its top sequent) that is labelled with a , then $\mathfrak{t}(u)(p)$ is defined and is a position labelled with a in the source sequent of u (its bottom sequent).*

Proof. This is quite straightforward, by induction on the length of u , and by looking at the first (or the last) rule crossed by u . We use notably the fact that, when the induced thread $\mathfrak{T}(u)(p)$ is followed top-down, the label a cannot be erased because we do not cross $(\text{Rec}(a))$ and the thread cannot reach a cut-formula because cut-formulas do not contain labels. ◀

► **Proposition 15.** *Every pre-proof of μMALL^ω that is the image of a proof in $\mu\text{MALL}_{\text{lab}}^\omega$ by label-erasing and total unfolding satisfies thread validity.*

Proof. Suppose π is a labelled circular representation.

- Let $[\pi]$ be its erasure. $[\pi]$ is thus a circular representation of a μMALL^ω preproof.
- Suppose b an infinite branch of $[\pi]$, that is an infinite ascending path in the tree-with-back-edges $[\pi]$, starting from the root.
- Let b_0 be the corresponding infinite branch in π .
- Let S_0 be the occurrence of sequent in π which is the lowest back-edge target infinitely often crossed by b_0 (lemma 31). Being the target of some back-edge(s), S_0 is the conclusion of a $(\nu_b(a))$ rule, which unfolds some $\nu X.A$.
- This implies that b_0 is of the form $b_0 = r \xrightarrow[u_0]{*} S_0 \xrightarrow[u_1]{*} S_1 \xrightarrow[\text{be}]{\rightarrow} S_0 \xrightarrow[u_2]{*} S_2 \xrightarrow[\text{be}]{\rightarrow} S_0 \cdots$ where r is the root of π and where the u_i s do not cross S_0 except at their sources.
- Let $p_0 = (0, \epsilon)$ be the position of the principal formula $\nu X.A$ in S_0 .
- Remark that, because of the existence of back-edges from every S_{i+1} to S_0 , all S_i s are identical sequents, except for the fact that a does not appear in S_0 whereas it appears at the only position p_0 in S_{i+1} .
- Now remark that for $i \geq 1$: $\mathfrak{T}(u_i)(p_0)$ is a ν -thread in u_i , its target is p_0 in S_i , which is labelled with a , in the occurrence of sequent just above S_0 , *i. e.* in the premise of $\nu_b(a)$, it goes through a position labelled with a (lemma 32), hence a position of $\nu X.A$ in the unfolding $A[\nu X.A]$, therefore, according to the definition of \mathfrak{T} , as described on Figure 6, p. 6, the source of $\mathfrak{T}(u_i)(p_0)$ is again the position p_0 of the main formula $\nu X.A$ in S_0 . To sum up: $\mathfrak{T}(u_i)(p_0)$ is a thread $(S_0, p_0) \xrightarrow[\mathfrak{T}(u_i)(p_0)]{*} (S_1, p_0)$, and it is progressing, because its source is the principal conclusion of the rule $(\nu_b(a))$.
- By glueing the $\mathfrak{T}(u_i)(p_0)$ together, we get an infinite thread

$$(S_0, p_0) \xrightarrow[\mathfrak{T}(u_1)(p_0)]{*} (S_1, p_0) \xrightarrow[\text{be}]{\rightarrow} (S_0, p_0) \xrightarrow[\mathfrak{T}(u_2)(p_0)]{*} (S_2, p_0) \xrightarrow[\text{be}]{\rightarrow} (S_0, p_0) \cdots$$

This thread is valid because every $\mathfrak{T}(u_i)(p_0)$ is progressing. And it is indeed a thread of $b_0 = r \xrightarrow[u_0]{*} S_0 \xrightarrow[u_1]{*} S_1 \xrightarrow[\text{be}]{\rightarrow} S_0 \xrightarrow[u_2]{*} S_2 \xrightarrow[\text{be}]{\rightarrow} S_0 \cdots$. Hence b_0 is valid, what was to be demonstrated. ◀

► **Proposition 16.** *μMALL proofs can be translated to μMALL^ω .*

Proof. The target of the usual translation $\mu\text{MALL} \rightarrow \mu\text{MALL}^\omega$ is included in μMALL^ω . See key case of the translation on figure 11. ◀

$$\frac{\frac{\vdash A[B], B^\perp \quad \vdash B, \Gamma}{\vdash \nu X.A, \Gamma} \nu_{\text{inv}}}{\vdash \nu X.A, \Gamma} \equiv \frac{\frac{\frac{\frac{\frac{}{\vdash \nu^a X.A, B^\perp} \succ(a)}{\vdash A[\nu^a X.A], A[B]^\perp} [A]}{\vdash A[\nu^a X.A], B^\perp} \nu_b(a)}{\vdash \nu X.A, B^\perp} \text{cut}}{\vdash \nu X.A, \Gamma} \text{cut}}{\vdash \nu X.A, \Gamma} \text{cut}}{\vdash \nu X.A, \Gamma} \text{cut}} \text{cut}$$

■ **Figure 11** translation $\mu\text{MALL} \rightarrow \mu\text{MALL}_{\text{lab}}^\triangleright$.

► **Proposition 22.** *A finite representation π of a μMALL^ω pre-proof is a representation of a $\mu\text{MALL}_{\text{lab}}^\triangleright$ proof iff every occurrence of a ν -rule of π has a validating set.*

Proof. Let us assume that every ν rule of π has a validating set. There is a finite number of ν rules in the representation; we choose a we label them with distinct variables a_1, \dots, a_n , in a way such that if the ν rule labelled by a_i is below the rule labelled by a_j in the representation then $i \leq j$. We denote by L_i a validating set for $\nu(a_i)$. We then do the following for each i , going from 1 to n : for each occurrence of ν -formula $\nu^V X.A$ that is at a position belonging to L_i , add the variable a_i to V , that is replace this occurrence of $\nu^V X.A$ with $\nu^{V, a_i} X.A$. By doing this it may happen that we break the validity of some rules of the representation: because L_i , although downward closed, is in general not upward closed, so we may end with the following situation:

$$\frac{\frac{\vdash A, C[\nu^V X.D] \quad \vdash A, C[\nu^V X.D]}{\vdash A \& B, C[\nu^V X.D]} \& \text{ becoming } \frac{\frac{\vdash A, C[\nu^{V, a} X.D] \quad \vdash B, C[\nu^V X.D]}{\vdash A \& B, C[\nu^{V, a} X.D]} \& \text{ which}$$
 is not anymore a valid rule. We then patch this by adding as many (LW) rules as needed on the premises:

$$\frac{\frac{\frac{\vdash A, C[\nu^{V, a} X.D]}{\vdash A, C[\nu^{V, a} X.D]} \text{ (LW)} \quad \frac{\vdash B, C[\nu^V X.D]}{\vdash B, C[\nu^{V, a} X.D]} \&}{\vdash A \& B, C[\nu^{V, a} X.D]} \&$$

Similarly it may happen that the source of a back-edge get a bigger labelling than the target of this back-edge; we patch this by adding (LW) rules under the source sequent of the back-edge. When this operation has been done for every i , from 1 to n , we obtain a validly labelled proof of $\mu\text{MALL}_{\text{lab}}^\triangleright$.

Conversely, let π_0 be a $\mu\text{MALL}_{\text{lab}}^\triangleright$ representation such that $\pi = |\pi_0|$. Up to renaming, we can assume that all (ν_b) rules of π_0 are labelled with distinct variables. For every (ν) rule occurrence in π , consider the corresponding ($\nu_b(a)$) rule in π_0 and let L_a be the set of all occurrences of ν -formulas in π_0 that carry the variable a in their labelling. The constraints on the labelling of $\mu\text{MALL}_{\text{lab}}^\triangleright$ proof precisely get L_a to be a validating set for the considered occurrence of (ν_b) in π . ◀

B Details and proofs for section 3.3

We illustrate the construction of the edges of the graph defined in definition 18 with the the following examples in which we have indexed the apparent ν -formulas by numbers representing vertices of the graph:

$$\frac{\frac{\frac{\vdash \nu_1 X.X, \nu_2 X.X \quad \vdash \mathbf{1} \oplus \nu_3 X.X}{\vdash \nu_4 X.X \otimes (\mathbf{1} \oplus \nu_5 X.X), \nu_6 X.X} \otimes}{\vdash \nu_1 X.X, (\mathbf{1} \oplus \nu_2 X.X), \nu_3 X.X} \otimes}{\frac{\frac{\vdash \nu_4 X.X \wp (\mathbf{1} \oplus \nu_5 X.X), \nu_6 X.X}{\vdash (\nu_4 Y.(\nu_5 X.(\nu_6 Y.X) \otimes X)) \otimes \nu_7 X.(\nu_8 Y.X) \otimes X, \nu_9 X.X} \wp}{\vdash \nu_1 X.(\nu_2 Y.X) \otimes X, \nu_3 X.X} \wp} \nu} \nu$$

induces edges $1 \rightarrow 4, 2 \rightarrow 6, 3 \rightarrow 5$,
induces edges $1 \rightarrow 4, 2 \rightarrow 5, 3 \rightarrow 6$ and
induces edges $4 \rightarrow 2, 6 \rightarrow 2, 8 \rightarrow 2, 5 \rightarrow 1, 7 \rightarrow 1, 9 \rightarrow 3$. Moreover, if the conclusion of this last rule is the target of a back-edge whose source is $\vdash \nu_{10} X.(\nu_{11} Y.X) \otimes X, \nu_{12} X.X$ then this back-edge also induces edges $1 \rightarrow 10, 2 \rightarrow 11, 3 \rightarrow 12$.

In the case of a cut formula, the formula has no corresponding formula in the conclusion sequent and in this case it induces an outgoing edge, pointing to the extra vertex \perp :

$$\frac{\frac{\vdash \nu_2 X.X \quad \vdash \mu X.X, \nu_3 X.X}{\vdash \nu_1 X.X} \text{cut}}{\text{induces edges } 2 \rightarrow \perp, 3 \rightarrow 1.}$$

► **Proposition 25.** *A finite representation π of a μMALL^ω pre-proof is strongly valid iff every ν -rule of π has a validating set iff it is the representation of a $\mu\text{MALL}_{\text{lab}}^\omega$ proof.*

Proof. The second equivalence is prop. 22, so that we need to check the first one:

Let us assume that π has a validating set. Let us consider one occurrence $\frac{\vdash A[\nu X.A], \Gamma}{\vdash \nu X.A, \Gamma}$ of a ν -rule in π and a path u in the subgraph above this ν -rule, going down, from the source of a back-edge targetting this ν -rule, to the ν -rule itself, ending by this ν -rule. u has then premise and conclusion equals to $\vdash \nu X.A, \Gamma$.

Let us denote by L a validating set of this (ν)-rule occurrence, and let us denote by t the maximal thread going down in u starting from the main $\nu X.A$ in its premise. This occurrence of $\nu X.A$ is in L , because L is a validating set. Then, because L is downward closed, all vertices of t are in L . Therefore the lowest vertex of t , which is a position in the $\vdash \nu X.A, \Gamma$ conclusion of the considered ν -rule, or \perp , is also in L . But in this last sequent occurrence, the only position that is in L is the one of the main $\nu X.A$, which is consequently the end point of t .

Conversely, let us consider an occurrence of a (ν)-rule in π , whose conclusion has the form $\vdash \nu X.A, \Gamma$, and let us assume that it has no validating set. It is, by prop. 21, equivalent to say that there is a path t such that:

- t stays above the considered occurrence of (ν)-rule;
 - t goes down from the source $\nu X.A, \Gamma$ of a back-edge targetting the (ν)-rule we consider, to the conclusion $\nu X.A, \Gamma$ of this (ν)-rule;
 - t starts from the main $\nu X.A$ of its premise;
 - t ends either on a cut-formula or on a position that is not the principal $\nu X.A$.
- u therefore violates strong validity (def. 24). ◀

C Details and proofs for section 5

Remember that this proposition is about the extended labelling of def. 29:

► **Proposition 30.** *If π is an extended labelled circular representation then $[\pi]$ is a circular representation of a valid μMALL^ω proof.*

Proof. First remark that lemma 32, as it is stated on p. 19, still holds for this extended labelling. The proof is the same as before, bearing in mind to replace every mention of $(\nu_b(a))$ with $(\text{Rec}(a))$. As for the previous labelling, the proof of this proposition crucially rely on it.

Suppose π is a labelled circular representation. Let $[\pi]$ be its erasure. $[\pi]$ is thus a circular representation of a μ MALL $^\omega$ preproof. Suppose b an infinite branch of $[\pi]$, that is an infinite ascending path in the tree-with-back-edges $[\pi]$, starting from the root. Let b_0 be the corresponding infinite branch in π . Let S_0 be the occurrence of sequent in π which is the lowest back-edge target infinitely often crossed by b_0 . Being the target of some back-edge(s), S_0 is the premise of a (Rec(a)) rule, for some variable a .

This implies that b_0 is of the form $b_0 = r \xrightarrow[u_0]{*} S_0 \xrightarrow[u_1]{*} S_1 \xrightarrow[\text{be}]{\rightarrow} S_0 \xrightarrow[u_2]{*} S_2 \xrightarrow[\text{be}]{\rightarrow} S_0 \cdots$ where r is the root of π and where the u_i do not cross S_0 except at their sources.

Remark that the positions labelled by a are the same in all S_i , as there are back-edges from every S_{i+1} to S_0 . The difference, however, is that these positions are labelled with $a-$ in S_0 and with $a+$ in every S_{i+1} . Let P_0 be the set of those positions. P_0 is finite and non empty. Now we would like, as in the proof of prop. 15, to construct an infinite thread along b_0 . However, because P_0 may contain more than one element, we cannot know by advance, for each S_i , which $p \in P_0$ will support an infinite thread. Thus, we will use König's lemma to show the existence of such a thread. Let T_0 be the tree whose vertices are the pairs (i, p) where $1 \leq i < \omega$ and $p \in P_0$, whose roots are the vertices of the form $(1, p)$ and where, for $i > 1$, the father of (i, p) is² $(i-1, t(u_i)(p))$. Here we have to prove that $t(u_i)(p)$ is defined and that it belongs to P_0 for every i and $p \in P_0$. This is ensured by lemma 32 thanks to the labels.

Remark that every edge in T_0 induces a progressing thread. Indeed, for $i \geq 1$ and $p \in P_0$:

- $\mathfrak{T}(u_i)(p)$ is a ν -thread in u_i ,
- its target is p in S_i , which is labelled with $a+$
- and its source is p in S_0 , which is labelled with $a-$.

An examination of the rules that may compose u_i shows that the only way for that to be true is that $\mathfrak{T}(u_i)(p)$ is progressing. Now T_0 is an infinite tree with a finite number of roots and an arity bounded by $\mathbf{Card}(P_0)$, hence, by König's lemma, it has an infinite branch $(1, p_1) \leftarrow (2, p_2) \leftarrow (3, p_3) \cdots$.

This infinite branch induces in turn an infinite thread

$$(S_0, p_0) \xrightarrow[\mathfrak{T}(u_1)(p_1)]{*} (S_1, p_1) \xrightarrow[\text{be}]{\rightarrow} (S_0, p_1) \xrightarrow[\mathfrak{T}(u_2)(p_2)]{*} (S_2, p_2) \xrightarrow[\text{be}]{\rightarrow} (S_0, p_2) \cdots$$

This thread is valid because every $\mathfrak{T}(u_i)(p_i)$ is progressing. And it is indeed a thread of $b_0 = r \xrightarrow[u_0]{*} S_0 \xrightarrow[u_1]{*} S_1 \xrightarrow[\text{be}]{\rightarrow} S_0 \xrightarrow[u_2]{*} S_2 \xrightarrow[\text{be}]{\rightarrow} S_0 \cdots$. Hence b_0 is valid, what was to be demonstrated. \blacktriangleleft

D Details of finitization for π_∞

To finitize π_∞ we try to apply the same method as for the example (8) p. 11, by expanding every labelled formula to a non-labelled one and expanding the rules that need it to match these transform. This works perfectly for H and G , which appear respectively as formulas of the premises (Rec(b)) and (Rec(a)). But the situation is more delicate for K for which we have to face a double difficulty: in the premise of (Rec(c)), K is not a formula of the sequent but a subformula, and it appears in two different formulas.

Let us try to transform this situation into one that would fit our method. First we would like to have only one formula containing K instead of the two I and J . Unfortunately, none of them can be unlabelled without breaking the labelling. Fortunately the solution to that is easy: I, J is simply equivalent to $L := I \wp J$.

² Recall that $t(u)$ and $\mathfrak{T}(u)$ are defined in defs. 6 and 8, p. 6.

Now we would like $I \wp J$ to be a ν -formula that we could label. We already made use, in the previous example, of the isomorphism $A[\nu X.B[A[X]]] \simeq \nu X.A[B[X]]$ (*)

to turn an almost- ν -formula into a real one. Let us apply that again.

The formula $L = I \wp J$ is equal to $L'[K]$ where $L'[Y] := I'[Y] \wp J'[Y]$, that is: $L = L'[\nu Y.I'[Y]]$. In order to apply an isomorphism of the form (*) we would like $I'[Y]$ to be of the form $M'[L'[Y]]$ for a given M' . This is unfortunately not the case as $I'[Y]$ is a subformula of $L'[Y]$. However, a careful examination of the flow of I , J and K along the loops of π_∞ makes apparent the fact that

$$I'[Y] = \mu Z.((Z \wp J'[Y]) \oplus \perp) \simeq \mu _ .((I'[Y] \wp J'[Y]) \oplus \perp) = M'[L'[Y]]$$

where $M'[Y]$ is defined to be $\mu _ .(Y \oplus \perp)$, in which we use the notation $\mu _ .A$ to denote a $\mu X.A$ with X not appearing free in A . This degenerate μ binder could be removed to simplify the formulas involved in the finitisation, but we keep it to stay as close as possible to the original structure of I , trying to preserve its head connective.

When we stick all that together we get $L = I \wp J \simeq L'[\nu Y.M'[L'[Y]]] \simeq \nu Y.L'[M'[Y]]$ which is a ν -formula that we know, when labelled, how to expand into an unlabelled formula. If we stopped here our analysis, we would then define:

$$C := F \wp G \wp H \quad L^{c-} := \nu Y.L'[M'[C^\perp \oplus Y]] \quad L^{c+} := C^\perp \oplus L^{c-}.$$

However we will do yet a bit more work in order to get the structure of L^{c-} closer to L 's one.

Indeed the isomorphism (*) can be used in the other direction:

$$\nu Y.L'[M'[C^\perp \oplus Y]] \simeq L'[\nu Y.M'[C^\perp \oplus L'[Y]]] = I'[\nu Y.M'[C^\perp \oplus L'[Y]]] \wp J'[\nu Y.M'[C^\perp \oplus L'[Y]]].$$

This, finally, leads us to define: $C := F \wp G \wp H$ $K^{c-} := \nu Y.M'[C^\perp \oplus L'[Y]]$ which allows to expand $I'[K^{c-}]$ and $J'[K^{c-}]$. On the other hand, this is not sufficient to define an expansion of K^{c+} , and we still need an *ad hoc* treatment for formulas containing it:

$$"I'[K^{c+}]" := I^{c+} := M'[C^\perp \oplus L'[K^{c-}]] \quad "I'[K^{c+}] \wp J'[K^{c+}]" := L^{c+} := C^\perp \oplus L'[K^{c-}]$$

With these expansions of labelled formulas into unlabelled formulas, we can finitize the derivation of fig. 9a into the very close derivation of fig. 9b, on which the rules dealing with labelling can be expanded into μ MALL derivations.

Parity Games with Weights

Sven Schewe¹

University of Liverpool, Liverpool L69 3BX, United Kingdom
sven.schewe@liverpool.ac.uk

Alexander Weinert²

Reactive Systems Group, Saarland University, 66123 Saarbrücken, Germany
weinert@react.uni-saarland.de

Martin Zimmermann

Reactive Systems Group, Saarland University, 66123 Saarbrücken, Germany
zimmermann@react.uni-saarland.de

Abstract

Quantitative extensions of parity games have recently attracted significant interest. These extensions include parity games with energy and payoff conditions as well as finitary parity games and their generalization to parity games with costs. Finitary parity games enjoy a special status among these extensions, as they offer a native combination of the qualitative and quantitative aspects in infinite games: the quantitative aspect of finitary parity games is a quality measure for the qualitative aspect, as it measures the limit superior of the time it takes to answer an odd color by a larger even one. Finitary parity games have been extended to parity games with costs, where each transition is labelled with a non-negative weight that reflects the costs incurred by taking it. We lift this restriction and consider parity games with costs with arbitrary integer weights. We show that solving such games is in $NP \cap CO-NP$, the signature complexity for games of this type. We also show that the protagonist has finite-state winning strategies, and provide tight exponential bounds for the memory he needs to win the game. Naturally, the antagonist may need infinite memory to win. Finally, we present tight bounds on the quality of winning strategies for the protagonist.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Infinite Games, Quantitative Games, Parity Games

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.36

Related Version Full version available online [28], <https://arxiv.org/abs/1804.06168>.

1 Introduction

Finite games of infinite duration offer a wealth of challenges and applications that has garnered to a lot of attention. The traditional class of games under consideration were games with a simple parity [19, 12, 11, 21, 2, 31, 15, 16, 29, 18, 25, 27, 26, 3, 17, 13, 20] or payoff [24, 32, 15, 1, 27] objective. These games form a hierarchy with very simple tractable reductions from parity games through mean payoff games [24, 32, 15, 1, 27] and discounted payoff games [32, 15, 27] to simple stochastic games [9].

¹ Supported by the EPSRC projects “Energy Efficient Control” (EP/M027287/1) and “Solving Parity Games in Theory and Practice” (EP/P020909/1).

² Supported by the project “TriCS” (ZI 1516/1-1) of the German Research Foundation (DFG) and the Saarbrücken Graduate School of Computer Science.



More recently, games with a mixture of the qualitative parity condition and further quantitative objectives have been considered, including mean payoff parity games [8] and energy parity games [4]. Finitary parity games [7] take a special role within the class of games with mixed parity and payoff objectives. To win a finitary parity game, Player 0 needs to enforce a play with a bound b such that almost all occurrences of an odd color are followed by a higher even color within at most b steps.

This is interesting, because it provides a natural link between the qualitative and quantitative objective. One aspect that attracted attention is that, as long as one is not interested in optimizing the bound b , these games are the only games of the lot that are known to be tractable [7]. However, the bound b itself is also interesting: It serves as a native quality measure, because it limits the response time [30].

This property calls for a generalization to different cost models, and a first generalization has been made with the introduction of parity games with costs [14]. In parity games with costs, the basic cost function of finitary parity games – where each step incurs the same cost – is replaced with different non-negative costs for different edges. In this paper, we generalize this further to *general* integer costs: We decorate the edges with integer weights. The quantitative aspect in these parity games with weights consists of having to answer almost all odd colors by a higher even color, such that the *absolute* value of the weight of the path to this even color is bounded by a bound b .

In addition to their conceptual charm, we show that parity games with weights are PTIME equivalent to energy parity games. This indicates that these games are part of a natural complexity class, whereas the games with a plain objective appear to form a hierarchy. We use the reduction from parity games with weights to energy parity games to solve them. This reduction goes through intermediate reductions to and from *bounded* parity games with weights. These games have the additional restriction that the limit superior of the absolute weight of initial sequences of unanswered requests in a play is finite. These bounded parity games with weights are then reduced to energy parity games. The other direction of the reduction is through simple gadgets that preserve the main elements of winning strategies in games that are extended in two steps by very simple gadgets. As a result, we obtain the same complexity results for parity games with weights as for energy parity games, i.e., $\text{NP} \cap \text{CO-NP}$, the signature complexity for finite games of infinite duration with parity conditions and their extensions. Thereby, we obtain an argument that these games might be representatives of a natural complexity class, lending a further argument for the relevance of two player games with mixed qualitative and quantitative winning conditions. Furthermore, Daviaud et al. recently showed that parity games with weights can even be solved in pseudo-quasi-polynomial time [10].

Naturally, parity games with weights subsume parity games (as a special case where all weights are zero), finitary parity games (as a special case where all weights are positive), and parity games with costs (as a special case where all weights are non-negative).

Finally, we show that the protagonist has finite-state winning strategies, and provide tight exponential bounds for the memory he needs to win the game. We also present tight bounds on the quality of winning strategies for the protagonist. Naturally, the antagonist may need infinite memory to win.

2 Preliminaries

We denote the non-negative integers by \mathbb{N} , the integers by \mathbb{Z} , and define $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$. As usual, we have $\infty > n$, $-\infty < n$, $n + \infty = \infty$, and $-\infty - n = -\infty$ for all $n \in \mathbb{Z}$.

An **arena** $\mathcal{A} = (V, V_0, V_1, E)$ consists of a finite, directed graph (V, E) and a partition $\{V_0, V_1\}$ of V into the positions of Player 0 (drawn as ellipses) and Player 1 (drawn as rectangles). The size of \mathcal{A} , denoted by $|\mathcal{A}|$, is defined as $|V|$. A **play** in \mathcal{A} is an infinite path $\rho = v_0v_1v_2\cdots$ through (V, E) . To rule out finite plays, we require every vertex to be non-terminal. We define $|\rho| = \infty$. Dually, for a finite play prefix $\pi = v_0\cdots v_j$ we define $|\pi| = j + 1$.

A **game** $\mathcal{G} = (\mathcal{A}, \text{Win})$ consists of an arena \mathcal{A} with vertex set V and a set $\text{Win} \subseteq V^\omega$ of winning plays for Player 0. The set of winning plays for Player 1 is $V^\omega \setminus \text{Win}$. A winning condition Win is 0-extendable if, for all $\rho \in V^\omega$ and all $w \in V^*$, $\rho \in \text{Win}$ implies $w\rho \in \text{Win}$. Dually, Win is 1-extendable if, for all $\rho \in V^\omega$ and all $w \in V^*$, $\rho \notin \text{Win}$ implies $w\rho \notin \text{Win}$.

A **strategy** for Player $i \in \{0, 1\}$ is a mapping $\sigma: V^*V_i \rightarrow V$ such that $(v, \sigma(wv)) \in E$ holds true for all $wv \in V^*V_i$. We say that σ is **positional** if $\sigma(wv) = \sigma(v)$ holds true for every $wv \in V^*V_i$. A play $v_0v_1v_2\cdots$ is **consistent** with a strategy σ for Player i , if $v_{j+1} = \sigma(v_0\cdots v_j)$ holds true for every j with $v_j \in V_i$. A strategy σ for Player i is a **winning strategy** for \mathcal{G} from $v \in V$ if every play that starts in v and is consistent with σ is won by Player i . If Player i has a winning strategy from v , then we say Player i wins \mathcal{G} from v . The **winning region** of Player i is the set of vertices, from which Player i wins \mathcal{G} ; it is denoted by $\mathcal{W}_i(\mathcal{G})$. **Solving** a game amounts to determining its winning regions. If $\mathcal{W}_0(\mathcal{G}) \cup \mathcal{W}_1(\mathcal{G}) = V$, then we say that \mathcal{G} is **determined**.

Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena and let $X \subseteq V$. The i -attractor of X is defined inductively as $\text{Attr}_i(X) = \text{Attr}_i^{|V|}(X)$, where $\text{Attr}_i^0(X) = X$ and

$$\begin{aligned} \text{Attr}_i^j(X) = & \text{Attr}_i^{j-1}(X) \cup \{v \in V_i \mid \exists v' \in \text{Attr}_i^{j-1}(X). (v, v') \in E\} \\ & \cup \{v \in V_{1-i} \mid \forall (v, v') \in E. v' \in \text{Attr}_i^{j-1}(X)\} . \end{aligned}$$

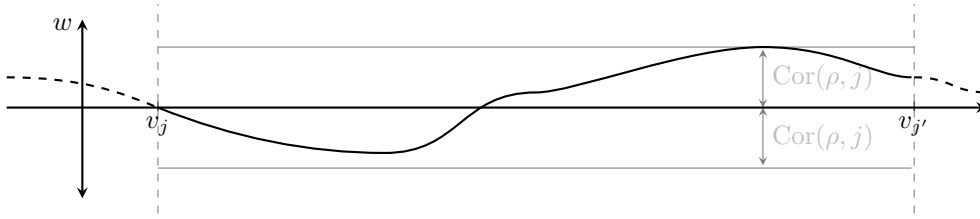
Hence, $\text{Attr}_i(X)$ is the set of vertices from which Player i can force the play to enter X : Player i has a positional strategy σ_X such that each play that starts in some vertex in $\text{Attr}_i(X)$ and is consistent with σ_X eventually encounters some vertex from X . We call σ_X an attractor strategy towards X . Moreover, the i -attractor can be computed in time linear in $|E|$ [23]. When we want to stress the arena \mathcal{A} the attractor is computed in, we write $\text{Attr}_i^{\mathcal{A}}(X)$.

A set $X \subseteq V$ is a trap for Player i , if every vertex in $X \cap V_i$ has only successors in X and every vertex in $X \cap V_{1-i}$ has at least one successor in X . In this case, Player $1 - i$ has a positional strategy τ_X such that every play starting in some vertex in X and consistent with τ_X never leaves X . We call such a strategy a trap strategy.

► **Remark 1.**

1. The complement of an i -attractor is a trap for Player i .
2. If X is a trap for Player i , then $\text{Attr}_{1-i}(X)$ is also a trap for Player i .
3. If Win is i -extendable and $(\mathcal{A}, \text{Win})$ determined, then $\mathcal{W}_{1-i}(\mathcal{A}, \text{Win})$ is a trap for Player i .

A **memory structure** $\mathcal{M} = (M, \text{init}, \text{upd})$ for an arena (V, V_0, V_1, E) consists of a finite set M of memory states, an initialization function $\text{init}: V \rightarrow M$, and an update function $\text{upd}: M \times E \rightarrow M$. The update function can be extended to finite play prefixes in the usual way: $\text{upd}^+(v) = \text{init}(v)$ and $\text{upd}^+(wv'v') = \text{upd}(\text{upd}^+(wv), (v, v'))$ for $w \in V^*$ and $(v, v') \in E$. A next-move function $\text{Nxt}: V_i \times M \rightarrow V$ for Player i has to satisfy $(v, \text{Nxt}(v, m)) \in E$ for all $v \in V_i$ and $m \in M$. It induces a strategy σ for Player i with memory \mathcal{M} via $\sigma(v_0\cdots v_j) = \text{Nxt}(v_j, \text{upd}^+(v_0\cdots v_j))$. A strategy is called **finite-state** if it can be implemented by a memory structure. We define $|\mathcal{M}| = |M|$. Slightly abusively, we say that the size of a finite-state strategy is the size of a memory structure implementing it.



■ **Figure 1** The cost-of-response of some request posed by visiting vertex v_j , which is answered by visiting vertex $v_{j'}$.

3 Parity Games with Weights

Fix an arena $\mathcal{A} = (V, V_0, V_1, E)$. A **weighting** for \mathcal{A} is a function $w: E \rightarrow \mathbb{Z}$. We define $w(\varepsilon) = w(v) = 0$ for all $v \in V$ and extend w to sequences of vertices of length at least two by summing up the weights of the traversed edges. Given a play (prefix) $\pi = v_0v_1v_2 \cdots$, we define the amplitude of π as $\text{Ampl}(\pi) = \sup_{j < |\pi|} |w(v_0 \cdots v_j)| \in \mathbb{N}_\infty$.

A **coloring** of V is a function $\Omega: V \rightarrow \mathbb{N}$. The classical parity condition requires almost all occurrences of odd colors to be answered by a later occurrence of a larger even color. Hence, let $\text{Ans}(c) = \{c' \in \mathbb{N} \mid c' \geq c \text{ and } c' \text{ is even}\}$ be the set of colors that “answer” a “request” for color c . We denote a vertex v of color c by v/c .

Fijalkow and Zimmermann introduced a generalization of the parity condition and the finitary parity condition [7], the parity condition with costs [14]. There, the edges of the arena are labeled with *non-negative weights* and the winning condition demands that there exists a bound b such that almost all requests are answered with weight at most b , i.e., the weight of the infix between the request and the response has to be bounded by b .

Our aim is to extend the parity condition with costs by allowing for the full spectrum of weights to be used, i.e., by also incorporating negative weights. In this setting, the weight of an infix between a request and a response might be negative. Thus, the extended condition requires the weight of the infix to be bounded from above and from below.³ To distinguish between the parity condition with costs and the extension introduced here, we call our extension the parity condition with weights.

Formally, let $\rho = v_0v_1v_2 \cdots$ be a play. We define the cost-of-response at position $j \in \mathbb{N}$ of ρ by

$$\text{Cor}(\rho, j) = \min\{\text{Ampl}(v_j \cdots v_{j'}) \mid j' \geq j, \Omega(v_{j'}) \in \text{Ans}(\Omega(v_j))\}$$

where we use $\min \emptyset = \infty$. As the amplitude of an infix only increases by extending the infix, $\text{Cor}(\rho, j)$ is the amplitude of the shortest infix that starts at position j and ends at an answer to the request posed at position j . We illustrate this notion in Figure 1.

We say that a request at position j is answered with cost b , if $\text{Cor}(\rho, j) = b$. Consequently, a request with an even color is answered with cost zero. The cost-of-response of an unanswered request is infinite, even if the amplitude of the remaining play is bounded. In particular, this means that an unanswered request at position j may be “unanswered with finite cost b ” (if the amplitude of the remaining play is $b \in \mathbb{N}$) or “unanswered with infinite cost” (if the amplitude of the remaining play is infinite). In either case, however, we have $\text{Cor}(\rho, j) = \infty$.

³ We discuss other possible interpretations of negative weights in Section 9.

We define the parity condition with weights as

$$\text{WeightParity}(\Omega, w) = \{\rho \in V^\omega \mid \limsup_{j \rightarrow \infty} \text{Cor}(\rho, j) \in \mathbb{N}\} .$$

I.e., ρ satisfies the condition if and only if there exists a bound $b \in \mathbb{N}$ such that almost all requests are answered with cost less than b . In particular, only finitely many requests may be unanswered, even with finite cost. Note that the bound b may depend on the play ρ .

We call a game $\mathcal{G} = (\mathcal{A}, \text{WeightParity}(\Omega, w))$ a parity game with weights, and we define $|\mathcal{G}| = |\mathcal{A}| + \log(W)$, where W is the largest absolute weight assigned by w ; i.e., we assume weights to be encoded in binary. If w assigns zero to every edge, then $\text{WeightParity}(\Omega, w)$ is a classical (max-) parity condition, denoted by $\text{Parity}(\Omega)$. Similarly, if w assigns positive weights to every edge, then $\text{WeightParity}(\Omega, w)$ is equal to the finitary parity condition over Ω , as introduced by Chatterjee and Henzinger [6]. Finally, if w assigns only non-negative weights, then $\text{WeightParity}(\Omega, w)$ is a parity condition with costs, as introduced by Fijalkow and Zimmermann [14]. In these cases, we refer to \mathcal{G} as a parity game, a finitary parity game, or a parity game with costs, respectively. We recall the characteristics of these games in Table 1 on Page 15.

4 Solving Parity Games with Weights

We now show how to solve parity games with weights. Our approach is inspired by the classic work on finitary parity games [7] and parity games with costs [14]: We first define a stricter variant of these games, which we call bounded parity games with weights, and then show two reductions:

- parity games with weights can be solved in polynomial time with oracles that solve bounded parity games with weights (in this section); and
- bounded parity games with weights can be solved in polynomial time with oracles that solve energy parity games (Section 5).

Furthermore, in Section 8 we polynomially reduce solving energy parity games to solving parity games with weights and thereby show that parity games with weights, bounded parity games with weights, and energy parity games belong to the same complexity class.

The energy parity games that we reduce to are known to be efficiently solvable [4, 10]: they are in $\text{NP} \cap \text{co-NP}$ and can be solved in pseudo-quasi-polynomial time.

We first introduce the **bounded parity condition with weights**, which is a strengthening of the parity condition with weights. Hence, it is also induced by a coloring and a weighting:

$$\begin{aligned} \text{BndWeightParity}(\Omega, w) &= \text{WeightParity}(\Omega, w) \\ &\cap \{\rho \in V^\omega \mid \text{no request in } \rho \text{ is unanswered with infinite cost}\} . \end{aligned}$$

Note that this condition allows for a finite number of unanswered requests, as long as they are unanswered with finite cost.

We solve parity games with weights by repeatedly solving bounded parity games with weights. To this end, we apply the following two properties of the winning conditions: We have $\text{BndWeightParity}(\Omega, w) \subseteq \text{WeightParity}(\Omega, w)$ as well as that $\text{WeightParity}(\Omega, w)$ is 0-extendable. Hence, if Player 0 has a strategy from a vertex v such that every consistent play has a suffix in $\text{BndWeightParity}(\Omega, w)$, then the strategy is winning for her from v w.r.t. $\text{WeightParity}(\Omega, w)$. Thus, $\text{Attr}_0(\mathcal{W}_0(\mathcal{A}, \text{BndWeightParity}(\Omega, w))) \subseteq \mathcal{W}_0(\mathcal{A}, \text{WeightParity}(\Omega, w))$. The algorithm that solves parity games with weights repeatedly

Algorithm 1 A fixed-point algorithm computing $\mathcal{W}_0(\mathcal{A}, \text{WeightParity}(\Omega, w))$.

```

 $k = 0; W_0^k = \emptyset; \mathcal{A}_k = \mathcal{A}$ 
repeat
   $k = k + 1$ 
   $X_k = \mathcal{W}_0(\mathcal{A}_{k-1}, \text{BndWeightParity}(\Omega, w))$ 
   $W_0^k = W_0^{k-1} \cup \text{Attr}_0^{\mathcal{A}_{k-1}}(X_k)$ 
   $\mathcal{A}_k = \mathcal{A}_{k-1} \setminus \text{Attr}_0^{\mathcal{A}_{k-1}}(X_k)$ 
until  $X_k = \emptyset$ 
return  $W_0^k$ 

```

removes attractors of winning regions of the bounded parity game with weights until a fixed point is reached. We will later formalize this sketch to show that the removed parts are a subset of Player 0's winning region in the parity game with weights.

To show that the obtained fixed point covers the complete winning region of Player 0, we use the following lemma to show that the remaining vertices are a subset of Player 1's winning region in the parity game with weights. The proof is very similar to the corresponding one for finitary parity games and parity games with costs.

► **Lemma 2.** *Let $\mathcal{G} = (\mathcal{A}, \text{WeightParity}(\Omega, w))$ and let $\mathcal{G}' = (\mathcal{A}, \text{BndWeightParity}(\Omega, w))$. If $\mathcal{W}_0(\mathcal{G}') = \emptyset$, then $\mathcal{W}_0(\mathcal{G}) = \emptyset$.*

Lemma 2 implies that the algorithm for solving parity games with weights by repeatedly solving bounded parity games with weights (see Algorithm 1) is correct. Note that we use an oracle for solving bounded parity games with weights. We provide a suitable algorithm in Section 5.

The loop terminates after at most $|\mathcal{A}|$ iterations (assuming the algorithm solving bounded parity games with weights terminates), as during each iteration at least one vertex is removed from the arena. The correctness proof relies on Lemma 2 and is similar to the one for finitary parity games [7] and for parity games with costs [14].

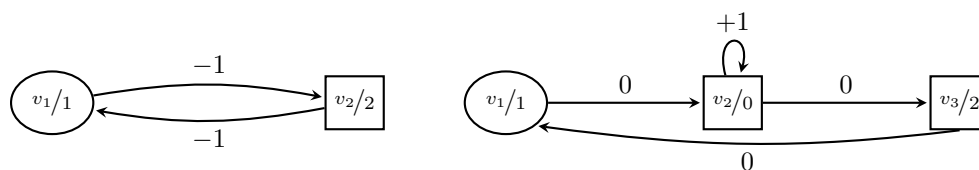
► **Lemma 3.** *Algorithm 1 returns $\mathcal{W}_0(\mathcal{A}, \text{WeightParity}(\Omega, w))$*

The winning strategy defined in the proof of Lemma 3 can be implemented by a memory structure of size $\max_{k \leq k^*} s_k$, where s_k is the size of a winning strategy σ_k for Player 0 in the bounded parity game with weights solved in the k -th iteration, and where k^* is the value of k at termination. To this end, one uses the fact that the winning regions X_k are disjoint and are never revisited once left. Hence, we can assume the implementations of the σ_k to use the same states.

5 Solving Bounded Parity Games with Weights

After having reduced the problem of solving parity games with weights to that of solving (multiple) bounded parity games with weights, we reduce solving bounded parity games with weights to solving (multiple) energy parity games [4].

Similarly to a parity game with weights, in an energy parity game, the vertices are colored and the edges are equipped with weights. It is the goal of Player 0 to satisfy the parity condition, while, at the same time, ensuring that the weight of every infix, its so-called energy level, is bounded from below. In contrast to a parity game with weights, however, the weights in an energy parity game are not tied to the requests and responses denoted by the coloring.



■ **Figure 2** The difference between energy parity games and parity games with weights.

Consider, for example, the games shown in Figure 2. In the game on the left-hand side, players only have a single, trivial strategy. If we interpret this game as a parity game with weights, Player 0 wins from every vertex, as each request is answered with cost one. If we, however, interpret that game as an energy parity game, Player 1 instead wins from every vertex, since the energy level decreases by one with every move. In the game on the right-hand side, the situation is mirrored: When interpreting this game as a parity game with weights, Player 1 wins from every vertex, as she can easily unbound the costs of the requests for color one by staying in vertex v_2 for an ever-increasing number of cycles. Dually, when interpreting this game as an energy parity game, Player 0 wins from every vertex, since the parity condition is clearly satisfied in every play, and Player 1 is only able to increase the energy level, while it is never decreased.

In Section 5.1, we introduce energy parity games formally and present how to solve bounded parity games with weights via energy games in Section 5.2.

5.1 Energy Parity Games

An energy parity game $\mathcal{G} = (\mathcal{A}, \Omega, w)$ consists of an arena $\mathcal{A} = (V, V_0, V_1, E)$, a coloring $\Omega: V \rightarrow \mathbb{N}$ of V , and an edge weighting $w: E \rightarrow \mathbb{Z}$ of E . Note that this definition is not compatible with the framework presented in Section 2, as we have not (yet) defined the winner of the plays. This is because they depend on an initial credit, which is existentially quantified in the definition of winning the game \mathcal{G} . Formally, the set of winning plays with initial credit $c_0 \in \mathbb{N}$ is defined as

$$\text{EnergyParity}_{c_0}(\Omega, w) = \text{Parity}(\Omega) \cap \{v_0 v_1 v_2 \cdots \in V^\omega \mid \forall j \in \mathbb{N}. c_0 + w(v_0 \cdots v_j) \geq 0\} .$$

Now, we say that Player 0 wins \mathcal{G} from v if there exists some initial credit $c_0 \in \mathbb{N}$ such that he wins $\mathcal{G}_{c_0} = (\mathcal{A}, \text{EnergyParity}_{c_0}(\Omega, w))$ from v (in the sense of the definitions in Section 2). If this is not the case, i.e., if Player 1 wins \mathcal{G}_{c_0} from v for every c_0 , then we say that Player 1 wins \mathcal{G} from v . Note that the initial credit is uniform for all plays, unlike the bound on the cost-of-response in the definition of the parity condition with weights, which may depend on the play.

Unravelling these definitions shows that Player 0 wins \mathcal{G} from v if there is an initial credit c_0 and a strategy σ , such that every play that starts in v and is consistent with σ satisfies the parity condition *and* the accumulated weight over the play prefixes (the energy level) never drops below $-c_0$. We call such a strategy σ a winning strategy for Player 0 in \mathcal{G} from v . Dually, Player 1 wins \mathcal{G} from v if, for every initial credit c_0 , there is a strategy τ_{c_0} , such that every play that starts in v and is consistent with τ_{c_0} violates the parity condition *or* its energy level drops below $-c_0$ at least once. Thus, the strategy τ_{c_0} may, as the notation suggests, depend on c_0 . However, Chatterjee and Doyen showed that using different strategies is not necessary: There is a uniform strategy τ that is winning from v for every initial credit c_0 .

► **Proposition 4** ([4]). *Let \mathcal{G} be an energy parity game. If Player 1 wins \mathcal{G} from v , then she has a single positional strategy that is winning from v in \mathcal{G}_{c_0} for every c_0 .*

We call such a strategy as in Proposition 4 a winning strategy for Player 1 from v . A play consistent with such a strategy either violates the parity condition, or the energy levels of its prefixes diverge towards $-\infty$.

Furthermore, Chatterjee and Doyen obtained an upper bound on the initial credit necessary for Player 0 to win an energy parity game, as well an upper bound on the size of a corresponding finite-state winning strategy.

► **Proposition 5** ([4]). *Let \mathcal{G} be an energy parity game with n vertices, d colors, and largest absolute weight W . The following are equivalent for a vertex v of \mathcal{G} :*

1. *Player 0 wins \mathcal{G} from v .*
2. *Player 0 wins $\mathcal{G}_{(n-1)W}$ from v with a finite-state strategy with at most ndW states.*

The previous proposition yields that finite-state strategies of bounded size suffice for Player 0 to win.

Such strategies do not admit long expensive descents, which we show by a straightforward pumping argument.

► **Lemma 6.** *Let \mathcal{G} be an energy parity game with n vertices and largest absolute weight W . Further, let σ be a finite-state strategy of size s , and let ρ be a play that starts in some vertex, from which σ is winning, and is consistent with σ . Every infix π of ρ satisfies $w(\pi) > -Wns$.*

Moreover, Chatterjee and Doyen gave an upper bound on the complexity of solving energy parity games, which was recently supplemented by Daviaud et al. with an algorithm solving them in pseudo-quasi-polynomial time.

► **Proposition 7** ([4, 10]). *The following problem is in $\text{NP} \cap \text{CO-NP}$ and can be solved in pseudo-quasi-polynomial time: “Given an energy parity game \mathcal{G} and a vertex v in \mathcal{G} , does Player 0 win \mathcal{G} from v ?”*

5.2 From Bounded Parity Games with Weights to Energy Parity Games

Let $\mathcal{G} = (\mathcal{A}, \text{BndWeightParity}(\Omega, w))$ be a bounded parity game with weights with vertex set V . Without loss of generality, we assume $\Omega(v) \geq 2$ for all $v \in V$. We construct, for each vertex v^* of \mathcal{A} , an energy parity game \mathcal{G}_{v^*} with the following property: Player 1 wins \mathcal{G}_{v^*} from some designated vertex induced by v^* if and only if she is able to unbound the amplitude for the request of the initial vertex of the play when starting from v^* . This construction is the technical core of the fixed-point algorithm that solves bounded parity games with weights via solving energy parity games.

The main obstacle towards this is that, in the bounded parity game with weights \mathcal{G} , Player 1 may win by unbounding the amplitude for a request from above or from below, while she can only win \mathcal{G}_{v^*} by unbounding the costs from below. We model this in \mathcal{G}_{v^*} by constructing two copies of \mathcal{A} . In one of these copies the edge weights are copied from \mathcal{G} , while they are inverted in the other copy. We allow Player 1 to switch between these copies arbitrarily. To compensate for Player 1’s power to switch, Player 0 can increase the energy level in the resulting energy parity game during each switch.

First, we define the set of polarities $P = \{+, -\}$ as well as $\overline{+} = -$ and $\overline{-} = +$. Given a vertex v^* of \mathcal{A} , define the “polarized” arena $\mathcal{A}_{v^*} = (V', V'_0, V'_1, E')$ of $\mathcal{A} = (V, V_0, V_1, E)$ with

- $V' = (V \times P) \cup (E \times P \times \{0, 1\})$,
- $V'_i = (V_i \times P) \cup (E \times P \times \{i\})$ for $i \in \{0, 1\}$, and
- E' contains the following edges for every edge $e = (v, v') \in E$ with $\Omega(v) \notin \text{Ans}(\Omega(v^*))$ and every polarity $p \in P$:
 - $((v, p), (e, p, 1))$: The player whose turn it is at v picks a successor v' . The edge $e = (v, v')$ is stored as well as the polarity p .
 - $((e, p, 1), (v', p))$: Then, Player 1 can either keep the polarity p unchanged and execute the move to v' , or
 - $((e, p, 1), (e, p, 0))$: she decides to change the polarity, and another auxiliary vertex is reached.
 - $((e, p, 0), (e, p, 0))$: If the polarity is to be changed, then Player 0 is able to use a self-loop to increase the energy level (see below), before
 - $((e, p, 0), (v', \bar{p}))$: he can eventually complete the polarity switch by moving to v' .
- Furthermore, for every vertex v with $\Omega(v) \in \text{Ans}(\Omega(v^*))$ and every polarity $p \in P$, E' contains the self-loop $((v, p), (v, p))$.⁴

Thus, a play in \mathcal{A}_{v^*} simulates a play in \mathcal{A} , unless Player 0 stops the simulation by using the self-loop at a vertex of the form $(e, p, 0)$ ad infinitum, and unless an answer to $\Omega(v^*)$ is reached. We define the coloring and the weighting for \mathcal{A}_{v^*} so that Player 0 loses in the former case and wins in the latter case. Furthermore, the coloring is defined so that all simulating plays that are not stopped have the same color sequence as the simulated play (save for irrelevant colors on the auxiliary vertices in $E \times P \times \{0, 1\}$). Hence, we define

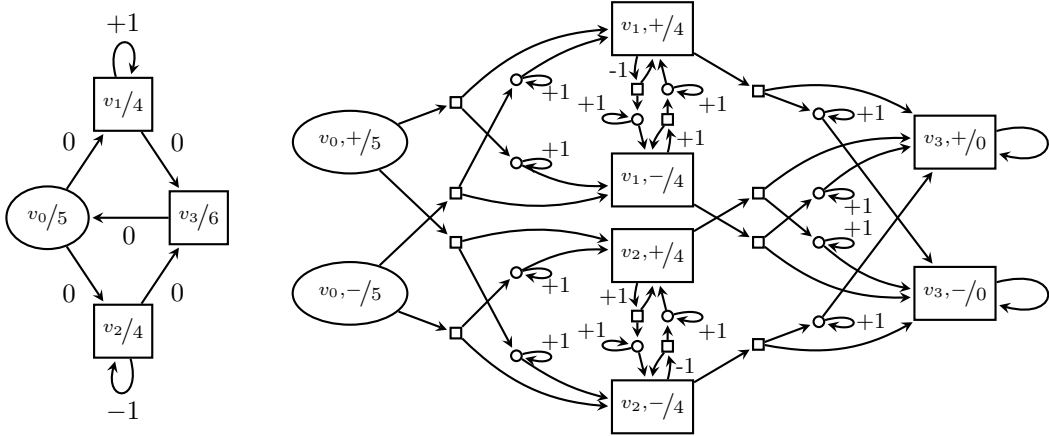
$$\Omega_{v^*}(v) = \begin{cases} \Omega(v') & \text{if } v = (v', p) \text{ with } v' \notin \text{Ans}(\Omega(v^*)) , \\ 0 & \text{if } v = (v', p) \text{ with } v' \in \text{Ans}(\Omega(v^*)) , \\ 1 & \text{otherwise} . \end{cases}$$

As desired, due to our assumption that $\Omega(v) \geq 2$ for all $v \in V$, the vertices from $E \times P \times \{0, 1\}$ do not influence the maximal color visited infinitely often during a play, unless Player 0 opts to remain in some $(e, p, 0)$ ad infinitum (and thereby violating the parity condition) or an answer to the color of v^* is reached (and thereby satisfying the parity condition).

Moreover, recall that our aim is to allow Player 1 to choose the polarity of edges by switching between the two copies of \mathcal{A} occurring in \mathcal{A}_{v^*} . Intuitively, Player 1 should opt for positive polarity in order to unbound the costs incurred by the request posed by v^* from above, while she should opt for negative polarity in order to unbound these costs from below. Since in an energy parity game, it is, broadly speaking, beneficial for Player 1 to move along edges of negative weight, we negate the weights of edges in the copy of \mathcal{A} with positive polarity. Thus, we define

$$w_{v^*}(e) = \begin{cases} -w(v, v') & \text{if } e = ((v, +), ((v, v'), +, 1)) , \\ w(v, v') & \text{if } e = ((v, -), ((v, v'), -, 1)) , \\ 1 & \text{if } e = ((e, p, 0), (e, p, 0)) , \\ 0 & \text{otherwise} . \end{cases}$$

⁴ Note that this definition introduces some terminal vertices, i.e., those of the form $((v, v'), p, i)$ with $\Omega(v) \in \text{Ans}(\Omega(v^*))$. However, these vertices also have no incoming edges. Hence, to simplify the definition, we just ignore them.



■ **Figure 3** A bounded parity game with weights \mathcal{G} and the associated energy parity game \mathcal{G}_{v_0} . The unnamed vertices of Player 1 (Player 0) are of the form $((v, v'), p, 1)$ (of the form $((v, v'), p, 0)$) when between the vertices (v, p) and (v', p') . All missing edge weights in \mathcal{G}_{v_0} are 0.

This definition implies that the self-loops at vertices of the form (v, p) with $\Omega(v) \in \text{Ans}(\Omega(v^*))$ have weight zero. Combined with the fact that these vertices have color zero, this allows Player 0 to win \mathcal{G}_{v^*} by reaching such a vertex. Intuitively, answering the request posed at v^* is beneficial for Player 0. In particular, if $\Omega(v^*)$ is even, then Player 0 wins \mathcal{G}_{v^*} trivially from (v^*, p) , as we then have $\Omega(v^*) \in \text{Ans}(\Omega(v^*))$.

Finally, define the energy parity game $\mathcal{G}_{v^*} = (\mathcal{A}_{v^*}, \Omega_{v^*}, w_{v^*})$. In the following, we are only interested in plays starting in vertex $(v^*, +)$ in \mathcal{G}_{v^*} .

► **Example 8.** Consider the bounded parity game with weights depicted on the left hand side of Figure 3 and the associated energy parity game \mathcal{G}_{v_0} on the right side. First, let us note that all other \mathcal{G}_v for $v \neq v_0$ are trivial in that they all consist of a single vertex (reachable from $(v, +)$), which has even color with a self-loop of weight zero. Hence, Player 0 wins each of these games from $(v, +)$.

Player 1 wins \mathcal{G} from v_0 , where a request for color 5 is opened, which is then kept unanswered with infinite cost by using the self-loop at v_1 or v_2 ad infinitum, depending on which successor Player 0 picks.

We show that Player 1 wins \mathcal{G}_{v_0} from $(v_0, +)$: the outgoing edges of $(v_0, +)$ correspond to picking the successor v_1 or v_2 as in \mathcal{G} . Before this is executed, however, Player 1 gets to pick the polarity of the successor: she should pick $+$ for v_1 and $-$ for v_2 . Now, Player 0 may use the self-loop at her “tiny” vertices ad infinitum. These vertices have color one, i.e., Player 1 wins the resulting play. Otherwise, we reach the vertex $(v_1, +)$ or $(v_2, -)$. From both vertices, Player 1 can enforce a loop of negative weight, which allows him to win by violating the energy condition.

Note that the winning strategy for Player 1 for \mathcal{G} from v is very similar to that for her for \mathcal{G}_{v_0} from $(v_0, +)$. We show that one direction holds in general: A winning strategy for Player 0 for \mathcal{G}_v from $(v, +)$ is “essentially” one for him in \mathcal{G} from v .

Note that the other direction does, in general, not hold. This can be seen by adding a vertex v_{-1} of color 3 with a single edge to v_0 . Then, vertices of the form (v_i, p) with $i \in \{1, 2\}$ in $\mathcal{G}_{v_{-1}}$ are winning sinks for Player 0. Hence, he wins $\mathcal{G}_{v_{-1}}$ from (v_{-1}, p) in spite of losing the bounded parity game with weights from v_{-1} .

Algorithm 2 A fixed-point algorithm computing $\mathcal{W}_1(\mathcal{A}, \text{BndWeightParity}(\Omega, w))$.

```

 $k = 0; W_1^k = \emptyset; \mathcal{A}_k = \mathcal{A}$ 
repeat
   $k = k + 1$ 
   $X_k = \{v^* \mid \text{Player 1 wins the energy parity game } ((\mathcal{A}_{k-1})_{v^*}, \Omega_{v^*}, w_{v^*}) \text{ from } (v^*, +)\}$ 
   $W_1^k = W_1^{k-1} \cup \text{Attr}_1^{\mathcal{A}_{k-1}}(X_k)$ 
   $\mathcal{A}_k = \mathcal{A}_{k-1} \setminus \text{Attr}_1^{\mathcal{A}_{k-1}}(X_k)$ 
until  $X_k = \emptyset$ 
return  $W_1^k$ 

```

Hence, the initial request the vertex v inducing \mathcal{G}_v plays a special role in the construction: It is the request Player 1 aims to keep unanswered with infinite cost. To overcome this and to complete our construction, we show a statement reminiscent of Lemma 2: If Player 0 wins \mathcal{G}_v from $(v, +)$ for every v , then she also wins \mathcal{G}_x from every vertex. With this relation at hand, one can again construct a fixed-point algorithm solving bounded parity games with weights using an oracle for solving energy parity games that is very similar to Algorithm 1.

Formally, we have the following lemma, which forms the technical core of our algorithm that solves bounded parity games with weights by solving energy parity games.

► **Lemma 9.** *Let \mathcal{G} be a bounded parity game with weights with vertex set V .*

1. *Let $v^* \in V$. If Player 1 wins \mathcal{G}_{v^*} from $(v^*, +)$, then $v^* \in \mathcal{W}_1(\mathcal{G})$.*
2. *If Player 0 wins \mathcal{G}_{v^*} from $(v^*, +)$ for all $v^* \in V$, then $\mathcal{W}_1(\mathcal{G}) = \emptyset$.*

This lemma is the main building block for the algorithm that solves bounded parity games with weights by repeatedly solving energy parity games, which is very similar to Algorithm 1. Indeed, we just swap the roles of the players: We compute 1-attractors instead of 0-attractors and we change the definition of X_k . Hence, we obtain the following algorithm (Algorithm 2).

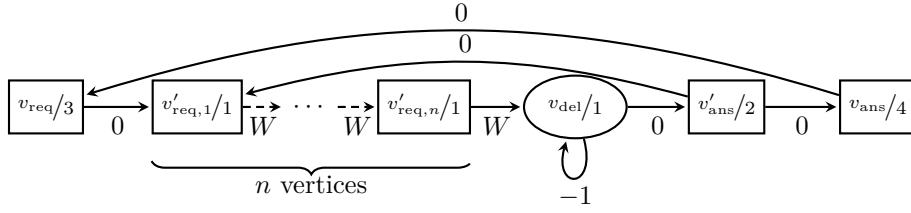
Algorithm 2 terminates after solving at most a quadratic number of energy parity games. Furthermore, the proof of correctness is analogous to the one for Algorithm 1, relying on Lemma 9. We only need two further properties: the 1-extendability of $\text{BndWeightParity}(\Omega, w)$, and an assertion that $\text{Attr}_1^{\mathcal{A}_{k-1}}(X_k)$ is a trap for Player 0 in \mathcal{A}_{k-1} . Both are easy to verify.

After plugging Algorithm 2 into Algorithm 1, Proposition 7 yields our main theorem, settling the complexity of solving parity games with weights.

► **Theorem 10.** *The following problem is in $\text{NP} \cap \text{co-NP}$ and can be solved in pseudo-quasi-polynomial time: “Given a parity game with weights \mathcal{G} and a vertex v in \mathcal{G} , does Player 0 win \mathcal{G} from v ?”*

6 Memory Requirements

We now discuss the upper and lower bounds on the memory required to implement winning strategies for either player. Recall that we use binary encoding to denote weights, i.e., weights may be exponential in the size of the game. In this section we show polynomial (in n , d , and W) upper and lower bounds on the necessary and sufficient memory for Player 0 to win parity games with weights. Due to the binary encoding of weights, these bounds are exponential in the size of the game. In contrast, Player 1 requires infinite memory.



■ **Figure 4** A game of size $\mathcal{O}(n)$ in which Player 0 only wins with strategies of size at least $nW + 1$.

► **Theorem 11.** *Let \mathcal{G} be a parity game with weights with n vertices, d colors, and largest absolute weight W assigned to any edge in \mathcal{G} . Moreover, let v be a vertex of \mathcal{G} .*

1. *Player 0 has a winning strategy σ from $\mathcal{W}_0(\mathcal{G})$ with $|\sigma| \in \mathcal{O}(nd^2W)$. This bound is tight.*
2. *There exists a parity game with weights \mathcal{G} , such that Player 1 has a winning strategy from each vertex v in \mathcal{G} , but she has no finite-state winning strategy from any v in \mathcal{G} .*

The proof of the second item of Theorem 11 is straightforward, since Player 1 already requires infinite memory to implement winning strategies in finitary parity games [7]. Since parity games with weights subsume finitary parity games, this result carries over to our setting. We show the game witnessing this lower bound on the right-hand side of Figure 2.

In contrast, exponential memory is sufficient, but also necessary, for Player 0. To this end, we first prove that the winning strategy for him constructed in the proof of Lemma 9.2 suffers at most a linear blowup in comparison to his winning strategies in the underlying energy parity games. This is sufficient as we have argued in Section 4 that the construction of a winning strategy for Player 0 in a parity game with weights suffers no blowup in comparison to the underlying bounded parity games with weights.

► **Lemma 12.** *Let \mathcal{G} , n , d , and W be as in Theorem 11. Player 0 has a finite-state winning strategy of size at most $d(6n)(d + 2)(W + 1)$ from $\mathcal{W}_0(\mathcal{G})$.*

Having established an upper bound on the memory required by Player 0, we now proceed to show that this exponential bound is indeed tight, which is witnessed by the games \mathcal{G}_n depicted in Figure 4.

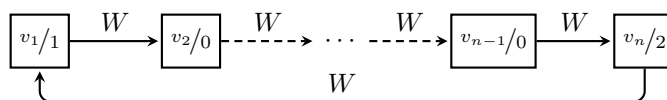
► **Lemma 13.** *Let $n, W \in \mathbb{N}$. There exists a parity game with weights $\mathcal{G}_{n,W}$ with n vertices and largest absolute weight W such that Player 0 wins \mathcal{G}_n from every vertex, but each winning strategy for her is of size at least $nW + 1$.*

7 Quality of Strategies

We have shown in the previous section that finite-state strategies of bounded size suffice for Player 0 to win in parity games with weights, while Player 1 clearly requires infinite memory. However, as we are dealing with a quantitative winning condition, we are not only interested in the size of winning strategies, but also in their quality. More precisely, we are interested in an upper bound on the cost of requests that Player 0 can ensure. In this section, we show that he can guarantee an exponential upper bound on such costs. Dually, Player 1 is required to unbound the cost of responses.

► **Theorem 14.** *Let \mathcal{G} be a parity game with weights with n vertices, d colors, and largest absolute weight W .*

There exists a $b \in \mathcal{O}((ndW)^2)$ and a strategy σ for Player 0 such that, for all plays ρ beginning in $\mathcal{W}_0(\mathcal{G})$ and consistent with σ , we have $\limsup_{j \rightarrow \infty} \text{Cor}(\rho, j) \leq b$. This bound is tight.



■ **Figure 5** The game $\mathcal{G}_{n,W}$ witnessing an exponential lower bound on the cost that Player 0 can ensure.

We first show that Player 0 can indeed ensure an upper bound as stated in Theorem 14. We obtain this bound via a straightforward pumping argument leveraging the upper bound on the size of winning strategies obtained in Lemma 12.

► **Lemma 15.** *Let \mathcal{G} , n , d , and W be as in the statement of Theorem 14 and let $s = d(6n)(d+2)(W+1)$. Player 0 has a winning strategy σ such that, for each play ρ that starts in $\mathcal{W}_0(\mathcal{G})$ and is consistent with σ , we have $\limsup_{j \rightarrow \infty} \text{Cor}(\rho, j) \leq nsW$.*

Having thus shown that Player 0 can indeed ensure an exponential upper bound on the incurred cost, we now proceed to show that this bound is tight. A simple example shows that there exists a series of parity games with weights, in which Player 0 wins from every vertex, but in which he cannot enforce a sub-exponential cost of any request.

► **Lemma 16.** *Let $n, W \in \mathbb{N}$. There exists a parity game with weights $\mathcal{G}_{n,W}$ with n vertices and largest absolute weight W as well as a vertex $v \in \mathcal{W}_0(\mathcal{G})$, such that for each winning strategy for Player 0 from v there exists a play ρ starting in v and consistent with σ with $\limsup_{j \rightarrow \infty} \text{Cor}(\rho, j) \geq (n-1)W$.*

Proof. We show the game $\mathcal{G}_{n,W}$ in Figure 5. The arena of $\mathcal{G}_{n,W}$ is a cycle with n vertices of Player 1, where each edge has weight W . Moreover, one vertex is labeled with color two, its directly succeeding vertex is labeled with color one. All remaining vertices have color zero.

Player 0 only has a single strategy in this game and there exist only n plays in $\mathcal{G}_{n,W}$, each starting in a different vertex of \mathcal{G}_n . In each play, each request for color one is only answered after $n-1$ steps, each contributing a cost of W . Hence, this request incurs a cost of $(n-1)W$. Moreover, as this request is posed and answered infinitely often in each play, we obtain the desired result. ◀

8 From Energy Parity Games to (Bounded) Parity Games with Weights

We have discussed in Sections 4 and 5 how to solve parity games with weights via solving bounded parity games with weights and how to solve the latter games by solving energy parity games, both steps with a polynomial overhead. An obvious question is whether one can also solve energy parity games by solving (bounded) parity games with weights. In this section, we answer this question affirmatively. We show how to transform an energy parity game into a bounded parity game with weights so that solving the latter also solves the former. Then, we show how to transform a bounded parity game with weights into a parity game with weights with the same relation: Solving the latter also solves the former. Both constructions here are gadget based and increase the size of the arenas only linearly. Hence, all three types of games are interreducible with at most polynomial overhead.

8.1 From Energy Parity Games to Bounded Parity Games with Weights

Note that, in an energy parity game, Player 0 wins if the energy increases without a bound, as long as there is a lower bound. However, in a bounded parity game, he has to ensure an upper and a lower bound. Thus, we show in a first step how to modify an energy parity

game so that Player 0 still has to ensure a lower bound on the energy, but can also *throw away* unnecessary energy during each transition, thereby also ensuring an upper bound. The most interesting part of this construction is to determine when energy becomes unnecessary to ensure a lower bound. Here, we rely on Lemma 6.

Formally, let $\mathcal{G} = (\mathcal{A}, \Omega, w)$ be an energy parity game with $\mathcal{A} = (V, V_0, V_1, E)$ where we assume w.l.o.g. that the minimal color in $\Omega(V)$ is strictly greater than 1. Now, we define $\mathcal{G}' = (\mathcal{A}', \Omega', w')$ with $\mathcal{A}' = (V', V'_0, V'_1, E')$ where

- $V' = V \cup E$, $V'_0 = V_0 \cup E$, and $V'_1 = V_1$,
- $E' = \{(v, e), (e, e), (e, v') \mid e = (v, v') \in E\}$,
- $\Omega'(v) = \Omega(v)$ and $\Omega'(e) = 1$, and
- $w'(v, e) = w(e)$, $w'(e, e) = -1$, and $w'(e, v') = 0$ for every $e = (v, v') \in E$.

Intuitively, every edge of \mathcal{A} is subdivided and a new vertex for Player 0 is added, where he can decrease the energy level. The negative weight ensures that he eventually leaves this vertex in order to satisfy an energy condition.

We say that a strategy σ for Player 0 in \mathcal{A}' is corridor-winning for him from some $v \in V$, if there is a $b \in \mathbb{N}$ such that every play ρ that starts in v and is consistent with σ satisfies the parity condition and $\text{Ampl}(\rho) \leq b$. Hence, instead of just requiring a lower bound on the energy level as in the energy parity condition, we also require a uniform upper bound on the energy level (where we w.l.o.g. assume these bounds to coincide).

► **Lemma 17.** *Let \mathcal{G} and \mathcal{G}' be as above and let $v \in V$. Player 0 has a winning strategy for \mathcal{G} from v if and only if Player 0 has a corridor-winning strategy for \mathcal{G}' from v .*

Now, we turn \mathcal{G}' into a bounded parity game with weights. In such a game, the cost-of-response of every request has to be bounded, but the overall energy level of the play may still diverge to $-\infty$. To rule this out, we open one unanswerable request at the beginning of each play, which has to be unanswered with finite cost in order to satisfy the bounded parity condition with weights. If this is the case, then the energy level of the play is always in a bounded corridor, i.e., we obtain a corridor-winning strategy.

Formally, for every vertex $v \in V$, we add a vertex \bar{v} to \mathcal{A}' of an odd color c^* that is larger than every color in $\Omega(V)$, i.e., the request can never be answered. Furthermore, \bar{v} has a single outgoing edge to v of weight 0, i.e., it is irrelevant whose turn it is. Call the resulting arena \mathcal{A}'' , the resulting coloring Ω'' , and the resulting weighting w'' , and let $\mathcal{G}'' = (\mathcal{A}'', \text{BndWeightParity}(\Omega'', w''))$.

► **Lemma 18.** *Let \mathcal{G}' and \mathcal{G}'' be as above and let $v \in V$. Player 0 has a corridor-winning strategy for \mathcal{G}' from v if and only if $\bar{v} \in \mathcal{W}_0(\mathcal{G}'')$.*

8.2 From Bounded Parity Games with Weights to Parity Games with Weights

Next, we show how to turn a bounded parity game with weights into a parity game with weights so that solving the latter also solves the former. The construction here uses the same restarting mechanism that underlies the proof of Lemma 2: as soon as a request has incurred a cost of b , restart the play and enforce a request of cost $b + 1$, and so on. Unlike the proof of Lemma 2, where Player 1 could restart the play at any vertex, here we always have to return to a fixed initial vertex we are interested in. While resetting, we have to answer all requests in order to prevent Player 1 to use the reset to prevent requests from being answered. Assume $v^* \in V$ is the initial vertex we are interested in. Then, we subdivide

■ **Table 1** Characteristic properties of variants of parity games.

	Complexity	Mem. Pl. 0/Pl. 1	Bounds
Parity Games [3]	quasi-poly.	pos./pos.	–
Energy Parity Games [4, 10]	pseudo-quasi-poly.	$\mathcal{O}(ndW)$ /pos.	$\mathcal{O}(nW)$
Finitary Parity Games [7]	poly.	pos./inf.	$\mathcal{O}(nW)$
Parity Games with Costs [14, 22]	quasi-poly.	pos./inf.	$\mathcal{O}(nW)$
Parity Games with Weights	pseudo-quasi-poly.	$\mathcal{O}(nd^2W)$ /inf.	$\mathcal{O}((ndW)^2)$

every edge in \mathcal{A}'' to allow Player 1 to restart the play by answering all open requests and then moving back to v^* .

Formally, fix a bounded parity game with weights $\mathcal{G} = (\mathcal{A}, \text{BndWeightParity}(\Omega, w))$ with $\mathcal{A} = (V, V_0, V_1, E)$ and a vertex $v^* \in V$. We define the parity game with weights $\mathcal{G}_{v^*} = (\mathcal{A}_{v^*}, \text{WeightParity}(\Omega_{v^*}, w_{v^*}))$ with $\mathcal{A}_{v^*} = (V', V'_0, V'_1, E')$ where

- $V' = V \cup E \cup \{\top\}$, $V'_0 = V_0$, and $V'_1 = V_1 \cup E \cup \{\top\}$,
- $E' = \{(v, e), (e, \top), (e, v') \mid e = (v, v') \in E\} \cup \{(\top, v^*)\}$,
- $\Omega_{v^*}(v) = \Omega(v)$, $\Omega_{v^*}(e) = 0$ for every $e \in E$, and $\Omega_{v^*}(\top) = 2 \max(\Omega(V))$, and
- $w_{v^*}(v, e) = w(e)$ for $(v, e) \in V \times E$ and $w_{v^*}(e') = 0$ for every other edge $e' \in E'$.

► **Lemma 19.** *Let \mathcal{G} and \mathcal{G}_{v^*} as above. Then, $v^* \in \mathcal{W}_0(\mathcal{G})$ if and only if $v^* \in \mathcal{W}_0(\mathcal{G}_{v^*})$.*

9 Conclusions and Future Work

We have established that parity games with weights and bounded parity games fall into the same complexity class as energy parity games. This is interesting, because, while solving such games has the signature complexity class $\text{NP} \cap \text{co-NP}$, they are not yet considered a class in their own right. It is also interesting because the properties appear to be inherently different: While they both combine the qualitative parity condition with quantified costs, parity games with weights *combine* these aspects on the property level, whereas energy parity games simply look at the combined – and totally unrelated – properties. We show the characteristic properties of parity games and of games with combinations of a parity condition with quantitative conditions relevant for this work in Table 1.

As future work, we are looking into the natural extensions of parity games with weights to Streett games with weights [7, 14], and at the complexity of determining optimal bounds and strategies that obtain them [30]. We are also looking at variations of the problem. The two natural variations are

- to use a one-sided definition (instead of the absolute value) for the amplitude of a play, i.e., using $\text{Ampl}(\pi) = \sup_{j < |\pi|} w(v_0 \cdots v_j) \in \mathbb{N}_\infty$ (instead of $\text{Ampl}(\pi) = \sup_{j < |\pi|} |w(v_0 \cdots v_j)| \in \mathbb{N}_\infty$), and
- to use an arbitrary consecutive subsequence of a play, i.e., $\text{Ampl}(\pi) = \sup_{j \leq k < |\pi|} |w(v_j \cdots v_k)| \in \mathbb{N}_\infty$.

There are good arguments in favor and against using these individual variations – and their combination to $\text{Ampl}(\pi) = \sup_{j \leq k < |\pi|} w(v_j \cdots v_k) \in \mathbb{N}_\infty$ – but we feel that the introduction of parity games with weights benefit from choosing one of the four combinations as *the* parity games with weights.

We expect the complexity to rise when changing from maximizing over the absolute value to maximizing over the value, as this appears to be close to pushdown boundedness games [5], and we conjecture this problem to be PSPACE complete.

References

- 1 Henrik Björklund and Sergei Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Appl. Math.*, 155(2):210–229, 2007. doi:10.1016/j.dam.2006.04.029.
- 2 Anca Browne, Edmund M. Clarke, Somesh Jha, David E. Long, and Wilfredo R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theo. Comp. Sci.*, 178(1–2):237–255, 1997. doi:10.1016/S0304-3975(96)00228-9.
- 3 C. S. Calude, S. Jain, B. Khossainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *STOC 2017*, pages 252–263. ACM Press, 2017. doi:10.1145/3055399.3055409.
- 4 Krishnendu Chatterjee and Laurent Doyen. Energy Parity Games. *Theo. Comp. Sci.*, 458:49–60, 2012. doi:10.1016/j.tcs.2012.07.038.
- 5 Krishnendu Chatterjee and Nathanaël Fijalkow. Infinite-state games with finitary conditions. In *CSL 2013*, volume 23 of *LIPICs*, pages 181–196. Schloss Dagstuhl–LZI, 2013. doi:10.4230/LIPICs.CSL.2013.181.
- 6 Krishnendu Chatterjee and Thomas A. Henzinger. Finitary winning in omega-regular games. In Holger Hermanns and Jens Palsberg, editors, *TACAS 2006*, volume 3920 of *LNCS*, pages 257–271, 2006. doi:10.1007/11691372_17.
- 7 Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. Finitary winning in omega-regular games. *Trans. Comput. Log.*, 11(1):1:1–1:27, 2009. doi:10.1145/1614431.1614432.
- 8 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdzinski. Mean-payoff parity games. In *LICS 2005*, pages 178–187. IEEE Computer Society, 2005. doi:10.1109/LICS.2005.26.
- 9 Anne Condon. On algorithms for simple stochastic games. In *Advances in Computational Complexity Theory*, pages 51–73. American Mathematical Society, 1993.
- 10 Laure Daviaud, Marcin Jurdzinski, and Ranko Lazic. A pseudo-quasi-polynomial algorithm for solving mean-payoff parity games. In *LICS 2018*, page (to appear), 2018.
- 11 E. Allen Emerson and Charanjit S. Jutla. Tree automata, μ -calculus and determinacy. In *FOCS 1991*, pages 368–377. IEEE Computer Society, 1991.
- 12 E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional μ -calculus. In *LICS 1986*, pages 267–278. IEEE Computer Society, 1986. doi:10.1109/SFCS.1991.185392.
- 13 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 *SPIN 2017*, pages 112–121. ACM, 2017. doi:10.1145/3092282.3092286.
- 14 Nathanaël Fijalkow and Martin Zimmermann. Parity and Streett games with costs. *LMCS*, 10(2), 2014. doi:10.2168/LMCS-10(2:14)2014.
- 15 Marcin Jurdziński. Deciding the winner in parity games is in $UP \cap co-UP$. *Information Processing Letters*, 68(3):119–124, November 1998. doi:10.1016/S0020-0190(98)00150-1.
- 16 Marcin Jurdziński. Small progress measures for solving parity games. In *STACS 2000*, volume 1770 of *LNCS*, pages 290–301, 2000. doi:10.1007/3-540-46541-3_24.
- 17 Marcin Jurdziński and Ranko Lazić. Succinct progress measures for solving parity games. In *LICS 2017*, pages 1–9. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005092.
- 18 Marcin Jurdziński, Mike Paterson, and Uri Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. on Comp.*, 38(4):1519–1532, 2008. doi:10.1137/070686652.
- 19 Dexter Kozen. Results on the propositional μ -calculus. *Theo. Comp. Sci.*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

- 20 Karoliina Lehtinen. A modal μ perspective on solving parity games in quasipolynomial time. In *LICS 2018*, page (to appear), 2018.
- 21 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.
- 22 Fabio Mogavero, Aniello Murano, and Loredana Sorrentino. On promptness in parity games. *Fundam. Inform.*, 139(3):277–305, 2015. doi:10.3233/FI-2015-1235.
- 23 Anil Nerode, Jeffrey B. Remmel, and Alexander Yakhnis. Mcnaughton games and extracting strategies for concurrent programs. *Ann. Pure Appl. Logic*, 78(1-3):203–242, 1996. doi:10.1016/0168-0072(95)00032-1.
- 24 Anuj Puri. *Theory of hybrid systems and discrete event systems*. PhD thesis, Computer Science Department, University of California, Berkeley, 1995.
- 25 Sven Schewe. An optimal strategy improvement algorithm for solving parity and pay-off games. In *CSL 2008*, volume 5213 of *LNCS*, pages 368–383, 2008. doi:10.1007/978-3-540-87531-4_27.
- 26 Sven Schewe. Solving parity games in big steps. *J. of Comp. and Sys. Sci.*, 84:243–262, 2017. doi:10.1016/j.jcss.2016.10.002.
- 27 Sven Schewe, Ashutosh Trivedi, and Thomas Varghese. Symmetric strategy improvement. In *ICALP 2015*, volume 9135 of *LNCS*, pages 388–400, 2015. doi:10.1007/978-3-662-47666-6_31.
- 28 Sven Schewe, Alexander Weinert, and Martin Zimmermann. Parity games with weights. *CoRR*, abs/1804.06168, 2018. arXiv:1804.06168.
- 29 Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *CAV 2000*, pages 202–215. Springer, 2000. doi:10.1007/10722167_18.
- 30 Alexander Weinert and Martin Zimmermann. Easy to win, hard to master: Optimal strategies in parity games with costs. *LMCS*, 13(3), 2017. doi:10.23638/LMCS-13(3:29)2017.
- 31 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theo. Comp. Sci.*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.
- 32 Uri Zwick and Mike S. Paterson. The complexity of mean payoff games on graphs. *Theo. Comp. Sci.*, 158(1-2):343–359, 1996. doi:10.1016/0304-3975(95)00188-3.

MacNeille Completion and Buchholz' Omega Rule for Parameter-Free Second Order Logics

Kazushige Terui

RIMS, Kyoto University, Japan
terui@kurims.kyoto-u.ac.jp

Abstract

Buchholz' Ω -rule is a way to give a syntactic, possibly ordinal-free proof of cut elimination for various subsystems of second order arithmetic. Our goal is to understand it from an algebraic point of view. Among many proofs of cut elimination for higher order logics, Maehara and Okada's algebraic proofs are of particular interest, since the essence of their arguments can be algebraically described as the (Dedekind-) *MacNeille completion* together with Girard's reducibility candidates. Interestingly, it turns out that the Ω -rule, formulated as a rule of logical inference, finds its algebraic foundation in the MacNeille completion.

In this paper, we consider a family of sequent calculi $\mathbf{LIP} = \bigcup_{n \geq -1} \mathbf{LIP}_n$ for the *parameter-free* fragments of second order intuitionistic logic, that corresponds to the family $\mathbf{ID}_{<\omega} = \bigcup_{n < \omega} \mathbf{ID}_n$ of arithmetical theories of inductive definitions up to ω . In this setting, we observe a formal connection between the Ω -rule and the MacNeille completion, that leads to a way of interpreting second order quantifiers in a first order way in Heyting-valued semantics, called the Ω -*interpretation*. Based on this, we give a (partly) algebraic proof of cut elimination for \mathbf{LIP}_n , in which quantification over reducibility candidates, that are genuinely second order, is replaced by the Ω -interpretation, that is essentially first order. As a consequence, our proof is locally formalizable in \mathbf{ID} -theories.

2012 ACM Subject Classification Theory of computation \rightarrow Proof theory

Keywords and phrases Algebraic cut elimination, Parameter-free second order logic, MacNeille completion, Omega-rule

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.37

Related Version Full version available at <https://arxiv.org/abs/1804.11066>.

Funding This work was supported by KAKENHI 25330013.

1 Introduction

This paper is concerned with cut elimination for subsystems of second order logics. It is of course very well known that the full second order classical/intuitionistic logics admit cut elimination. Then why are we interested in their subsystems? A primary reason is that proving cut elimination for a subsystem is often very hard if one is sensitive to the metatheory within which (s)he works. This is witnessed by the vast literature in the traditional proof theory. In fact, proof theorists are not just interested in proving cut elimination itself, but in identifying a *characteristic principle* P (e.g. ordinals, ordinal diagrams, combinatorial principles and inductive definitions) for each system of logic, arithmetic and set theory, by proving cut elimination within a weak metatheory (e.g. \mathbf{PRA} , \mathbf{IS}_1 and \mathbf{RCA}_0) extended by P . Our motivation is to understand those hard proofs and results from an algebraic perspective.



© Kazushige Terui;
licensed under Creative Commons License CC-BY
27th EACSL Annual Conference on Computer Science Logic (CSL 2018).

Editors: Dan Ghica and Achim Jung; Article No. 37; pp. 37:1–37:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One can distinguish several types of cut elimination proofs for higher order logics/arithmetic: (i) syntactic proofs by ordinal assignment (e.g. Gentzen's consistency proof for **PA**), (ii) syntactic but ordinal-free proofs, (iii) semantic proofs based on Schütte's semivaluation or its variants (e.g. [30]), (iv) algebraic proofs based on completions (the list is not intended to be exhaustive). Historically (i) and (iii) precede (ii) and (iv), but understanding (i) takes years just to catch up with the expanding universe of ordinal notations, while (iii) is slightly unsatisfactory for the truly constructive logician since it involves *reductio ad absurdum* and weak König's lemma. Hence we address (ii) and (iv) in this paper.

For (ii), a very useful and versatile technique is Buchholz' Ω -rule. Introduced in the context of ordinal analysis of **ID**-theories [11] and further developed in, e.g., [14], it later yielded an ordinal-free proof of cut elimination for fragments/extensions of Π_1^1 -**CA**₀ [12, 4, 3]. However, the Ω -rule is notoriously complicated, and is hard to grasp its meaning at a glance. Even its semantic soundness is not clear at all. While Buchholz gives an account based on the BHK interpretation [11], we will try to give an algebraic account in this paper.

For (iv), there is a very conspicuous algebraic proof of cut elimination for higher order logics which may be primarily ascribed to Maehara [24] and Okada [26, 28]. In contrast to (iii), these algebraic proofs are fully constructive; no use of *reductio ad absurdum* or any nondeterministic principle. More importantly, it extends to proofs of normalization for proof nets and typed lambda calculi [27]. While their arguments can be described in various dialects (e.g. phase semantics in linear logic), apparently most neutral and most widely accepted would be to speak in terms of algebraic completions: the essence of their arguments can be described as the (*Dedekind-*)*MacNeille completion* together with Girard's reducibility candidates, as we will explain in Section 6.

Having a syntactic technique on one hand and an algebraic methodology on the other, it is natural to ask the relationship between them. To make things concrete, we consider, in addition to the standard sequent calculus **LI2** for second order intuitionistic logic, a family of subcalculi $\mathbf{LIP} = \bigcup_{n \geq -1} \mathbf{LIP}_n$ for the *parameter-free* fragments of **LI2**. **LIP** is the intuitionistic counterpart of the classical sequent calculus studied in [32]. Although we primarily work on intuitionistic logic, all results in this paper (except Proposition 11) carry over to classical logic too.

As we will see, cut elimination based on the Ω -rule technique works for **LIP**. Moreover, it turns out to be intimately related to the MacNeille completion in that the Ω -rule in our setting is not sound in Heyting-valued semantics in general, but is sound when the underlying algebra is the MacNeille completion of the Lindenbaum algebra. This observation leads to a curious way of interpreting second order formulas in a first order way, that we call the *Ω -interpretation*. The basic idea already appears in Altenkirch and Coquand [6], but ours is better founded and accommodates the existential quantifier too.

The Ω -rule and Ω -interpretation are two sides of the same coin. Combining them together, we obtain a (partly) algebraic proof of cut elimination for \mathbf{LIP}_n ($n \geq 0$), that is comparable with Aehlig's result [1] for the parameter-free, negative fragments of second order Heyting arithmetic. As with [1], our proof does not rely on (second order quantification over) reducibility candidates, and is formalizable in theories of finitely iterated inductive definitions.

The rest of this paper is organized as follows. In Section 2 we recall some basics of the MacNeille completion. In Section 3 we give some background on iterated inductive definitions and then introduce a family of sequent calculi $\mathbf{LIP} = \bigcup \mathbf{LIP}_n$. In Section 4 we transform the arithmetical Ω -rule into a logical one and explain how it works for **LIP**. In Section 5, we turn to the algebraic side of the Ω -rule, establish a connection with the MacNeille completion, and

motivate the Ω -interpretation. In Section 6, we review an algebraic proof of cut elimination for **LI2**, and then gives an algebraic proof for **LIP_n** based on the Ω -interpretation. Appendix A fully describes the sequent calculi studied in this paper. Omitted proofs are found in the full version of this paper available at <https://arxiv.org/abs/1804.11066>.

2 MacNeille completion

Let $\mathbf{A} = \langle A, \wedge, \vee \rangle$ be a lattice. A *completion* of \mathbf{A} is an embedding $e : \mathbf{A} \rightarrow \mathbf{B}$ into a complete lattice $\mathbf{B} = \langle B, \wedge, \vee \rangle$. We often assume that e is an inclusion map so that $\mathbf{A} \subseteq \mathbf{B}$.

For example, let $[0, 1]_{\mathbb{Q}} := [0, 1] \cap \mathbb{Q}$ be the chain of rational numbers in the unit interval (seen as a lattice). Then it admits an obvious completion $[0, 1]_{\mathbb{Q}} \subseteq [0, 1]$. For another example, let \mathbf{A} be a Boolean algebra. Then it also admits a completion $e : \mathbf{A} \rightarrow \mathbf{A}^{\sigma}$, where $\mathbf{A}^{\sigma} := \langle \wp(\text{uf}(\mathbf{A})), \cap, \cup, -, A, \emptyset \rangle$, the powerset algebra on the set of ultrafilters of \mathbf{A} , and $e(a) := \{u \in \text{uf}(\mathbf{A}) : a \in u\}$.

A completion $\mathbf{A} \subseteq \mathbf{B}$ is \vee -dense if $x = \vee\{a \in A : a \leq x\}$ holds for every $x \in B$. It is \wedge -dense if $x = \wedge\{a \in A : x \leq a\}$. A \vee -dense and \wedge -dense completion is called a *MacNeille completion*.

► **Theorem 1.** *Every lattice \mathbf{A} has a MacNeille completion unique up to isomorphism [8, 29]. A MacNeille completion is regular, i.e., preserves all joins and meets that already exist in \mathbf{A} .*

Coming back to the previous examples:

- $[0, 1]_{\mathbb{Q}} \subseteq [0, 1]$ is MacNeille, since $x = \inf\{a \in \mathbb{Q} : x \leq a\} = \sup\{a \in \mathbb{Q} : a \leq x\}$ for any $x \in [0, 1]$. It is regular since if $q = \lim_{n \rightarrow \infty} q_n$ holds in \mathbb{Q} , then it holds in \mathbb{R} too.
- $e : \mathbf{A} \rightarrow \mathbf{A}^{\sigma}$ is not regular when \mathbf{A} is an infinite Boolean algebra. In fact, the Stone space $\text{uf}(\mathbf{A})$ is compact, so collapses any infinite union of open sets into a finite one. It is actually a *canonical extension*, that has been extensively studied in ordered algebra and modal logic [23, 21, 20].

MacNeille completions behave better than canonical extensions in preservation of existing limits, but the price to pay is loss of generality. Let $\mathcal{DL}(\mathcal{HA}, \mathcal{BA}, \text{resp.})$ be the variety of distributive lattices (Heyting algebras, Boolean algebras, resp.).

► **Theorem 2.** ■ *\mathcal{DL} is not closed under MacNeille completions [18].*

- \mathcal{HA} and \mathcal{BA} are closed under MacNeille completions.
- \mathcal{HA} and \mathcal{BA} are the only nontrivial subvarieties of \mathcal{HA} closed under MacNeille completions [9].

As is well known, completion is a standard algebraic way to prove conservativity of extending first order logics to higher order ones. The above result indicates that MacNeille completions work for classical and intuitionistic logics, but not for proper intermediate logics. See [33] for more on MacNeille completions.

Now an easy but crucial observation follows.

► **Proposition 3.** *A completion $\mathbf{A} \subseteq \mathbf{B}$ is MacNeille iff the rules below are valid:*

$$\frac{\{a \leq y\}_{a \leq x}}{x \leq y} \quad \frac{\{x \leq a\}_{y \leq a}}{x \leq y}$$

where x, y range over B and a over A .

The left rule has infinitely many premises indexed by the set $\{a \in A : a \leq x\}$. It states that if $a \leq x$ implies $a \leq y$ for every $a \in A$, then $x \leq y$. This is valid just in case $x = \bigvee\{a \in A : a \leq x\}$. Likewise, the right rule states that if $y \leq a$ implies $x \leq a$ for every $a \in A$, then $x \leq y$. This is valid just in case $y = \bigwedge\{a \in A : y \leq a\}$.

As we will see, the above looks very similar to the Ω -rule. This provides a link between lattice theory and proof theory.

3 Parameter-free second order intuitionistic logic

3.1 Arithmetic

We here recall theories of inductive definitions. Let \mathbf{IS}_1 , \mathbf{PA} and $\mathbf{PA2}$ be the first order arithmetic with Σ_1^0 induction, that with full induction, and the second order arithmetic with full induction and comprehension, respectively. Given a theory T of arithmetic, $T[X]$ denotes the extension of T with a single set variable X and atomic formulas of the form $X(t)$.

A great many subsystems of $\mathbf{PA2}$ are considered in the literature. For instance, the system $\Pi_1^1\text{-CA}_0$ is obtained by restricting the induction and comprehension axiom schemata to Π_1^1 formulas. Even weaker are theories of iterated inductive definitions \mathbf{ID}_n with $n < \omega$, that are obtained as follows.

\mathbf{ID}_0 is just \mathbf{PA} . To obtain \mathbf{ID}_{n+1} , consider a formula $\varphi(X, x)$ in $\mathbf{ID}_n[X]$ which contains no first order free variables other than x and no negative occurrences of X . It can be seen as a monotone map $\varphi^{\mathbb{N}} : \wp(\mathbb{N}) \rightarrow \wp(\mathbb{N})$ sending a set $X \subseteq \mathbb{N}$ to $\{n \in \mathbb{N} : \mathbb{N} \models \varphi(X, n)\}$, so it has a least fixed point $I_\varphi^{\mathbb{N}}$. Based on this intuition, one adds a unary predicate symbol I_φ for each such φ to the language of \mathbf{ID}_n and axioms

$$\varphi(I_\varphi) \subseteq I_\varphi, \quad \varphi(\tau) \subseteq \tau \rightarrow I_\varphi \subseteq \tau$$

for every abstract $\tau = \lambda x.\xi(x)$ in the new language. Here $\varphi(I_\varphi)$ is a shorthand for the abstract $\lambda x.\varphi(I_\varphi, x)$ and $\tau_1 \subseteq \tau_2$ is for $\forall x.\tau_1(x) \rightarrow \tau_2(x)$. The induction schema is extended to the new language. This defines the system \mathbf{ID}_{n+1} . Notice that \mathbf{ID}_{n+1} does not involve any set variable. Finally, let $\mathbf{ID}_{<\omega}$ be the union of all \mathbf{ID}_n with $n < \omega$.

Clearly $\mathbf{ID}_{<\omega}$ can be seen as a subsystem of $\Pi_1^1\text{-CA}_0$. In fact, any fixed point atom $I_\varphi(t)$ can be replaced by second order formula

$$I_\varphi(t) := \forall X.\forall x(\varphi(X, x) \rightarrow X(x)) \rightarrow X(t).$$

Given a formula ψ of $\mathbf{ID}_{<\omega}$, we write ψ^I for the formula of $\mathbf{PA2}$ obtained by repeating the above replacement. This makes the axioms of $\mathbf{ID}_{<\omega}$ all provable in $\Pi_1^1\text{-CA}_0$.

The converse is not strictly true, but it is known that $\mathbf{ID}_{<\omega}$ has the same proof theoretic strength and the same arithmetical consequences with $\Pi_1^1\text{-CA}_0$.

Let us point out that a typical use of inductive definition is to define a provability predicate. Let T be a sequent calculus system, and suppose that we are given a formula $\varphi(X, x)$ saying that there is a rule in T with conclusion sequent x (coded by a natural number) and premises $Y \subseteq X$. Then $I_\varphi^{\mathbb{N}}$ gives the set of all provable sequents in T . Notice that the premise set Y can be infinite. It is for this reason that \mathbf{ID} -theories are suitable metatheories for infinitary proof systems. See [13] for more on inductive definitions.

3.2 Second order intuitionistic logic

In this subsection, we formally introduce sequent calculus $\mathbf{LI2}$ for the second order intuitionistic logic with full comprehension, that is an intuitionistic counterpart of Takeuti's classical calculus $\mathbf{G}^1\mathbf{LC}$ [31].

Consider a language L that consists of (first order) function symbols and predicate symbols. A typical example is the language $L_{\mathbf{PA}}$ of Peano arithmetic, which contains a predicate symbol for equality and function symbols for all primitive recursive functions. Let

- **Var**: a countable set of term variables x, y, z, \dots ,
- **Tm**(L): the set of first order terms t, u, v, \dots over L ,
- **VAR**: the set of set variables X, Y, Z, \dots

The set $\mathbf{FM}(L)$ of second order formulas is defined by:

$$\varphi, \psi ::= p(\vec{t}) \mid X(t) \mid \perp \mid \varphi \star \psi \mid Qx.\varphi \mid QX.\varphi,$$

where $p \in L$, $\star \in \{\wedge, \vee, \rightarrow\}$ and $Q \in \{\forall, \exists\}$. We define $\top := \perp \rightarrow \perp$. When the language L is irrelevant, we write $\mathbf{Tm} := \mathbf{Tm}(L)$ and $\mathbf{FM} := \mathbf{FM}(L)$. Given φ , let $\mathbf{FV}(\varphi)$ and $\mathbf{Fv}(\varphi)$ be the set of free set variables and that of free term variables in φ , respectively.

Typical formulas in $\mathbf{FM}(L_{\mathbf{PA}})$ are

$$\begin{aligned} \mathbf{N}(t) &:= \forall X. [\forall x (X(x) \rightarrow X(x+1)) \wedge X(0) \rightarrow X(t)], \\ \mathbf{E}(t) &:= \forall X. \forall x. [t = x \wedge X(x) \rightarrow X(t)]. \end{aligned}$$

We assume the standard variable convention that α -equivalent formulas are syntactically identical, so that substitutions can be applied without variable clash. A *term substitution* is a function $\circ : \mathbf{Var} \rightarrow \mathbf{Tm}$. Given $\varphi \in \mathbf{FM}$, the substitution instance φ° is defined as usual. Likewise, a *set substitution* is a function $\bullet : \mathbf{VAR} \rightarrow \mathbf{ABS}$, where $\mathbf{ABS} := \{\lambda x. \xi : \xi \in \mathbf{FM}\}$ is the set of *abstracts*. Instance φ^\bullet is obtained by replacing each atomic formula $X(t)$ with $X^\bullet(t)$ and applying β -reduction.

Let $\mathbf{SEQ} := \{\Gamma \Rightarrow \Pi : \Gamma, \Pi \subseteq_{\text{fin}} \mathbf{FM}, |\Pi| \leq 1\}$ be the set of *sequents of LI2*. We write Γ, Δ to denote $\Gamma \cup \Delta$. Rules of **LI2** include:

$$\begin{array}{c} \frac{}{\Gamma, \varphi \Rightarrow \varphi} \text{ (id)} \quad \frac{\varphi(\tau), \Gamma \Rightarrow \Pi}{\forall X. \varphi(X), \Gamma \Rightarrow \Pi} \text{ (\forall X left)} \quad \frac{\Gamma \Rightarrow \varphi(Y)}{\Gamma \Rightarrow \forall X. \varphi(X)} \text{ (\forall X right)} \\ \frac{\Gamma \Rightarrow \varphi \quad \varphi, \Gamma \Rightarrow \Pi}{\Gamma \Rightarrow \Pi} \text{ (cut)} \quad \frac{\varphi(Y), \Gamma \Rightarrow \Pi}{\exists X. \varphi(X), \Gamma \Rightarrow \Pi} \text{ (\exists X left)} \quad \frac{\Gamma \Rightarrow \varphi(\tau)}{\Gamma \Rightarrow \exists X. \varphi(X)} \text{ (\exists X right)} \end{array}$$

where $\tau \in \mathbf{ABS}$ and rules $(\forall X \text{ right})$ and $(\exists X \text{ left})$ are subject to the eigenvariable condition $Y \notin \mathbf{FV}(\Gamma, \Pi)$. The inference rules for other connectives can be found in Appendix A. The indicated occurrence of $\forall X. \varphi(X)$ in $(\forall X \text{ left})$ is the *main formula* and $\varphi(\tau)$ is the *minor formula* of rule $(\forall X \text{ left})$. The same terminology applies to other inference rules too.

A well known fact essentially due to [31] is that if a Π_2^0 sentence φ is provable in **PA2**, then $\forall y. \mathbf{E}(y), \Gamma^{\mathbf{N}} \Rightarrow \varphi^{\mathbf{N}}$ is provable in **LI2**, where Γ is a finite set of true Π_1^0 sentences (equality axioms, basic axioms of Peano arithmetic and defining axioms of primitive recursive functions), and $\varphi^{\mathbf{N}}$ is obtained from φ by relativizing each first order quantifier Qx to $Qx \in \mathbf{N}$. In particular if φ is Σ_1^0 , we obtain $\forall y. \mathbf{E}(y), \Gamma \Rightarrow \varphi$, and the assumption $\forall y. \mathbf{E}(y)$ can be eliminated by another relativization with respect to \mathbf{E} , so that we eventually obtain $\Gamma \Rightarrow \varphi$ in **LI2**. A consequence is that

$$\mathbf{IS}_1 \vdash \mathbf{CE}(\mathbf{LI2}) \rightarrow \mathbf{1CON}(\mathbf{PA2}),$$

where $\mathbf{CE}(\mathbf{LI2})$ is a Π_2^0 sentence stating that **LI2** admits cut elimination, and $\mathbf{1CON}(\mathbf{PA2})$ is that **PA2** is *1-consistent*, that is, all provable Σ_1^0 sentences are true.

Thus 1-consistency of **PA2** is reduced to cut elimination for **LI2**. We also have the converse, also provably in \mathbf{IS}_1 . The reason is that cut elimination for **LI2** is “locally” provable in **PA2**, that is, whenever $\mathbf{LI2} \vdash \Gamma \Rightarrow \Pi$, **PA2** proves a Σ_1^0 statement “ $\mathbf{LI2} \vdash^{cf} \Gamma \Rightarrow \Pi$ ” (that

is, “ $\Gamma \Rightarrow \Pi$ is cut-free provable in **LI2**”), and moreover, a derivation of the latter statement (in **PA2**) can be primitive recursively obtained from any derivation of the former (in **LI2**). Hence 1-consistency of **PA2** implies cut elimination for **LI2** (in **IS₁**). See [7] for a concise explanation.

The equivalence holds because **PA2** and **LI2** have a “matching” proof theoretic strength. We are going to introduce subsystems of **LI2** that match $\mathbf{ID}_{<\omega} = \bigcup_{n \in \omega} \mathbf{ID}_n$ in this sense.

3.3 Parameter-free fragments

Now let us introduce parameter-free subsystems of **LI2**. We first define the set $\mathbf{FMP}_n \subseteq \mathbf{FM}$ of *parameter-free formulas at level n* for every $n \geq -1$.

\mathbf{FMP}_{-1} is just the set of formulas in **FM** without second order quantifiers. It is also denoted by **Fm**. For $n \geq 0$, \mathbf{FMP}_n is defined by:

$$\varphi, \psi ::= p(\vec{t}) \mid t \in X \mid \perp \mid \varphi \star \psi \mid Qx.\varphi \mid QX.\xi,$$

where $\star \in \{\wedge, \vee, \rightarrow\}$, $Q \in \{\forall, \exists\}$ and ξ is any formula in \mathbf{FMP}_{n-1} such that $\mathbf{FV}(\xi) \subseteq \{X\}$. Thus $QX.\xi$ is free of set parameters, though may contain first order free variables. Finally, **FMP** is the union of all \mathbf{FMP}_n .

For instance, both $\mathbf{N}(t)$ and $\mathbf{E}(t)$ belong to \mathbf{FMP}_0 so that relativizations $\varphi^{\mathbf{N}}$, $\varphi^{\mathbf{E}}$ belong to \mathbf{FMP}_0 too, whenever φ is an arithmetical formula. Furthermore, each fixed point atom I_φ with φ arithmetical translates to

$$I_\varphi^{\mathbf{N}}(t) := \forall X.\forall x \in \mathbf{N}(\varphi^{\mathbf{N}}(X, x) \rightarrow X(x)) \rightarrow X(t),$$

that belongs to \mathbf{FMP}_1 . We write $\varphi^{I\mathbf{N}}$ to denote the translation of \mathbf{ID}_1 -formula φ in \mathbf{FMP}_1 . Likewise, any formula φ of \mathbf{ID}_n translates to a formula $\varphi^{I\mathbf{N}}$ in \mathbf{FMP}_n . On the other hand, second order definitions of positive connectives $\{\exists, \vee\}$:

$$\begin{aligned} \exists X.\varphi(X) &:= \forall Y.\forall X(\varphi(X) \rightarrow Y(*)) \rightarrow Y(*), \\ \varphi \vee \psi &:= \forall Y.(\varphi \rightarrow Y(*)) \wedge (\psi \rightarrow Y(*)) \rightarrow Y(*) \end{aligned}$$

with $Y \notin \mathbf{FV}(\varphi, \psi)$ and $*$ a constant, are no longer available. They do not belong to **FMP**, so restricting to the negative fragment $\{\forall, \wedge, \rightarrow\}$ causes a serious loss of expressivity in the parameter-free setting.

Sequent calculus **LIP** (resp. **LIP_n**) is obtained from **LI2** by restricting the formulas to **FMP** (resp. \mathbf{FMP}_n). Most importantly, when one applies rules ($\forall X$ left) and ($\exists X$ right) to introduce $QX.\varphi$, the minor formula $\varphi(\tau)$ must belong to **FMP** (resp. \mathbf{FMP}_n).

LIP is an intuitionistic counterpart of the classical calculus studied in [32], and **LIP₋₁** is just the ordinary sequent calculus for first order intuitionistic logic, that is also denoted by **LI**.

As before, arithmetical systems \mathbf{ID}_n reduce to logical systems **LIP_n**. For every Π_2^0 sentence φ of \mathbf{ID}_n , $\mathbf{ID}_n \vdash \varphi$ implies $\mathbf{LIP}_n \vdash \forall y.\mathbf{E}(y), \Gamma^{\mathbf{N}} \Rightarrow \varphi^{I\mathbf{N}}$, where Γ is a finite set of true Π_1^0 sentences. In particular, if φ is a Σ_1^0 sentence of **PA**, we obtain $\mathbf{LIP}_n \vdash \Gamma \Rightarrow \varphi$. As a consequence,

$$\mathbf{IS}_1 \vdash \mathbf{CE}(\mathbf{LIP}_n) \rightarrow \mathbf{1CON}(\mathbf{ID}_n), \quad \mathbf{IS}_1 \vdash \mathbf{CE}(\mathbf{LIP}) \rightarrow \mathbf{1CON}(\mathbf{ID}_{<\omega}).$$

The converse is obtained by proving cut elimination for **LIP_n** locally within \mathbf{ID}_n .

4 Ω -rule

4.1 Introduction to Ω -rule

Cut elimination in a higher order setting is tricky, since a principal reduction step

$$\frac{\frac{\Gamma \Rightarrow \varphi(Y)}{\Gamma \Rightarrow \forall X.\varphi(X)} (\forall X \text{ right}) \quad \frac{\varphi(\tau) \Rightarrow \Pi}{\forall X.\varphi(X) \Rightarrow \Pi} (\forall X \text{ left})}{\Gamma \Rightarrow \Pi} (\text{cut}) \quad \Longrightarrow \quad \frac{\Gamma \Rightarrow \varphi(\tau) \quad \varphi(\tau) \Rightarrow \Pi}{\Gamma \Rightarrow \Pi} (\text{cut})$$

may yield a bigger cut formula so that one cannot simply argue by induction on the complexity of the cut formula. The Ω -rule, introduced by [11], is an alternative of rule ($\forall X$ left) that allows us to circumvent this difficulty. Buchholz [12] includes an ordinal-free proof of (partial) cut elimination for a parameter-free subsystem \mathbf{BI}_1^- of analysis. It was later extended to complete cut elimination for the same system [4], and to complete cut elimination for $\Pi_1^1\text{-CA}_0 + \mathbf{BI}$ (bar induction) [3]. The Ω -rule further finds applications in modal fixed point logics [22, 25]. It is used to show strong normalization for the parameter-free fragments of System F, provably in \mathbf{ID} -theories [5].

As a starter, let us consider the most direct translation of the arithmetical Ω -rule [12] into our setting¹. We extend \mathbf{LI} by enlarging the formulas to \mathbf{FMP}_0 and adding rules ($\forall X$ right) and

$$\frac{\{ \Delta, \Gamma \Rightarrow \Pi \}_{\Delta \in |\forall X.\varphi|^b}}{\forall X.\varphi, \Gamma \Rightarrow \Pi} (\Omega^b)$$

where $|\forall X.\varphi|^b$ consists of $\Delta \subseteq_{\text{fin}} \mathbf{Fm}$ such that $\mathbf{LI} \vdash^{cf} \Delta \Rightarrow \varphi(Y)$ for some $Y \notin \mathbf{FV}(\Delta)$ (recall that “ cf ” indicates cut-free provability).

Rule (Ω^b) has infinitely many premises indexed by $|\forall X.\varphi|^b$. Observe a similarity with the characteristic rules of MacNeille completion (Proposition 3). In Section 5, we will provide a further link between them.

(Ω^b) is intended to be an alternative of ($\forall X$ left). Indeed, we can prove $\forall X.\varphi \Rightarrow \varphi(\tau)$ for an arbitrary abstract τ as follows. Let $\Delta \in |\forall X.\varphi|^b$, that is, $\mathbf{LI} \vdash^{cf} \Delta \Rightarrow \varphi(Y)$ for some $Y \notin \mathbf{FV}(\Delta)$. We then have $\Delta \Rightarrow \varphi(\tau)$ in the extended system by substituting τ for Y . Hence rule (Ω^b) yields $\forall X.\varphi \Rightarrow \varphi(\tau)$.

Moreover, rule (Ω^b) suggests a natural step of cut elimination. Consider a cut:

$$\frac{\frac{\Gamma \Rightarrow \varphi(Y)}{\Gamma \Rightarrow \forall X.\varphi(X)} (\forall X \text{ right}) \quad \frac{\{ \Delta \Rightarrow \Pi \}_{\Delta \in |\forall X.\varphi|^b}}{\forall X.\varphi \Rightarrow \Pi} (\Omega^b)}{\Gamma \Rightarrow \Pi} (\text{cut})$$

If $\Gamma \subseteq_{\text{fin}} \mathbf{Fm}$ and $\Gamma \Rightarrow \varphi(Y)$ is cut-free provable, then Γ belongs to $|\forall X.\varphi|^b$, so the conclusion $\Gamma \Rightarrow \Pi$ is just one of the infinitely many premises.

However, rule (Ω^b) cannot be combined with the standard rules for first order quantifiers.

► **Proposition 4.** *System $\mathbf{LI} + (\forall X \text{ right}) + (\Omega^b)$ is inconsistent.*

¹ Actually the original rule has assumptions indexed by *derivations* of $\Delta \Rightarrow \varphi(Y)$, not by Δ 's themselves. As an advantage, one obtains a concrete operator for cut elimination and reduces the complexity of inductive definition: the original semiformal system can be defined by inductive definition on a bounded formula, while ours requires a Π_1^1 formula. However, this point is irrelevant for the subsequent argument.

Proof. Consider formula $\varphi := X(c) \rightarrow X(x)$ with c a constant. We claim that $\forall X.\varphi \Rightarrow \perp$ is provable. Let $\Delta \in |\forall X.\varphi|^b$, that is, $\mathbf{LI} \vdash^{cf} \Delta \Rightarrow Y(c) \rightarrow Y(x)$ for some $Y \notin \mathbf{FV}(\Delta)$. Since the sequent is first order and $Y(c) \rightarrow Y(x)$ is not provable, Craig's interpolation theorem yields $\Delta \Rightarrow \perp$. Hence $\forall X.\varphi \Rightarrow \perp$ follows by (Ω^b) . Since both $\exists x.\forall X.\varphi \Rightarrow \perp$ and $\Rightarrow \exists x.\forall X.\varphi$ are provable, we obtain \perp . \blacktriangleleft

The primary reason for inconsistency is that (Ω^b) is not closed under term substitutions, while the standard treatment of first order quantifiers assumes that all rules are closed under term substitutions. Hence we have to weaken first order quantifier rules to obtain a consistent system. A reasonable way is to replace $(\forall x \text{ right})$ and $(\exists x \text{ left})$ with Schütte's ω -rules:

$$\frac{\{\Gamma \Rightarrow \varphi(t)\}_{t \in \mathcal{Tm}}}{\Gamma \Rightarrow \forall x.\varphi(x)} \quad (\omega \text{ right}) \qquad \frac{\{\varphi(t), \Gamma \Rightarrow \Pi\}_{t \in \mathcal{Tm}}}{\exists x.\varphi(x), \Gamma \Rightarrow \Pi} \quad (\omega \text{ left})$$

This allows us to prove *partial* cut elimination: if a sequent $\Gamma \Rightarrow \Pi$ is provable, then it is cut-free provable, *provided that* $\Gamma \cup \Pi \subseteq \mathbf{Fm}$. To prove *complete* cut elimination, we need to work with more sophisticated calculi.

4.2 Cut elimination by Ω -rule

We now introduce a family of infinitary sequent calculi and use them to prove complete cut elimination for \mathbf{LIP} . The proof idea is entirely due to [3].

We first prepare an isomorphic copy of each \mathbf{FMP}_n , denoted by $\overline{\mathbf{FMP}}_n$. $\overline{\mathbf{FMP}}_{-1}$ is just $\mathbf{FMP}_{-1} = \mathbf{Fm}$. For $n \geq 0$, $\overline{\mathbf{FMP}}_n$ is defined by:

$$\vartheta, \vartheta' ::= p(\vec{t}) \mid t \in X \mid \perp \mid \vartheta \star \vartheta' \mid Qx.\vartheta \mid \overline{\forall}X.\chi \mid \overline{\exists}X.\chi,$$

where $\star \in \{\wedge, \vee, \rightarrow\}$, $Q \in \{\forall, \exists\}$ and χ is any formula in $\overline{\mathbf{FMP}}_{n-1}$ such that $\mathbf{FV}(\chi) \subseteq \{X\}$. Given $\vartheta \in \overline{\mathbf{FMP}} := \bigcup \overline{\mathbf{FMP}}_n$, its *level* is defined by $\text{level}(\vartheta) := \min\{k : \vartheta \in \overline{\mathbf{FMP}}_k\}$. Given a formula $\varphi \in \mathbf{FMP}$, $\overline{\varphi} \in \overline{\mathbf{FMP}}$ is obtained by overlining all the second order quantifiers in it.

We are going to introduce a hybrid calculus $\mathbf{LI}\Omega_n$ for each $n \geq -1$ in which sequents are made of formulas in $\mathbf{FMP} \cup \overline{\mathbf{FMP}}_n$. Those in $\overline{\mathbf{FMP}}_n$ are intended to be potential cut formulas, i.e., ancestors of cut formulas in a derivation (called *implicit* in [32]), and are treated by using Ω -rules. Those in \mathbf{FMP} are remaining formulas, that are treated as in \mathbf{LIP} .

Calculus $\mathbf{LI}\Omega_{-1}$ is just \mathbf{LIP} where sequents consist of formulas in $\mathbf{FMP} = \mathbf{FMP} \cup \overline{\mathbf{FMP}}_{-1}$ and cut formulas are restricted to $\mathbf{Fm} = \overline{\mathbf{FMP}}_{-1}$.

Suppose that $\mathbf{LI}\Omega_{k-1}$ has been defined for every $0 \leq k \leq n$. For each $\overline{\forall}X.\vartheta$ and $\overline{\exists}X.\vartheta$ of level k , let

$$\begin{aligned} |\overline{\forall}X.\vartheta(X)| &:= \{\Delta : \mathbf{LI}\Omega_{k-1} \vdash^{cf} \Delta \Rightarrow \vartheta(Y) \text{ for some } Y \notin \mathbf{FV}(\Delta)\} \\ |\overline{\exists}X.\vartheta(X)| &:= \{(\Delta \Rightarrow \Lambda) : \mathbf{LI}\Omega_{k-1} \vdash^{cf} \vartheta(Y), \Delta \Rightarrow \Lambda \text{ for some } Y \notin \mathbf{FV}(\Delta, \Lambda)\}. \end{aligned}$$

Note that $\Delta \cup \Lambda \subseteq \mathbf{FMP} \cup \overline{\mathbf{FMP}}_{k-1}$. Calculus $\mathbf{LI}\Omega_n$ is defined as follows:

- Sequents consist of formulas in $\mathbf{FMP} \cup \overline{\mathbf{FMP}}_n$.
- Cut formulas are restricted to $\overline{\mathbf{FMP}}_n$.
- First order quantifiers are treated by rules $(\forall x \text{ left})$, $(\exists x \text{ right})$, $(\omega \text{ right})$ and $(\omega \text{ left})$.
- Second order quantifiers in \mathbf{FMP} are treated by rules $(\forall X \text{ left})$, $(\forall X \text{ right})$, $(\exists X \text{ left})$ and $(\exists X \text{ right})$ as in \mathbf{LIP} .

- Second order quantifiers in $\overline{\text{FMP}}_n$ are treated by the following rules ($k = 0, \dots, n$):

$$\frac{\vartheta(Y), \Gamma \Rightarrow \Pi}{\overline{\exists}X.\vartheta(X), \Gamma \Rightarrow \Pi} (\overline{\exists}X \text{ left}) \qquad \frac{\Gamma \Rightarrow \vartheta(Y)}{\Gamma \Rightarrow \overline{\forall}X.\vartheta(X)} (\overline{\forall}X \text{ right})$$

$$\frac{\{\Delta, \Gamma \Rightarrow \Pi\}_{\Delta \in |\overline{\forall}X.\vartheta|}}{\overline{\forall}X.\vartheta, \Gamma \Rightarrow \Pi} (\Omega_k \text{ left}) \qquad \frac{\Gamma \Rightarrow \vartheta(Y) \quad \{\Delta, \Gamma \Rightarrow \Pi\}_{\Delta \in |\overline{\forall}X.\vartheta|}}{\Gamma \Rightarrow \Pi} (\tilde{\Omega}_k \text{ left})$$

$$\frac{\{\Gamma, \Delta \Rightarrow \Lambda\}_{(\Delta \Rightarrow \Lambda) \in |\overline{\exists}X.\vartheta|}}{\Gamma \Rightarrow \overline{\exists}X.\vartheta} (\Omega_k \text{ right}) \qquad \frac{\{\Gamma, \Delta \Rightarrow \Lambda\}_{(\Delta \Rightarrow \Lambda) \in |\overline{\exists}X.\vartheta|} \quad \vartheta(Y), \Gamma \Rightarrow \Pi}{\Gamma \Rightarrow \Pi} (\tilde{\Omega}_k \text{ right})$$

where k is the level of $\overline{\forall}X.\vartheta$, $\overline{\exists}X.\vartheta$ and rules $(\overline{\exists}X \text{ left})$, $(\overline{\forall}X \text{ right})$, $(\tilde{\Omega}_k \text{ left})$ and $(\tilde{\Omega}_k \text{ right})$ are subject to the eigenvariable condition ($Y \notin \text{FV}(\Gamma, \Pi)$).

- Other connectives are treated as in **LIP**. See Appendix A for a complete list of inference rules.

It is admittedly complicated. First of all, notice that the rule $(\tilde{\Omega}_k \text{ left})$ is derivable by combining $(\overline{\forall}X \text{ right})$, $(\Omega_k \text{ left})$ and (cut) . It is nevertheless included for a technical reason. The same applies to rule $(\tilde{\Omega}_k \text{ right})$.

On the other hand, rules $(\Omega_k \text{ left})$ and $(\Omega_k \text{ right})$ are our real concern. The former should be read as follows: whenever $\mathbf{LI}\Omega_{k-1} \vdash^{cf} \Delta \Rightarrow \vartheta(Y)$ implies $\mathbf{LI}\Omega_n \vdash \Delta, \Gamma \Rightarrow \Pi$ for every Δ with $Y \notin \text{FV}(\Delta)$, one can conclude $\mathbf{LI}\Omega_n \vdash \overline{\forall}X.\vartheta, \Gamma \Rightarrow \Pi$.

Now let us list some key lemmas for cut elimination. The proofs are found in the full version.

► **Lemma 5** (Embedding). $\mathbf{LIP}_n \vdash \Gamma \Rightarrow \Pi$ implies $\mathbf{LI}\Omega_n \vdash \Gamma \Rightarrow \Pi$.

► **Lemma 6**. $\mathbf{LI}\Omega_n \vdash \Gamma \Rightarrow \Pi$ implies $\mathbf{LI}\Omega_n \vdash^{cf} \Gamma \Rightarrow \Pi$.

► **Lemma 7** (Collapsing). $\mathbf{LI}\Omega_n \vdash^{cf} \Gamma \Rightarrow \Pi$ implies $\mathbf{LI}\Omega_{n-1} \vdash^{cf} \Gamma \Rightarrow \Pi$, provided that $\Gamma \cup \Pi \subseteq \text{FMP} \cup \overline{\text{FMP}}_{n-1}$.

Proof. By induction on the length of the cut-free derivation of $\Gamma \Rightarrow \Pi$ in $\mathbf{LI}\Omega_n$. If it ends with $(\tilde{\Omega}_n \text{ left})$ (see above), we have $\mathbf{LI}\Omega_{n-1} \vdash^{cf} \Gamma \Rightarrow \vartheta(Y)$ by the induction hypothesis, noting that $\vartheta(Y) \in \overline{\text{FMP}}_{n-1}$. Hence $\Gamma \in |\overline{\forall}X.\vartheta|$, so $\Gamma, \Gamma \Rightarrow \Pi$ is among the premises. Therefore $\mathbf{LI}\Omega_{n-1} \vdash^{cf} \Gamma \Rightarrow \Pi$ by the induction hypothesis again.

Rule $(\tilde{\Omega}_n \text{ left})$ is treated similarly. When $n = 0$, one has to replace $(\omega \text{ right})$ and $(\omega \text{ left})$ by $(\forall x \text{ right})$ and $(\exists x \text{ left})$ respectively, that is easy. ◀

► **Theorem 8** (Cut elimination). $\mathbf{LIP} \vdash \Gamma \Rightarrow \Pi$ implies $\mathbf{LIP} \vdash^{cf} \Gamma \Rightarrow \Pi$.

Proof. The sequent is provable in \mathbf{LIP}_n for some $n < \omega$, so in $\mathbf{LI}\Omega_n$ by Lemma 5. Noting that $\Gamma \cup \Pi \subseteq \text{FMP}$, we obtain a cut-free derivation in $\mathbf{LI}\Omega_{-1}$ by Lemmas 6 and 7, that is also a cut-free derivation in **LIP**. ◀

Of course the above argument can be restricted to a proof of cut elimination for \mathbf{LIP}_n . From a metatheoretical point of view, the most significant part is to define provability predicates $\mathbf{LI}\Omega_{-1}, \dots, \mathbf{LI}\Omega_n$. $\mathbf{LI}\Omega_{-1}$ is finitary, so is definable in $\mathbf{PA} = \mathbf{ID}_0$. $\mathbf{LI}\Omega_0$ is obtained by an inductive definition relying on $\mathbf{LI}\Omega_{-1}$, so is definable in \mathbf{ID}_1 . By repetition, we observe that $\mathbf{LI}\Omega_n$ is definable in \mathbf{ID}_{n+1} . Moreover, $\mathbf{LI}\Omega$ is definable with a uniform inductive definition in \mathbf{ID}_ω . Once a suitable provability predicate has been defined, the rest of argument can be smoothly formalized. Hence we obtain a folklore:

$$\mathbf{ID}_{n+1} \vdash \text{CE}(\mathbf{LIP}_n), \quad \mathbf{ID}_\omega \vdash \text{CE}(\mathbf{LIP}).$$

5 Ω -rule and MacNeille completion

In this section, we establish a formal connection between the Ω -rule and the MacNeille completion. Let us start by introducing algebraic semantics for full second order calculus **LI2**.

Let L be a language. A (complete) *Heyting-valued prestructure* for L is $\mathcal{M} = \langle \mathbf{A}, M, \mathcal{D}, \mathcal{L} \rangle$ where $\mathbf{A} = \langle A, \wedge, \vee, \rightarrow, \top, \perp \rangle$ is a complete Heyting algebra, M is a nonempty set (*term domain*), $\emptyset \neq \mathcal{D} \subseteq A^M$ (*abstract domain*) and \mathcal{L} consists of a function $f^{\mathcal{M}} : M^n \rightarrow M$ for each n -ary function symbol $f \in L$ and $p^{\mathcal{M}} : M^n \rightarrow A$ for each n -ary predicate symbol $p \in L$. Thus $p^{\mathcal{M}}$ is an \mathbf{A} -valued subset of M^n .

It is not our purpose to systematically develop a model theory for intuitionistic logic. We will use prestructures only for proving conservative extension and cut elimination. Hence we assume $M = \mathsf{Tm}$ and $f^{\mathcal{M}}(\vec{t}) = f(\vec{t})$ below, that simplifies the interpretation of formulas a lot.

A *valuation* on \mathcal{M} is a function $\mathcal{V} : \mathsf{VAR} \rightarrow \mathcal{D}$. The *interpretation* of formulas $\mathcal{V} : \mathsf{FM} \rightarrow \mathbf{A}$ is inductively defined as follows:

$$\begin{array}{llll} \mathcal{V}(p(\vec{t})) & := & p^{\mathcal{M}}(\vec{t}) & \mathcal{V}(X(t)) & := & \mathcal{V}(X)(t) \\ \mathcal{V}(\perp) & := & \perp & \mathcal{V}(\varphi \star \psi) & := & \mathcal{V}(\varphi) \star \mathcal{V}(\psi) \\ \mathcal{V}(\forall x.\varphi(x)) & := & \bigwedge_{t \in \mathsf{Tm}} \mathcal{V}(\varphi(t)) & \mathcal{V}(\exists x.\varphi(x)) & := & \bigvee_{t \in \mathsf{Tm}} \mathcal{V}(\varphi(t)) \\ \mathcal{V}(\forall X.\varphi) & := & \bigwedge_{F \in \mathcal{D}} \mathcal{V}[F/X](\varphi) & \mathcal{V}(\exists X.\varphi) & := & \bigvee_{F \in \mathcal{D}} \mathcal{V}[F/X](\varphi) \end{array}$$

where $\star \in \{\wedge, \vee, \rightarrow\}$ and $\mathcal{V}[F/X]$ is an update of \mathcal{V} that maps X to F . \mathcal{V} can also be extended to a function $\mathcal{V} : \mathsf{ABS} \rightarrow \mathbf{A}^{\mathsf{Tm}}$ by $\mathcal{V}(\lambda x.\varphi)(t) := \mathcal{V}(\varphi[t/x])$. \mathcal{M} is called a *Heyting-valued structure* if $\mathcal{V}(\tau) \in \mathcal{D}$ holds for every valuation \mathcal{V} and every $\tau \in \mathsf{ABS}$. Clearly \mathcal{M} is a Heyting-valued structure if $\mathcal{D} = \mathbf{A}^{\mathsf{Tm}}$. Such a structure is called *full*.

Given a sequent $\Gamma \Rightarrow \Pi$, let $\mathcal{V}(\Gamma) := \bigwedge \{\mathcal{V}(\varphi) : \varphi \in \Gamma\}$ ($:= \top$ if Γ is empty). $\mathcal{V}(\Pi) := \mathcal{V}(\psi)$ if $\Pi = \{\psi\}$, and $\mathcal{V}(\Pi) := \perp$ if Π is empty. It is routine to verify:

► **Lemma 9** (Soundness). *If $\mathbf{LI2} \vdash \Gamma \Rightarrow \Pi$, then $\Gamma \Rightarrow \Pi$ is valid, that is, $\mathcal{V}(\Gamma^\circ) \leq \mathcal{V}(\Pi^\circ)$ holds for every valuation \mathcal{V} on every Heyting structure \mathcal{M} and every term substitution \circ .*

To illustrate use of algebraic semantics, we prove an elementary fact that **LI2** is a conservative extension of **LI**.

Let \mathbf{L} be the Lindenbaum algebra for **LI**, that is, $\mathbf{L} := \langle \mathsf{Fm}/\sim, \wedge, \vee, \rightarrow, \top, \perp \rangle$ where $\varphi \sim \psi$ iff $\mathbf{LI} \vdash \varphi \leftrightarrow \psi$. The equivalence class of φ with respect to \sim is denoted by $[\varphi]$. \mathbf{L} is a Heyting algebra in which

$$(*) \quad [\forall x.\varphi(x)] = \bigwedge_{t \in \mathsf{Tm}} [\varphi(t)], \quad [\exists x.\varphi(x)] = \bigvee_{t \in \mathsf{Tm}} [\varphi(t)]$$

hold. Given a sequent $\Gamma \Rightarrow \Pi$, elements $[\Gamma]$ and $[\Pi]$ in \mathbf{L} are naturally defined.

Let \mathbf{G} be a *regular* completion of \mathbf{L} . Then $\mathcal{M}(\mathbf{G}) := \langle \mathbf{G}, \mathsf{Tm}, \mathbf{G}^{\mathsf{Tm}}, \mathcal{L} \rangle$ is a full Heyting structure, where \mathcal{L} consists of a \mathbf{G} -valued predicate $p^{\mathcal{M}(\mathbf{G})}$ defined by $p^{\mathcal{M}(\mathbf{G})}(\vec{t}) := [p(\vec{t})]$ for each $p \in L$ (in addition to interpretations of function symbols). Define a valuation \mathcal{I} by $\mathcal{I}(X)(t) := [X(t)]$. We then have $\mathcal{I}(\varphi) = [\varphi]$ for every $\varphi \in \mathsf{Fm}$ by regularity (be careful here: $(*)$ may fail in \mathbf{G} if it is not regular).

Now, suppose that **LI2** proves $\Gamma \Rightarrow \Pi$ with $\Gamma \cup \Pi \subseteq \mathsf{Fm}$. Then we have $\mathcal{I}(\Gamma) \leq \mathcal{I}(\Pi)$ by Lemma 9, so $[\Gamma] \leq [\Pi]$, that is, $\mathbf{LI} \vdash \Gamma \Rightarrow \Pi$. This proves that **LI2** is a conservative extension of **LI**.

Although this argument cannot be fully formalized in **PA2** because of Gödel's second incompleteness, it does admit a local formalization in **PA2**. In contrast, the above argument, when applied to **LIP_n**, cannot be locally formalized in **ID_n**. The reason is simply that

\mathbf{ID}_n does not have second order quantifiers, which are needed to write down the definitions of $\mathcal{V}(\forall X.\varphi)$ and $\mathcal{V}(\exists X.\varphi)$. To circumvent this, a crucial observation is that $\mathcal{V}(\forall X.\varphi)$ and $\mathcal{V}(\exists X.\varphi)$ admit alternative first order definitions if the completion is MacNeille. It is here that one finds a connection between the MacNeille completion and the Ω -rule.

► **Theorem 10.** *Let \mathbf{L} be the Lindenbaum algebra for \mathbf{LI} and $\mathbf{L} \subseteq \mathbf{G}$ a regular completion. $\mathcal{M}(\mathbf{G})$ and \mathcal{I} are defined as above. For every sentence $\forall X.\varphi$ in \mathbf{FMP}_0 , the following are equivalent.*

1. $\mathcal{I}(\forall X.\varphi) = \bigvee \{a \in \mathbf{L} : a \leq \mathcal{I}(\forall X.\varphi)\}$.
2. $\mathcal{I}(\forall X.\varphi) = \bigvee \{[\Delta] \in \mathbf{L} : \Delta \in |\forall X.\varphi|^b\}$.
3. *The inference below is sound for every $y \in \mathbf{G}$:*

$$\frac{\{ \mathcal{I}(\Delta) \leq y \}_{\Delta \in |\forall X.\varphi|^b}}{\mathcal{I}(\forall X.\varphi) \leq y}$$

If \mathbf{G} is the MacNeille completion of \mathbf{F} , all the above hold.

Proof. (1. \Leftrightarrow 2.) Let $a = [\Delta]$. It is sufficient to prove that $a \leq \mathcal{I}(\forall X.\varphi)$ iff $\Delta \in |\forall X.\varphi|^b$, i.e., $\mathbf{LI} \vdash^{cf} \Delta \Rightarrow \varphi(Y)$ for some $Y \notin \mathbf{FV}(\Delta)$. If $a \leq \mathcal{I}(\forall X.\varphi)$, choose $Y \notin \mathbf{FV}(\Delta)$ and let $F_Y(t) := [Y(t)]$. We then have $[\Delta] \leq \mathcal{I}[F_Y/X](\varphi(X)) = [\varphi(Y)]$, that is, $\mathbf{LI} \vdash \Delta \Rightarrow \varphi(Y)$. By cut elimination for \mathbf{LI} , we obtain $\mathbf{LI} \vdash^{cf} \Delta \Rightarrow \varphi(Y)$. Conversely, suppose that $\mathbf{LI} \vdash^{cf} \Delta \Rightarrow \varphi(Y)$ with $Y \notin \mathbf{FV}(\Delta)$. It implies $[\Delta] = \mathcal{I}(\Delta) = \mathcal{I}[F/Y](\Delta) \leq \mathcal{I}[F/Y](\varphi(Y))$ for every $F \in \mathbf{G}^{\text{tm}}$ by Lemma 9. Hence $[\Delta] \leq \mathcal{I}(\forall X.\varphi(X))$.

(2. \Rightarrow 3.) Straightforward by noting that $[\Delta] = \mathcal{I}(\Delta)$.

(3. \Rightarrow 2.) Let $y := \bigvee \{[\Delta] \in \mathbf{L} : \Delta \in |\forall X.\varphi|^b\}$. Then $\mathcal{I}(\Delta) = [\Delta] \leq y$ holds for every $\Delta \in |\forall X.\varphi|^b$, so $\mathcal{I}(\forall X.\varphi) \leq y$ by 3. Since $\Delta \in |\forall X.\varphi|^b$ implies $[\Delta] \leq \mathcal{I}(\forall X.\varphi)$ as proved above, we also have $y \leq \mathcal{I}(\forall X.\varphi)$. ◀

The equivalence in Theorem 10 is quite suggestive, since 3. is an algebraic interpretation of rule (Ω^b), while 1. is a characteristic of the MacNeille completion (Proposition 3). Equation 2. suggests a way of interpreting second order formulas without using second order quantifiers at the meta-level. All these are true if the completion is MacNeille. It should be mentioned that essentially the same as 2. has been already observed by Altenkirch and Coquand [6] in the context of lambda calculus (without making any connection to the Ω -rule and the MacNeille completion). Indeed, they consider a logic which roughly amounts to the negative fragment of our \mathbf{LIP}_0 and employ equation 2. to give a “finitary” proof of (partial) normalization theorem for a parameter-free fragment of System F (see also [2, 5] for extensions). However, their argument is technically based on a downset completion, that is not MacNeille. As is well known, such a naive completion does not work well for the positive connectives $\{\exists, \vee\}$. In contrast, when \mathbf{G} is the MacNeille completion of \mathbf{L} , we also have

$$\mathcal{I}(\exists X.\varphi) = \bigwedge \{[\Delta] \rightarrow [\Lambda] \in \mathbf{L} : (\Delta \Rightarrow \Lambda) \in |\exists X.\varphi|^b\},$$

where $(\Delta \Rightarrow \Lambda) \in |\exists X.\varphi|^b$ iff $\mathbf{LI} \vdash^{cf} \varphi(Y), \Delta \Rightarrow \Lambda$ for some $Y \notin \mathbf{FV}(\Delta, \Lambda)$. We thus claim that the insight by Altenkirch and Coquand is augmented and better understood in terms of the MacNeille completion.

It is interesting to see that (second order) \forall is interpreted by (first order) \bigvee while \exists is by \bigwedge . We call this style of interpretation the Ω -interpretation, that is the algebraic side of the Ω -rule, and that will play a key role in the next section. We conclude our discussion by reporting a counterexample for general soundness.

► **Proposition 11.** *There is a Heyting-valued structure in which (Ω^b) is not sound.*

Proof. Let A be the three-element chain $\{0 < 0.5 < 1\}$ seen as a Heyting algebra. Consider the language that only consists of a term constant $*$. Then a full Heyting-valued structure $\mathcal{A} := \langle \mathbf{A}, \text{Tm}, \mathbf{A}^{\text{Tm}}, \mathcal{L} \rangle$ is naturally obtained. Let $\varphi := (X(*) \rightarrow \perp) \vee X(*)$. It is easy to see that $\mathcal{V}(\forall X.\varphi) = 0.5$ for every valuation \mathcal{V} .

Now consider the following instance:

$$\frac{\{ \Delta \Rightarrow \perp \}_{\Delta \in |\forall X.\varphi|^b}}{\forall X.\varphi \Rightarrow \perp} (\Omega^b)$$

We claim that it is not sound for a valuation \mathcal{V} such that $\mathcal{V}(X(t)) = 0$ for every $X \in \text{VAR}$ and $t \in \text{Tm}$. Suppose that $\Delta \in |\forall X.\varphi|^b$, i.e., $\mathbf{LI} \vdash^{cf} \Delta \Rightarrow \varphi(Y)$ with $Y \notin \text{FV}(\Delta)$. Then $\mathcal{V}(\Delta) \leq \bigwedge_{F \in \mathbf{A}^{\text{Tm}}} \mathcal{V}[F/X](\varphi) = 0.5$ by Lemma 9. But Δ is first order, so only takes value 0 or 1 under our assumption on \mathcal{V} . Hence $\mathcal{V}(\Delta) = 0$, that is, all premises are satisfied. However, $\mathcal{V}(\forall X.\varphi) = 0.5 > 0$, that is, the conclusion is not satisfied. ◀

This invokes a natural question. Is it possible to find a *Boolean-valued* counterexample? In other words, is the Ω -rule classically sound? This question is left open.

6 Algebraic cut elimination

6.1 Polarities and Heyting frames

This section is devoted to algebraic proofs of cut elimination. We begin with a very old concept due to Birkhoff [10], that provides a uniform framework for both MacNeille completion and cut elimination.

A *polarity* $\mathbf{W} = \langle W, W', R \rangle$ consists of two sets W, W' and a binary relation $R \subseteq W \times W'$. Given $X \subseteq W$ and $Z \subseteq W'$, let

$$X^\triangleright := \{z \in W' : x R z \text{ for every } x \in X\}, \quad Z^\triangleleft := \{x \in W : x R z \text{ for every } z \in Z\}.$$

For example, let $\mathbf{Q} := \langle \mathbb{Q}, \mathbb{Q}, \leq \rangle$. Then X^\triangleright is the set of upper bounds of X and Z^\triangleleft is the set of lower bounds of Z . Hence $(X^{\triangleright\triangleleft}, X^\triangleright)$ is a Dedekind cut for every $X \subseteq \mathbb{Q}$ bounded above.

The pair $(\triangleright, \triangleleft)$ forms a *Galois connection*:

$$X \subseteq Z^\triangleleft \iff X^\triangleright \supseteq Z$$

so induces a closure operator $\gamma(X) := X^{\triangleright\triangleleft}$ on $\wp(W)$, that is, $X \subseteq \gamma(Y)$ iff $\gamma(X) \subseteq \gamma(Y)$ for any $X, Y \subseteq W$. Note that $X \subseteq W$ is closed iff there is $Z \subseteq W'$ such that $X = Z^\triangleleft$.

In the following, we write $\gamma(x) := \gamma(\{x\})$, $x^\triangleright := \{x\}^\triangleright$ and $z^\triangleleft := \{z\}^\triangleleft$. Let

$$\mathcal{G}(\mathbf{W}) := \{X \subseteq W : X = \gamma(X)\},$$

$$X \wedge Y := X \cap Y, \quad X \vee Y := \gamma(X \cup Y), \quad \top := W \text{ and } \perp := \gamma(\emptyset).$$

► **Lemma 12.** *If \mathbf{W} is a polarity, then $\mathbf{W}^+ := \langle \mathcal{G}(\mathbf{W}), \wedge, \vee \rangle$ is a complete lattice.*

The lattice \mathbf{W}^+ is not always distributive because of the use of γ in the definition of \vee . To ensure distributivity, we have to impose a further structure on \mathbf{W} .

A *Heyting frame* is $\mathbf{W} = \langle W, W', R, \circ, \varepsilon, \backslash \rangle$, where

- $\langle W, W', R \rangle$ is a polarity,
- $\langle W, \circ, \varepsilon \rangle$ is a monoid,

- $\parallel : W \times W' \longrightarrow W'$ satisfies $x \circ y R z \iff y R x \parallel z$ for every $x, y \in W$ and $z \in W'$,
- the following inferences are valid:

$$\frac{x \circ y R z}{y \circ x R z} (e) \quad \frac{\varepsilon R z}{x R z} (w) \quad \frac{x \circ x R z}{x R z} (c)$$

Clearly $x R z$ is an analogue of a sequent and (e), (w) and (c) correspond to exchange, weakening and contraction rules. By removing some/all of them, one obtains *residuated frames* that work for substructural logics as well [19, 16].

► **Lemma 13.** *If \mathbf{W} is a Heyting frame, $\mathbf{W}^+ := \langle \mathcal{G}(\mathbf{W}), \wedge, \vee, \rightarrow, \top, \perp \rangle$ is a complete Heyting algebra, where $X \rightarrow Y := \{y \in W : x \circ y \in Y \text{ for every } x \in X\}$.*

Polarities and Heyting frames are handy devices to obtain MacNeille completions. Let $\mathbf{A} = \langle A, \wedge, \vee, \rightarrow, \top, \perp \rangle$ be a Heyting algebra. Then $\mathbf{W}_{\mathbf{A}} := \langle A, A, \leq, \wedge, \top, \rightarrow \rangle$ is a Heyting frame. Notice that the third condition above amounts to $x \wedge y \leq z$ iff $y \leq x \rightarrow z$.

► **Theorem 14.** *If \mathbf{A} is a Heyting algebra, then $\gamma : \mathbf{A} \longrightarrow \mathbf{W}_{\mathbf{A}}^+$ is a MacNeille completion.*

6.2 Algebraic cut elimination for full second order logic

We here outline an algebraic proof of cut elimination for the full second order calculus **LI2** that we attribute to Maehara [24] and Okada [26, 28]. This will be useful for a comparison with the parameter-free case **LIP**_{*n*+1}, that is to be discussed in the next subsection.

Let $\wp_{\text{fin}}(\text{FM})$ be the set of finite sets of formulas, so that $\langle \wp_{\text{fin}}(\text{FM}), \cup, \emptyset \rangle$ is a commutative idempotent monoid. Recall that **SEQ** denotes the set of sequents of **LI2**. There is a natural map $\parallel : \wp_{\text{fin}}(\text{FM}) \times \text{SEQ} \longrightarrow \text{SEQ}$ defined by $\Gamma \parallel (\Sigma \Rightarrow \Pi) := (\Gamma, \Sigma \Rightarrow \Pi)$. So

$$\mathbf{CF} := \langle \wp_{\text{fin}}(\text{FM}), \text{SEQ}, \Rightarrow_{\mathbf{LI2}}^{\text{cf}}, \cup, \emptyset, \parallel \rangle$$

is a Heyting frame, where $\Gamma \Rightarrow_{\mathbf{LI2}}^{\text{cf}} (\Sigma \Rightarrow \Pi)$ iff **LI2** $\vdash^{\text{cf}} \Gamma, \Sigma \Rightarrow \Pi$. In the following, we simply write φ for sequent $(\emptyset \Rightarrow \varphi) \in \text{SEQ}$.

CF is a frame in which $\Gamma \in \varphi^{\triangleleft}$ holds iff $\Gamma \Rightarrow \varphi$ is cut-free provable in **LI2**. In particular, $\varphi \in \varphi^{\triangleleft}$ always holds, so $\varphi \in \gamma(\varphi) \subseteq \varphi^{\triangleleft}$. It should also be noted that each $X \in \mathcal{G}(\mathbf{CF})$ is closed under weakening: if $\Delta \in X$ and $\Delta \subseteq \Sigma$, then $\Sigma \in X$.

Define a Heyting prestructure $\mathcal{CF} := \langle \mathbf{CF}^+, \text{Tm}, \mathcal{D}, \mathcal{L} \rangle$ by $p^{\mathcal{CF}}(\vec{t}) := \gamma(p(\vec{t}))$ for each predicate symbol p and

$$\mathcal{D} := \{F \in \mathcal{G}(\mathbf{CF})^{\text{Tm}} : F \text{ matches some } \tau \in \text{ABS}\},$$

where F matches $\lambda x. \xi(x)$ just in case $\xi(t) \in F(t) \subseteq \xi(t)^{\triangleleft}$ holds for every $t \in \text{Tm}$. This choice of $\mathcal{D} \subseteq \mathcal{G}(\mathbf{CF})^{\text{Tm}}$ is a logical analogue of Girard's *reducibility candidates* as noticed by Okada.

Given a set substitution \bullet and a valuation $\mathcal{V} : \text{VAR} \longrightarrow \mathcal{D}$, we say that \mathcal{V} matches \bullet if $\mathcal{V}(X)$ matches $X^{\bullet} \in \text{ABS}$ for every $X \in \text{VAR}$. That is, $X^{\bullet}(t) \in \mathcal{V}(X)(t) \subseteq X^{\bullet}(t)^{\triangleleft}$ holds for every $X \in \text{VAR}$ and $t \in \text{Tm}$. The following is what Okada [28] calls his *main lemma*.

► **Lemma 15.** *Let $\bullet : \text{VAR} \longrightarrow \text{ABS}$ be a substitution and \mathcal{V} be a valuation that matches \bullet . Then for every $\varphi \in \text{FM}$,*

$$\varphi^{\bullet} \in \mathcal{V}(\varphi) \subseteq \varphi^{\bullet \triangleleft}.$$

As a consequence, $\mathcal{V}(\tau) \in \mathcal{D}$ for every $\tau \in \text{ABS}$ (recall that $\mathcal{V}(\lambda x.\xi(x))(t) := \mathcal{V}(\xi(t))$). That is, \mathcal{CF} is a Heyting structure. For another consequence, define a valuation \mathcal{I} by $\mathcal{I}(X)(t) := \gamma(X(t))$, that matches the identity substitution. Then we have $\varphi \in \mathcal{I}(\varphi) \subseteq \varphi^\triangleleft$. More generally, for every sequent $\Gamma \Rightarrow \Pi$ we have $\Gamma \in \mathcal{I}(\Gamma)$ (by closure under weakening and $\mathcal{I}(\Gamma) = \bigcap \{\mathcal{I}(\varphi) : \varphi \in \Gamma\}$) and $\mathcal{I}(\Pi) \subseteq \Pi^\triangleleft$.

► **Theorem 16** (Completeness and cut elimination). *For every sequent $\Gamma \Rightarrow \Pi$, the following are equivalent.*

1. $\Gamma \Rightarrow \Pi$ is provable in **LI2**.
2. $\Gamma \Rightarrow \Pi$ is valid in all Heyting structures.
3. $\Gamma \Rightarrow \Pi$ is cut-free provable in **LI2**.

Proof. (1. \Rightarrow 2.) holds by Lemma 9, and (2. \Rightarrow 3.) by $\Gamma \in \mathcal{I}(\Gamma) \subseteq \mathcal{I}(\Pi) \subseteq \Pi^\triangleleft$ in \mathcal{CF} . ◀

Recall that the frame **CF** is defined by referring to cut-free provability in **LI2**. But the above theorem states that it coincides with provability. As a consequence, we have $\gamma(\varphi) = \varphi^\triangleleft$ for every formula φ , so that there is exactly one closed set X such that $\varphi \in X \subseteq \varphi^\triangleleft$. Hence the complete algebra **CF**⁺ can be restricted to a subalgebra **CF**₀⁺ with underlying set $\{\gamma(\varphi) : \varphi \in \text{FM}\}$. It is easy to see that **CF**₀⁺ is isomorphic to the Lindenbaum algebra for **LI2** (defined analogously to **L** in Section 5) and **CF**⁺ is the MacNeille completion of **CF**₀⁺. To sum up:

► **Proposition 17.** ***CF**⁺ is the MacNeille completion of the Lindenbaum algebra for **LI2**.*

Thus it turns out *a fortiori* that the essence of Maehara and Okada's proof lies in "MacNeille completion + Girard's reducibility candidates."

6.3 Algebraic cut elimination for **LIP**_{*n*+1}

We now proceed to an algebraic proof of cut elimination for **LIP**_{*n*+1} ($n \geq -1$). Although we have already shown cut elimination for **LIP**_{*n*+1} in Section 3, the proof does not formalize in **ID**_{*n*+1} but only in **ID**_{*n*+2}. Our goal here is to give another proof that locally formalizes in **ID**_{*n*+1}. To this end, we combine the algebraic argument in the previous subsection with the Ω -interpretation technique discussed in Section 5. To be more precise, our proof is only partly algebraic, since we employ calculus **LI** Ω _{*n*} and presuppose Lemmas 6 and 7 for **LI** Ω _{*n*} (but *not* for **LI** Ω _{*n*+1} unlike before).

Define a Heyting frame by

$$\mathbf{CF}_n := \langle \wp_{\text{fin}}(\text{FMP}_{n+1} \cup \overline{\text{FMP}}_n), \text{SEQ}_n, \Rightarrow_n^{cf}, \cup, \emptyset, \setminus \rangle,$$

where SEQ_n consists of sequents $\Gamma \Rightarrow \Pi$ with $\Gamma \cup \Pi \subseteq \text{FMP}_{n+1} \cup \overline{\text{FMP}}_n$, and $\Gamma \Rightarrow_n^{cf} (\Sigma \Rightarrow \Pi)$ holds just in case **LI** Ω _{*n*} $\vdash^{cf} \Gamma, \Sigma \Rightarrow \Pi$. This yields a *full* Heyting structure $\mathcal{CF}_n := \langle \mathbf{CF}_n^+, \text{Im}, \mathcal{G}(\mathbf{CF}_n)^{\text{Im}}, \mathcal{L} \rangle$, where $p^{\mathcal{CF}_n}(\vec{t}) := \gamma(p(\vec{t}))$.

Let $\mathcal{I} : \text{VAR} \rightarrow \mathcal{G}(\mathbf{CF}_n)^{\text{Im}}$ be a valuation given by $\mathcal{I}(X)(t) := \gamma(X(t))$. The interpretation $\mathcal{I} : \text{FMP}_{n+1} \rightarrow \mathcal{G}(\mathbf{CF}_n)$ is defined as in Section 5, except that

$$\begin{aligned} \mathcal{I}(\forall X.\varphi) &:= \gamma(\{\Delta : \Delta \Rightarrow_n^{cf} \overline{\varphi}(Y) \text{ for some } Y \notin \text{FV}(\Delta)\}), \\ \mathcal{I}(\exists X.\varphi) &:= \{(\Delta \Rightarrow \Lambda) : \overline{\varphi}(Y), \Delta \Rightarrow_n^{cf} \Lambda \text{ for some } Y \notin \text{FV}(\Delta, \Lambda)\}^\triangleleft. \end{aligned}$$

This interpretation is inspired by Theorem 10. As before, it avoids use of second order quantifiers at the meta-level, that is what we have called the Ω -interpretation in Section 5. Notice the use of overlining. The main lemma nevertheless holds with respect to \mathcal{I} .

► **Lemma 18.** $\bar{\varphi} \in \mathcal{I}(\varphi) \subseteq \bar{\varphi}^\triangleleft$ for every $\varphi \in \text{FMP}_n$. $\varphi \in \mathcal{I}(\varphi) \subseteq \varphi^\triangleleft$ for every $\varphi \in \text{FMP}_{n+1}$.

The following lemma is the hardest part of the proof.

► **Lemma 19.** Suppose that $F \in \mathcal{G}(\mathbf{CF}_n)^{\text{Tm}}$ satisfies $\tau(t) \in F(t) \subseteq \tau(t)^\triangleleft$ for some $\tau(x) \in \text{FMP}_{n+1}$. Then for every $\forall X.\varphi \in \text{FMP}_{n+1}$, we have $\mathcal{I}(\forall X.\varphi) \subseteq \mathcal{I}[F/X](\varphi) \subseteq \mathcal{I}(\exists X.\varphi)$.

Once the hardest lemma has been proved, the rest is an easy soundness argument.

► **Lemma 20.** If $\text{LIP}_{n+1} \vdash \Gamma \Rightarrow \Pi$, then $\mathcal{I}(\Gamma^\circ) \subseteq \mathcal{I}(\Pi^\circ)$ holds for every substitution \circ .

Proof. We assume $\circ = \text{id}$ for simplicity. The proof proceeds by induction on the length of the derivation.

Suppose that it ends with $(\forall X \text{ left})$ with main formula $\forall X.\varphi$ and minor formula $\varphi(\tau)$. Define $F \in \mathcal{G}(\mathbf{CF}_n)^{\text{Tm}}$ by $F(t) := \mathcal{I}(\tau(t))$. By Lemma 18, this F satisfies the precondition of Lemma 19. Hence $\mathcal{I}(\forall X.\varphi) \subseteq \mathcal{I}[F/X](\varphi) = \mathcal{I}(\varphi(\tau))$, where the last equation can be shown by induction on φ . Soundness of $(\forall X \text{ left})$ follows immediately.

Suppose that the derivation ends with:

$$\frac{\Gamma \Rightarrow \varphi(Y)}{\Gamma \Rightarrow \forall X.\varphi} \text{ (\forall X right)}$$

Let $\Delta \in \mathcal{I}(\Gamma)$. We may assume that $Y \notin \text{FV}(\Delta)$, since otherwise we can rename Y to a new set variable. By the induction hypothesis and Lemma 18, we have $\Delta \in \mathcal{I}(\varphi(Y)) \subseteq \bar{\varphi}(Y)^\triangleleft$. Hence $\Delta \in \mathcal{I}(\forall X.\varphi)$. The other cases are similar. ◀

► **Lemma 21.** If $\text{LIP}_{n+1} \vdash \Gamma \Rightarrow \Pi$, then $\text{LI}\Omega_n \vdash^{cf} \Gamma \Rightarrow \Pi$.

Proof. $\Gamma \in \mathcal{I}(\Gamma) \subseteq \mathcal{I}(\Pi) \subseteq \Pi^\triangleleft$ by Lemmas 20 and 18. ◀

Combining it with Lemma 7, we obtain:

► **Theorem 22 (Cut elimination).** Suppose that $\Gamma \cup \Pi \subseteq \text{FMP}_{n+1}$. If $\Gamma \Rightarrow \Pi$ is provable in LIP_{n+1} , then it is cut-free provable in LIP_{n+1} .

As before, the algebra \mathbf{CF}_n^+ coincides with the MacNeille completion of the Lindenbaum algebra for $\text{LI}\Omega_n$. Hence our proof can be described as ‘‘MacNeille completion + Ω -interpretation’’ in contrast to Maehara and Okada’s proof.

What is the gain of an algebraic proof compared with the syntactic one in Section 4? In order to prove Lemma 21, we have only employed provability predicate $\text{LI}\Omega_n$, that is definable in ID_{n+1} . Thus we have saved one inductive definition. Furthermore, the above argument can be locally formalized in ID_{n+1} . Hence by letting $m := n + 1$ we obtain a folklore:

$$\mathbf{I}\Sigma_1 \vdash \text{CE}(\text{LIP}_m) \leftrightarrow \text{1CON}(\text{ID}_m), \quad \mathbf{I}\Sigma_1 \vdash \text{CE}(\text{LIP}) \leftrightarrow \text{1CON}(\text{ID}_{<\omega}).$$

To our knowledge, the idea of combining the Ω -rule with a semantic argument to save one inductive definition is due to Aehlig [1], where Tait’s computability predicate is used instead of the MacNeille completion. He works on the parameter-free, *negative* fragments of second order Heyting arithmetic without induction, and proves a weak form of cut elimination in the matching ID -theories. That is comparable with our result, but ours is concerned with the *full* cut elimination theorem for a logical system with the *full* set of connectives (recall that second order definitions of positive connectives are not available in the parameter-free setting).

Conclusion. In this paper we have brought the Ω -rule into the logical setting, and studied it from an algebraic perspective. We have found an intimate connection with the MacNeille completion (Theorem 10), that is important in two ways. First, it provides a link between syntactic and algebraic approaches to cut elimination. Second, it leads to an algebraic form of the Ω -rule, called the Ω -interpretation, that augments a partial observation by Altenkirch and Coquand [6]. These considerations have led to Theorem 22, the intuitionistic analogue of Takeuti's fundamental cut elimination theorem [32], proved (partly) algebraically.

We prefer the algebraic approach, since it provides a *uniform* perspective to the complicated situation in nonclassical logics. Recall that there is a limitation on MacNeille completions: it does not work for proper intermediate logics (Theorem 2). On the other hand:

- There are infinitely many substructural logics such that the corresponding varieties of algebras are closed under MacNeille completions. As a consequence, these logics, when suitably formalized as sequent calculi, admit an algebraic proof of cut elimination [15, 16].
- There are infinitely many intermediate logics for which *hyper*-MacNeille completions work. As a consequence, these logics, when suitably formalized as *hyper*-sequent calculi, admit an algebraic proof of cut elimination [15, 17].

Thus proving cut elimination amounts to finding a suitable notion of algebraic completion. Although this paper has focused on the easiest case of parameter-free intuitionistic logics, we hope that our approach will eventually lead to an algebraic understanding of *hard* results in proof theory.

References

- 1 K. Aehlig. Induction and inductive definitions in fragments of second order arithmetic. *Journal of Symbolic Logic*, 70:1087–1107, 2005.
- 2 K. Aehlig. Parameter-free polymorphic types. *Annals of Pure and Applied Logic*, 156:3–12, 2008.
- 3 R. Akiyoshi. An ordinal-free proof of the complete cut-elimination theorem for $\Pi_1^1\text{-CA} + \text{BI}$ with the ω -rule. *IfCoLog Journal of Logics and their Applications*, 4(4):867–883, 2017.
- 4 R. Akiyoshi and G. Mints. An extension of the Omega-rule. *Archive for Mathematical Logic*, 55(3):593–603, 2016.
- 5 R. Akiyoshi and K. Terui. Strong normalization for the parameter-free polymorphic lambda calculus based on the Omega-rule. *Proceedings of FSCD 2016*, 5:1–15, 2016.
- 6 T. Altenkirch and T. Coquand. A finitary subsystem of the polymorphic λ -calculus. *Proceedings of TLCA 2001*, 22–28, 2001.
- 7 T. Arai. Cut-eliminability in second order logic calculi. <https://arxiv.org/abs/1701.00929v1>, 2017.
- 8 B. Banaschewski. Hüllensysteme und Erweiterungen von Quasi-Ordnungen. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 2: 35–46, 1956.
- 9 J. Harding and G. Bezhanishvili. MacNeille completions of Heyting algebras. *The Houston Journal of Mathematics*, 30(4):937–952, 2004.
- 10 G. Birkhoff. *Lattice Theory*. AMS, 1940.
- 11 W. Buchholz. The $\Omega_{\mu+1}$ -rule. In [13], pages 188–233, 1981.
- 12 W. Buchholz. Explaining the Gentzen-Takeuti reduction steps. *Archive for Mathematical Logic*, 40:255–272, 2001.
- 13 W. Buchholz, S. Feferman, W. Pohlers and W. Sieg. *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical Studies*, LNM 897, Springer, 1981.

- 14 W. Buchholz and K. Schütte. *Proof Theory of Impredicative Subsystems of Analysis*, Bibliopolis, 1988.
- 15 A. Ciabattoni, N. Galatos and K. Terui. From axioms to analytic rules in nonclassical logics. *Proceedings of LICS 2008*, pp. 229–240, 2008.
- 16 A. Ciabattoni, N. Galatos and K. Terui. Algebraic proof theory for substructural logics: cut-elimination and completions. *Annals of Pure and Applied Logic*, 163(3):266-290, 2012.
- 17 A. Ciabattoni, N. Galatos and K. Terui. Algebraic proof theory: Hypersequents and hyper-completions. *Annals of Pure and Applied Logic*, 168(3): 693–737, 2017.
- 18 N. Funayama. On the completion by cuts of distributive lattices. *Proceedings of the Imperial Academy, Tokyo*, 20:1–2, 1944.
- 19 N. Galatos and P. Jipsen. Residuated frames with applications to decidability. *Transactions of the AMS*, 365(3):1219–1249, 2013.
- 20 M. Gehrke and J. Harding. Bounded lattice expansions. *Journal of Algebra*, 238(1):345–371, 2001.
- 21 M. Gehrke and B. Jónsson. Bounded distributive lattice expansions. *Mathematica Scandinavica*, 94(1):13–45, 2004.
- 22 G. Jäger and T. Studer. A Buchholz rule for modal fixed point logics. *Logica Universalis* 5(1):1–19, 2011.
- 23 B. Jónsson and A. Tarski. Boolean algebras with operators I. *American Journal of Mathematics*, 73: 891–939, 1951.
- 24 S. Maehara. Lattice-valued representation of the cut-elimination theorem. *Tsukuba Journal of Mathematics*, 15(9):509–521, 1991.
- 25 G. Mints and T. Studer. Cut-elimination for the mu-calculus with one variable. *Fixed Points in Computer Science*, 77: 47–54, 2012.
- 26 M. Okada. Phase semantics for higher order completeness, cut-elimination and normalization proofs (extended abstract). *Electric Notes in Theoretical Computer Science*, 3: 154, 1996.
- 27 M. Okada. Phase semantic cut-elimination and normalization proofs of first- and higher-order linear logic. *Theoretical Computer Science*, 227:333–396, 1999.
- 28 M. Okada. A uniform semantic proof for cut-elimination and completeness of various first and higher order logics. *Theoretical Computer Science*, 281(1-2): 471–498, 2002.
- 29 J. Schmidt. Zur Kennzeichnung der Dedekind-MacNeilleschen Hülle einer geordneten Menge. *Archiv der Mathematik*, 7:241–249, 1956.
- 30 W. Tait. A nonconstructive proof of Gentzen’s Hauptsatz for second order predicate logic. *Bulletin of American Mathematical Society*, 72:980–983, 1966.
- 31 G. Takeuti. On the generalized logic calculus. *Japanese Journal of Mathematics*, 23:39–96, 1953.
- 32 G. Takeuti. On the fundamental conjecture of GLC V. *Journal of the Mathematical Society of Japan*, 10(2):121–134, 1958.
- 33 M. Theunissen and Y. Venema. MacNeille completions of lattice expansions. *Algebra Universalis*, 57:143–193, 2007.

A

 Definitions of sequent calculi

A.1 Sequent calculi LI2, LIP and LIP_n

Sequents of **LI2** consist of formulas in FM. Inference rules are as follows:

$$\begin{array}{c}
 \frac{}{\Gamma, \varphi \Rightarrow \varphi} \text{ (id)} \qquad \frac{\Gamma \Rightarrow \varphi \quad \varphi, \Gamma \Rightarrow \Pi}{\Gamma \Rightarrow \Pi} \text{ (cut)} \\
 \\
 \frac{}{\perp, \Gamma \Rightarrow \Pi} (\perp \text{ left}) \qquad \frac{\Gamma \Rightarrow}{\Gamma \Rightarrow \perp} (\perp \text{ right}) \\
 \\
 \frac{\varphi_i, \Gamma \Rightarrow \Pi}{\varphi_1 \wedge \varphi_2, \Gamma \Rightarrow \Pi} (\wedge \text{ left}) \qquad \frac{\Gamma \Rightarrow \varphi_1 \quad \Gamma \Rightarrow \varphi_2}{\Gamma \Rightarrow \varphi_1 \wedge \varphi_2} (\wedge \text{ right}) \\
 \\
 \frac{\varphi_1, \Gamma \Rightarrow \Pi \quad \varphi_2, \Gamma \Rightarrow \Pi}{\varphi_1 \vee \varphi_2, \Gamma \Rightarrow \Pi} (\vee \text{ left}) \qquad \frac{\Gamma \Rightarrow \varphi_i}{\Gamma \Rightarrow \varphi_1 \vee \varphi_2} (\vee \text{ right}) \\
 \\
 \frac{\Gamma \Rightarrow \varphi_1 \quad \varphi_2, \Gamma \Rightarrow \Pi}{\varphi_1 \rightarrow \varphi_2, \Gamma \Rightarrow \Pi} (\rightarrow \text{ left}) \qquad \frac{\varphi_1, \Gamma \Rightarrow \varphi_2}{\Gamma \Rightarrow \varphi_1 \rightarrow \varphi_2} (\rightarrow \text{ right}) \\
 \\
 \\
 \\
 \frac{\varphi(t), \Gamma \Rightarrow \Pi}{\forall x. \varphi(x), \Gamma \Rightarrow \Pi} (\forall x \text{ left}) \qquad \frac{\Gamma \Rightarrow \varphi(y) \quad y \notin \text{Fv}(\Gamma)}{\Gamma \Rightarrow \forall x. \varphi(x)} (\forall x \text{ right}) \\
 \\
 \frac{\varphi(y), \Gamma \Rightarrow \Pi \quad y \notin \text{Fv}(\Gamma, \Pi)}{\exists x. \varphi(x), \Gamma \Rightarrow \Pi} (\exists x \text{ left}) \qquad \frac{\Gamma \Rightarrow \varphi(t)}{\Gamma \Rightarrow \exists x. \varphi(x)} (\exists x \text{ right}) \\
 \\
 \frac{\varphi(\tau), \Gamma \Rightarrow \Pi}{\forall X. \varphi(X), \Gamma \Rightarrow \Pi} (\forall X \text{ left}) \qquad \frac{\Gamma \Rightarrow \varphi(Y) \quad Y \notin \text{FV}(\Gamma)}{\Gamma \Rightarrow \forall X. \varphi(X)} (\forall X \text{ right}) \\
 \\
 \frac{\varphi(Y), \Gamma \Rightarrow \Pi \quad Y \notin \text{FV}(\Gamma, \Pi)}{\exists X. \varphi(X), \Gamma \Rightarrow \Pi} (\exists X \text{ left}) \qquad \frac{\Gamma \Rightarrow \varphi(\tau)}{\Gamma \Rightarrow \exists X. \varphi(X)} (\exists X \text{ right})
 \end{array}$$

LIP (resp. **LIP_n** with $n \geq -1$) is obtained by restricting the formulas to FMP (resp. FMP_n).

A.2 Sequent calculi LIΩ_n

LIΩ₋₁ is just **LIP** where cut formulas are restricted to Fm.

For $n \geq 0$, sequents of **LIΩ_n** consist of formulas in $\text{FMP} \cup \overline{\text{FMP}}_n$. Inference rules are (id), (cut), those for propositional connectives and the following rules (where ϑ stands for a

formula in $\overline{\text{FMP}}_{n-1}$):

$$\begin{array}{l}
\frac{\varphi(t), \Gamma \Rightarrow \Pi}{\forall x.\varphi(x), \Gamma \Rightarrow \Pi} \text{ (\forall x left)} \qquad \frac{\{ \Gamma \Rightarrow \varphi(t) \}_{t \in \text{TM}}}{\Gamma \Rightarrow \forall x.\varphi(x)} \text{ (\omega right)} \\
\frac{\{ \varphi(t), \Gamma \Rightarrow \Pi \}_{t \in \text{TM}}}{\exists x.\varphi(x), \Gamma \Rightarrow \Pi} \text{ (\omega left)} \qquad \frac{\Gamma \Rightarrow \varphi(t)}{\Gamma \Rightarrow \exists x.\varphi(x)} \text{ (\exists x right)} \\
\frac{\varphi(\tau), \Gamma \Rightarrow \Pi}{\forall X.\varphi(X), \Gamma \Rightarrow \Pi} \text{ (\forall X left)} \qquad \frac{\Gamma \Rightarrow \varphi(Y) \quad Y \notin \text{FV}(\Gamma)}{\Gamma \Rightarrow \forall X.\varphi(X)} \text{ (\forall X right)} \\
\frac{\varphi(Y), \Gamma \Rightarrow \Pi \quad Y \notin \text{FV}(\Gamma, \Pi)}{\exists X.\varphi(X), \Gamma \Rightarrow \Pi} \text{ (\exists X left)} \qquad \frac{\Gamma \Rightarrow \varphi(\tau)}{\Gamma \Rightarrow \exists X.\varphi(X)} \text{ (\exists X right)} \\
\frac{\vartheta(Y), \Gamma \Rightarrow \Pi \quad Y \notin \text{FV}(\Gamma, \Pi)}{\exists X.\vartheta(X), \Gamma \Rightarrow \Pi} \text{ (\exists X left)} \qquad \frac{\Gamma \Rightarrow \vartheta(Y) \quad Y \notin \text{FV}(\Gamma)}{\Gamma \Rightarrow \forall X.\vartheta(X)} \text{ (\forall X right)} \\
\frac{\{ \Delta, \Gamma \Rightarrow \Pi \}_{\Delta \in |\overline{\forall X.\vartheta}|}}{\overline{\forall X.\vartheta}, \Gamma \Rightarrow \Pi} \text{ (\tilde{\Omega}_k left)} \qquad \frac{\Gamma \Rightarrow \vartheta(Y) \quad \{ \Delta, \Gamma \Rightarrow \Pi \}_{\Delta \in |\overline{\forall X.\vartheta}|}}{\Gamma \Rightarrow \Pi} \text{ (\tilde{\Omega}_k left)} \\
\frac{\{ \Gamma, \Delta \Rightarrow \Lambda \}_{(\Delta \Rightarrow \Lambda) \in |\exists X.\vartheta|}}{\Gamma \Rightarrow \exists X.\vartheta} \text{ (\Omega_k right)} \qquad \frac{\{ \Gamma, \Delta \Rightarrow \Lambda \}_{(\Delta \Rightarrow \Lambda) \in |\exists X.\vartheta|} \quad \vartheta(Y), \Gamma \Rightarrow \Pi}{\Gamma \Rightarrow \Pi} \text{ (\tilde{\Omega}_k right)}
\end{array}$$

where $k = 0, \dots, n$, which is determined by the level of the main formula $\overline{Q}X.\vartheta$. Rules $(\tilde{\Omega}_k \text{ left})$ and $(\tilde{\Omega}_k \text{ right})$ are subject to the eigenvariable condition ($Y \notin \text{FV}(\Gamma, \Pi)$). Index sets are defined by:

$$\begin{aligned}
|\overline{\forall X.\vartheta(X)}| &:= \{ \Delta : \mathbf{LI}\Omega_{k-1} \vdash^{cf} \Delta \Rightarrow \vartheta(Y) \text{ for some } Y \notin \text{FV}(\Delta) \} \\
|\exists X.\vartheta(X)| &:= \{ (\Delta \Rightarrow \Lambda) : \mathbf{LI}\Omega_{k-1} \vdash^{cf} \vartheta(Y), \Delta \Rightarrow \Lambda \text{ for some } Y \notin \text{FV}(\Delta, \Lambda) \}.
\end{aligned}$$

