31st EACSL Annual Conference on Computer Science Logic

CSL 2023, February 13–16, 2023, Warsaw, Poland

Edited by

Bartek Klin

Elaine Pimentel

LIPIcs – Vol. 252 – CSL 2023 www.dagstuhl.de/lipics

Editors

Bartek Klin
University of Oxford, UK
bartek.klin@cs.ox.ac.uk

Elaine Pimentel
University College London, UK
e.pimentel@ucl.ac.uk

ACM Classification 2012
Theory of computation → Logic

ISBN 978-3-95977-264-8

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-264-8.

Publication date
February, 2023

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

Digital Object Identifier: 10.4230/LIPIcs.CSL.2023.0

ISBN 978-3-95977-264-8 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

31st EACSL Annual Conference on Computer Science Logic (CSL 2023).
Editors: Bartek Klin and Elaine Pimentel
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the papers presented at CSL 2023, the 31st meeting in the conference series Computer Science Logic (CSL), the annual conference of the European Association for Computer Science Logic (EACSL). CSL 2023 was held from 13th to 16th February 2023. It was organised at the University of Warsaw.

CSL started as a series of international workshops, and became an international conference in 1992. Previous instalments of CSL were held in Göttingen (2022, on-line), Ljubljana (2021, on-line), Barcelona (2020), Birmingham (2018), Stockholm (2017), Marseille (2016), Berlin (2015), Vienna (2014), Torino (2013), Fontainebleau (2012), Bergen(2011), Brno (2010), Coimbra (2009), Bologna (2008), Lausanne (2007), Szeged (2006), Oxford (2005), Karpacz (2004), Vienna (2003), Edinburgh (2002), Paris (2001), Munich (2000), Madrid (1999), Brno (1998), Aarhus (1997), Utrecht (1996), Paderborn (1995), Kazimierz (1994), Swansea (1993) and San Miniato (1992).

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

The conference received 93 abstracts of which 75 were followed up by full-paper submissions. The programme committee selected 34 papers for presentation at the conference. Each paper was reviewed by at least three members of the programme committee, with the help of external reviewers. The submission and reviewing process, programme committee discussion, and author notifications were all handled by the Easychair conference management system. In addition to the contributed papers, there were five invited talks, by: Claudia Faggian (Université Paris Cité, France), Nina Gierasimczuk (Danish Technical University, Denmark), Dale Miller (Inria Saclay, France), Michał Pilipczuk (University of Warsaw, Poland) and Davide Sangiorgi (University of Bologna, Italy). We thank the invited speakers for their stimulating talks and papers, which greatly contributed to the success of the conference. One of the major regular events at CSL conferences is the presentation of the Ackermann Award: the annual EACSL award for an outstanding dissertation in the area of logic in computer science. The recipients of the award are selected by jury from a field of international nominees, and the recipients receive their award at a ceremony at which they give a prize lecture on their dissertation. This year, the jury elected to give the Ackermann Award 2022 to Alexander Bentkamp for his thesis "Superposition for Higher-Order Logic" defended at Vrije Universiteit Amsterdam (The Netherlands) under the supervision of Jasmin Blanchette, Uwe Waldmann, and Wan Fokkink. The award was presented during the conference. The citation for the award is included in the proceedings.

A significant event at CSL 2023 was the presentation of the Helena Rasiowa Award, named after the eminent Polish mathematician and logician Helena Rasiowa (1917 – 1994) whose work had an essential impact on the emerging field of logic in computer science. The

31st EACSL Annual Conference on Computer Science Logic (CSL 2023).
Editors: Bartek Klin and Elaine Pimentel
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Helena Rasiowa Award, presented for the first time at CSL 2022, is given to the best paper, as decided by the programme committee, that is written solely by students or to which students were the main contributors. There was a strong field of candidates for this award edition, with 10 of the accepted papers eligible. From these, the programme committee selected Hiromi Tanaka as the recipient of the 2023 Helena Rasiowa Award, for his paper "Tower-Complete Problems in Contraction-Free Substructural Logics". Hiromi Tanaka is a PhD student at the Keio University under the supervision of Tatsuya Kashiwabata.

CSL 2023 also had three affiliated workshops: Fixpoints in Computer Science (FICS), Logic Mentoring Workshop (LMW), and Schwentickfest.

We are very grateful to all the members of the CSL 2023 programme committee and external reviewers for their careful and efficient evaluation of the papers submitted. We would like to thank also the members of the organisation committee Lorenzo Clemente, Wojciech Czerwiński, Radosław Piórkowski, from the University of Warsaw, for taking care to ensure a smooth-running and enjoyable conference. It was as always a pleasure to work with Thomas Schwentick/Maribel Fernandez who, as the EACSL presidents until 2022/from 2023, provided excellent guidance. The proceedings of CSL 2023 are published as a volume in the LIPIcs series. We thank Michael Wagner, Michael Didas and all the Dagstuhl/LIPIcs team for their ongoing support and for the high quality preparation of these proceedings. Last, but not least, we are very grateful to the University of Warsaw for supporting the organisation of this conference.

Bartek Klin and Elaine Pimentel 25th November 2022

Program Committee Members

Matteo Acclavio (University of Luxembourg, Luxembourg)
Patricia Bouyer-Decitre (LSV, CNRS & ENS Paris-Saclay, France)
Agata Ciabattoni (TU Wien, Austria)
Diana Costa (University of Lisbon, Portugal)
Alejandro Díaz-Caro (Univ. N. de Quilmes & CONICET-Univ. de BA, Argentina)
Martín Escardó (University of Birmingham, UK)
Rajeev Goré (The Australian National University, Australia)
Giulio Guerrieri (Huawei Edinburgh Research Centre, UK)
Shin-ya Katsumata (National Institute of Informatics, Japan)
Delia Kesner (Université Paris Cité, France)
Sandra Kiefer (Max Planck Institute for Software Systems, Germany)
Bartek Klin (University of Oxford, UK, co-chair)
Naoki Kobayashi (The University of Tokyo, Japan)
Stepan Kuznetsov (Steklov Mathematical Institute of RAS, Russia)
Martin Lück (Leibniz Universität Hannover, Germany)
Meena Mahajan (The Institute of Mathematical Sciences, HBNI, India)
Filip Murlak (University of Warsaw, Poland)
Daniele Nantes (University of Brasília, Brazil)
Elaine Pimentel (UCL, UK, co-chair)
Paolo Pistone (University of Bologna, Italy)
Ana Sokolova (University of Salzburg, Austria)
Lutz Straßburger (Inria Saclay – Île-de-France, France)
Pascal Schweitzer (TU Darmstadt, Germany)
Martin Zimmermann (Aalborg University, Denmark)
Yoni Zohar (Bar Ilan University, Israel)

31st EACSL Annual Conference on Computer Science Logic (CSL 2023).
Editors: Bartek Klin and Elaine Pimentel
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Beniamino Accattoli
Shaull Almagor
Pablo Barenbaum
Victoria Barrett
Nicolas Behr
Ulrich Berger
Steffen van Bergerem
Yves Bertot
Achim Blumensath
James Brotherston
Florian Bruse
Andrei Bulatov
Davide Catta
Ranald Clouston
Bob Coecke
Liron Cohen
Fredrik Dahlqvist
Anupam Das
Laure Daviaud
Anuj Dawar
Bérénice Delcroix-Oger
Antonin Delpeuch
Arnaud Durand
Cristina Feier
Giulio Fellin
Christian Fermüller
Nathanaël Fijalkow
Robert Freiman
Moses Ganardi
Herman Geuvers
Guido Gherardi
Dan Ghica
Iris van der Giessen
Alessandro Di Giorgio
Erich Grädel
Jim de Groot
Victor Gutierrez-Basulto
Albert Gutowski
Anselm Haak
Amar Hadzihasanovic
Tuomas Hakoniemi
Miika Hannula
Valentina Harizanov
Masahito Hasegawa
Lauri Hella
Jelle Hellings
Loic Helouet
Federico Holik
Ahmed Irfan
Jules Jacobs
Ismaël Jecker
Christian Johansen
Matthieu Josuat-Vergès
Jean Christoph Jung
Jens Keppeler
Marie Kerjean
Stefan Kiefer
Emanuel Kieronski
Yuichi Komorida
Cynthia Kop
Eryk Kopczynski
Andre Kornell
Denis Kuperberg
Timo Lang
Graham Leigh
Moritz Lichter
Etienne Lozes
Tim Lyon
Ian Mackie
Yasir Mahmood
Octavio Malherbe
Makai Mann
Nicolas Markey
Dan Marsden
Guillaume Massas
Filip Mazowiecki
Damiano Mazza
Alexandre Miquel
Marianela Morales Elena
Sean Moss
Carlos Olarte
Paulo Oliva
Federico Olmedo
Benedikt Pago
Pierre-Marie Pédrot
Robin Piedeleu
Diogo Poças
Femke van Raamsdonk
Luca Reggio
Fabian Reiter
Mark Reynolds
Daniel Rogozin
Claudio Sacerdoti Coen
Ken Sakayori
Alessio Santamaria
Yury Savateev
Sylvain Schmitz
Monika Seisenberger
Ying Sheng
A V Sreejith
B Srivathsan
Isar Stubbe
Martin Sulzmann
S P Suresh
Grégoire Sutre
Eugenia Ternovska
Lê Thành Dũng Nguyễn
Neil Thapen
Riccardo Treglia
Pierre Vandenhove
Gabriele Vanoni
Daniele Varacca
Lionel Vaux Auclair
Daniel Ventura
Marcos Villagra
Dominik Wehr
Thorsten Wißmann
Chuangjie Xu
Marc Zeitoun

31st EACSL Annual Conference on Computer Science Logic (CSL 2023).
Editors: Bartek Klin and Elaine Pimentel
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Ackermann Award 2022

By Jean Goubault-Larrecq and Thomas Schwentick
For the Jury of the EACSL Ackermann Award

The 18th Ackermann Award was presented at CSL'23 in Warsaw, Poland. The 2022 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 fulfilment of the PhD degree between 1 January 2020 and 31 December 2021. The Jury received eleven nominations for the 2022 Award. The candidates came from a number of different countries around the world. The institutions at which the nominees obtained their doctorates represent different countries in Europe, Asia and North America.

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

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

The wide range of excellent candidates presented the jury with an excruciating task. After an extensive discussion, the jury decided to award the **2022 Ackermann Award** to:

- Alexander Bentkamp from Germany for his thesis

  Superposition for Higher-Order Logic

  approved by *Vrije Universiteit Amsterdam* in 2021.

Citation

Alexander Bentkamp receives the *2022 Ackermann Award* of the European Association of Computer Science Logic (EACSL) for his thesis

Superposition for Higher-Order Logic.

The thesis is in the domain of automated theorem proving. The mechanization of proofs is of growing importance in several areas of mathematics and computer science, and is already of paramount importance in ascertaining the correctness of critical software and hardware.

Mechanizing proofs has different meanings. In proof assistants, human guidance is required. With automated theorem provers, the computer finds proofs automatically. Already with first-order logic, this is an undecidable task, although one for which efficient tools exist, notably those based on so-called superposition procedures. It had been widely believed since the 1990s that superposition could not be extended much beyond first-order logic, and that higher-order logic would remain the realm of proof assistants for a long time.

In a series of breakthroughs, A. Bentkamp manages to extend superposition calculi to higher-order logic. He does this in three steps: by first extending superposition to the lambda-free fragment, then to the more expressive clausal fragment, and finally to full

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

31st EACSL Annual Conference on Computer Science Logic (CSL 2023).
Editors: Bartek Klin and Elaine Pimentel
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

higher-order logic. In each case, he obtains sound and complete calculi: all higher-order theorems can be proved by his calculi, and only them. He also demonstrates that these calculi are several orders of magnitude more efficient than previous proposals, through his award-winning, Zipperposition-based implementations.

Background to the thesis

Automated theorem proving is one of the oldest fields of computer science logic. Herbrand and others laid out the bases of first-order theorem proving as early as 1930, the first automated first-order provers were created in the 1960s, and the highly efficient superposition calculi were devised in the 1990s by Bachmair and Ganzinger. Andrews and Huet were among the pioneers of automated higher-order logic theorem proving in the 1970s and 1980s. Before A. Bentkamp, the most common higher-order theorem proving procedures relied on translations to first-order logic, as first hinted by Robinson in 1970, and then following Kerber, Dougherty, and others, starting from the 1990s. However, the sophisticated recipes that modern first-order provers rely for efficient theorem proving, notably the use of term orderings, are essentially nullified by the various translations.

Extending superposition to higher-order calculi presented formidable challenges already for the so-called lambda-free fragment of higher-order logic, which is essentially first-order logic augmented with the possibility of applying variables to terms inside terms. For one, one may need to superpose inside variables, or even below, which seems untenable. One also requires term orderings that are monotonic and ground-total, as needed in standard approaches to superposition, and that runs in conflict with the need for variable applications. In more general fragments of first-order logic, additional difficulties accumulate, and notably the need to guess the shapes of formulae that should instantiate Boolean variables.

Contributions of the thesis

A. Bentkamp first considers the lambda-free fragment, and, in a first breakthrough, shows how to extend superposition with term orderings that may fail to be monotonic. He obtains calculi that are complete, even in the presence of redundancy criteria. The rules are familiar from superposition, but also surprising in some ways: some of them offer no guidance in some cases, but these cases appear to much rarer than what one would expect in the empirical evaluations. The key concept is that of green contexts and green terms: superposition only occurs at green subterms, and the term orderings have to be compatible with green contexts (not all contexts) on ground terms. The completeness proofs also have to be adapted to this new, more permissive setting. This is especially challenging in the presence of redundancy criteria.

In an ironic twist, A. Bentkamp then shows an alternative route to a complete superposition calculus for lambda-free higher-logic, by designing a ground-total simplification ordering for untyped lambda-tree terms.

Next, A. Bentkamp explores a more expressive fragment of higher-order logic, the clausal fragment. Here the terms may involve lambda-abstractions as well, which incurs additional difficulties. Crucially, A. Bentkamp shows that one can retain completeness by restricting superposition inferences to unapplied subterms occurring in the first-order outer skeleton of clauses. Surprisingly, he decides to use full higher-order unification, even in the case of so-called flex-flex pairs. It had been widely believed since Huet that one should never try to solve flex-flex pairs, which incur a form of blind search; but solving them appears to be more efficient in the end. This requires a form of unification that regularly pauses and gives back control to the proof search engine.

Finally, A. Bentkamp goes to full higher-order logic, complete with polymorphism, extensionality and the axiom of choice in the form of Hilbert's epsilon symbol. Here, one cannot convert to clausal form statically. Instead, clausification is also performed during proof search. On top of that, new rules implement Boolean rewriting, and a clever, so-called $Q_{\approx}$-normalization procedure is introduced to deal with quantifiers so as to preserve completeness while retaining efficiency. The completeness proof proceeds by a generalization of the techniques used in previous chapters, which involve several intermediate logics and calculi.

It seems clear that the thesis will have a lasting impact in the field. Beyond the level of mastery and of novelty that the thesis displays, and also the number of bold and sometimes surprising decisions that were made, and proved successful in the end, one should stress that this thesis, despite a high degree of technical sophistication, is a pleasure to read.

Biographical sketch

Alexander Bentkamp carried out his PhD at the Vrije Universiteit Amsterdam under the supervision of Jasmin Blanchette, Uwe Waldmann, and Wan Fokkink. He won a best junior researcher paper award at FSCD 2020, a best student paper award at CADE 2021, and the 1st place in the higher-order category of the CADE ATP System Competition (CASC) in 2020, 2021 and 2022. Furthermore, he won the Bill McCune PhD Award for distinguished PhD theses in Automated Reasoning, and the E.W. Beth Outstanding Dissertation Prize for outstanding PhD dissertations in Logic, Language, and Information. He is currently a postdoctoral researcher at the mathematical institute of the Heinrich-Heine-Universität Düsseldorf.

Jury

The jury for the **Ackermann Award 2022** consisted of ten 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);
- Maribel Fernandez (King's College London);
- Delia Kesner (IRIF, U Paris Cité);
- Slawomir Lasota (U Warsaw);
- Jean Goubault-Larrecq (ENS Paris-Saclay);
- Prakash Panangaden (McGill University);
- Simona Ronchi Della Rocca (University of Torino), the vice-president of EACSL;
- Thomas Schwentick (TU Dortmund), the president of EACSL;
- Alexandra Silva (Cornell University), ACM SigLog representative;
- James Worrell (U Oxford).

Previous winners

Previous winners of the Ackermann Award were

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

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

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

2008, Bertinoro:
Krishnendu Chatterjee from India.

2009, Coimbra:
Jakob Nordström from Sweden.

2010, Brno:
no award given.

2011, Bergen:
Benjamin Rossman from USA.

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

2013, Turin:
Matteo Mio from Italy.

2014, Vienna:
Michael Elberfeld from Germany.

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

2016, Marseille:
Nicolai Kraus from Germany

2017, Stockholm:
Amaury Pouly from France.

2018, Birmingham:
Amina Doumane from France.

2019, Barcelona (conference in 2020):
Antoine Mottet from France.

2020, Ljubljana (conference online in 2021)
Benjamin Kaminski from Germany.

2021, Göttingen (conference online in 2022)
Marie Fortin from France, and
Sandra Kiefer from Germany.

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

Asymptotic Rewriting

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

Abstract

Rewriting is a foundation for the operational theory of programming languages. The process of rewriting describes the *computation of a result* (typically, a normal form), with lambda-calculus being the paradigmatic example for rewriting as an abstract form of program execution. Taking this view, the execution of a program is formalized as a specific *evaluation strategy*, while the general rewriting theory allows for *program transformations, optimizations, parallel/distributed implementations*, and provides a base on which to reason about program equivalence.

In this talk, we discuss what happens when the notion of termination is *asymptotic*, that is, the result of computation appears as a *limit*, as opposed to reaching a normal form in a *finite* number of steps. A natural example is probabilistic computation.

▶ **Example 1.** A probabilistic program P is a stochastic model generating a distribution over all possible outputs of P. Even if the termination probability is 1 (*almost sure termination*), that degree of certitude is typically not reached in a finite number of steps, but *as a limit*. A standard example is a term M that reduces to either a normal form or M itself, with equal probability $1/2$. After n steps, M is in normal form with probability $\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^n}$. *Only at the limit* this computation terminates with probability 1.

▶ **Example 2.** Infinitary lambda-calculi (where the limits are infinitary terms such as Böhm trees), streams, algebraic rewriting systems, effectful computation (e.g. computation with outputs), quantum lambda-calculi provide several other relevant examples.

Instances of asymptotic computation are quite diverse, and moreover the specific syntax of each system may be rather complex. In the talk, we present asymptotic rewriting in a way which is independent of the specific details of each calculus, and we provide a toolkit of proof-techniques which are of general application. To do so, we rely on Quantitative Abstract Rewriting System [3, 4], building on work by Ariola and Blom [1], which enrich with quantitative information the theory of Abstract Rewriting Systems (ARS) (see e.g. [5] or [2]). ARS are indeed the core of finitary rewriting, capturing the common substratum of rewriting theory and term transformation, independently from the particular structure of the objects. It seems then natural to seek a similar foundation for asymptotic computation. The issue is that the arguments relying on finitary termination do not transfer, in general, to limits (a game changer being that asymptotic termination does not provide a well-founded order): we need to develope an opportune formalization and suitable proof techniques.

The goal is then to identify and develop methods which only rely on the asymptotic argument – abstracting from structure specific to a setting – and so will apply to any concrete instance. For example, in infinitary lambda calculus, the limit is usually a (possibly infinite) limit term, while in probabilistic lambda calculus, the limit is a distribution over (finite) terms. The former is concerned with the depth of the redexes, the latter with the probability of reaching a result. The abstract notions of limit and of normalization subsumes both, and so abstract results apply to either setting.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Equational logic and rewriting; Theory of computation → Lambda calculus

Keywords and phrases rewriting, probabilistic rewriting, confluence, strategies, asymptotic normalization, lambda calculus

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.1

Category Invited Talk

References

1 Zena M. Ariola and Stefan Blom. Skew confluence and the lambda calculus with letrec. *Annals of Pure and Applied Logic*, 117(1):95–168, 2002. doi:10.1016/S0168-0072(01)00104-X.

2 Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998. doi:10.1017/CBO9781139172752.

3 Claudia Faggian. Probabilistic rewriting and asymptotic behaviour: on termination and unique normal forms. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/lmcs-18(2:5)2022.

4 Claudia Faggian and Giulio Guerrieri. Strategies for asymptotic normalization. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022*, volume 228 of *LIPIcs*, pages 17:1–17:24. Schloss Dagstuhl, 2022. doi:10.4230/LIPIcs.FSCD.2022.17.

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

Inductive Inference and Epistemic Modal Logic

Nina Gierasimczuk
Department of Applied Mathematics and Computer Science,
Technical University of Denmark, Lyngby, Denmark

Abstract

This paper is concerned with a link between inductive inference and dynamic epistemic logic. The bridge was first introduced in [20, 21, 22]. We present a synthetic view on subsequent contributions: inductive truth-tracking properties of belief revision policies seen as belief upgrade methods; topological interpretation and characterisation of inductive inference; discussion of the adequacy of the topological semantics of modal logic for characterising inductive inference. We briefly present the topological Dynamic Logic for Learning Theory. Finally, we discuss several surprising results obtained in computational inductive inference that challenge the usual understanding of certainty, and of rational inquiry as consistent and conservative learning.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Models of learning

Keywords and phrases modal logic, dynamic epistemic logic, inductive inference, topological semantics, computational learning theory, finite identifiability, identifiability in the limit

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.2

Category Invited Talk

1 Introduction

Modal logic is applied in a variety of domains. One particular direction, *epistemic logic*, interprets modalities as coding information states of an (artificial) agent. The dynamic extensions of epistemic logic, referred to with the umbrella term of Dynamic Epistemic Logic (DEL, see [7, 19, 8]), allow analysing the events of incorporating new information into the existing knowledge, for instance through communication. Such adoption of new information can be of course called "learning", but inductive inference and computational learning theory have stronger demands: the process of learning is considered successful if over time it converges to a state of familiarity with the true underlying structure of the world. For instance we would say "Alice learned Polish", if Alice managed to acquire sufficient familiarity with the grammar of the Polish language. Theories of inductive inference are concerned with this long-term horizon of learning, rather than with learning understood as ways of incorporating single pieces of information. In this survey paper we will discuss a connection between inductive inference and modal logic, by way of DEL. We hope to show that modal logic can contribute to our understanding of learning processes in general.

The link between dynamic epistemic logic and computational learning theory was first introduced in [20, 21, 22], where it was shown that exact learning in finite time (also known as finite identification, see [27, 29]) can be modelled in DEL, and that the elimination process of learning by erasing [26] can be seen as iterated upgrade of Dynamic Doxastic Logic. In the present paper we will outline the continuations developed in a number of contributions. In Section 3 we will discuss inductive truth-tracking properties of belief revision policies seen as belief upgrade methods [4, 6]. Section 4 will display a topological interpretation of inductive inference that allows characterising favourable conditions for learning in the limit in terms of general topology [5]. As modal logic can be given topological semantics, in Section 4.1 we will discuss a variety of modal logic characterisations of relevant kinds of topological spaces, to finally present Dynamic Logic for Learning Theory, which extends

Subset Space Logics [16] with dynamic observation modalities and a learning operator [3]. Up until this point our set up abstracts away from computational constraints. In Section 5 we will discuss several surprising results obtained in computational inductive inference that challenge our understanding of knowledge as certainty, and of rational inquiry as requiring conjectures to be consistent (always accounting for all truthful information obtained so far) and conservative (never changing unless encountering information that is inconsistent with the current conjecture). In the final part of the paper we will mention some related directions of research.

1.1 Inductive Inference

Inductive inference is concerned with identifying a theory that underlies its partial presentation. Examples include isolating the right general scientific hypothesis on the basis of a limited number of empirical experiments, or finding a correct grammar that had produced finitely many observed sentences. This problem is difficult because each finite sample might be consistent with many different hypotheses of infinite generative capacities. *Computational* inductive inference poses effectivity constraints: hypotheses and learners should be computable objects. The field of *formal* learning theory (for an overview, see [33]) often abstracts away from the assumptions of computability. This means retaining some countability and enumerability assumptions, but giving up learner's recursivity. As a consequence, within this more general setting, only the purely "structural" results of computational learning theory can be proven (for a formal account of this type of distinctions, see [15]). The first part of this paper, until Section 5, will focus on the structural aspects of learnable concepts, and the resulting notions of knowledge and belief. After that we will discuss some interesting consequences of computability.

We will start by discussing the basic setting of inductive inference and present some examples. Let $\mathbb{N}$ be the set of all natural numbers (including 0), we call any $S \subseteq \mathbb{N}$ a language. In the general case, we will be interested in any countable class of languages together with its hypothesis space (a naming system), i.e., in the class $\mathcal{C} = (S_i)_{i \in I}$, the indices i will serve as names used by the learner to refer to a particular (possibly infinite) language. I is a countable index set, in most cases we will consider it to be $\mathbb{N}$. We will also sometimes take $\mathcal{C}$ to be a finite set of (finite) languages.

▶ **Definition 1.** *A* text (positive presentation) τ *of* S *is any infinite sequence enumerating all and only the elements from* S *(allowing repetitions). We will use the following notation:* τ_n *is the* n*-th element of* τ*;* $\tau{\restriction}n$ *is the sequence* $(\tau_0, \tau_2, \ldots, \tau_{n-1})$*;* $\mathrm{set}(\tau)$ *is the set of elements that occur in* τ*; and if* τ *and* σ *be two sequences, then* $\tau^\frown\sigma$ *stands for their concatenation.*

▶ **Definition 2.** *A learner* L *is a function* $L : \mathbb{N}^* \to \mathbb{N} \cup \{\uparrow\}$*. The function is allowed to refrain from giving a natural number answer, in that case the output is marked by* $\uparrow$*. A learner* L *is* (at most) once defined *on* $\mathcal{C}{=}(S_i)_{i\in\mathbb{N}}$ *iff for any text* τ *for a language in* $\mathcal{C}$ *and* $n, k \in \mathbb{N}$ *such that* $n{\neq}k$ *we have* $L(\tau{\restriction}n){=}\uparrow$ *or* $L(\tau{\restriction}k){=}\uparrow$*.*

▶ **Definition 3** (Finite identifiability [24])**.** *Let* $\mathcal{C} = (S_i)_{i\in\mathbb{N}}$*, and let* L *be a learner.*

1. L *finitely identifies* S_i *in* $\mathcal{C}$ *on* τ *iff* L *is once defined on* τ *and the defined value is* i*.*
2. L *finitely identifies* S_i *in* $\mathcal{C}$ *iff it finitely identifies* S_i *on every* τ *for* S_i*;*
3. L *finitely identifies* $\mathcal{C}$ *iff it finitely identifies every* S_i *in* $\mathcal{C}$*.*

A class $\mathcal{C}$ *is finitely identifiable iff there is a learner* L *that finitely identifies* $\mathcal{C}$*.*

▶ **Example 4.** We will now discuss two classes that are finitely identifiable, and one (very simple) class that is not. First consider the class containing all pairs of natural numbers, $\mathcal{C}_{pair} = \{\{n, k\} \mid n, k \in \mathbb{N}\}$. To see that $\mathcal{C}_{pair}$ is finitely identifiable, consider the hypothesis space which uses a diagonal indexing of $\mathcal{C}_{pair}$. Let's take a learner that on the input of initial segments of τ outputs $\uparrow$ as long as it sees only one number begin repeated, upon seeing the second number the learner outputs the corresponding index and, after that, switches back to the $\uparrow$ answer forever. The learner finitely identifies every element of $\mathcal{C}_{pair}$ because every pair is a subset of only one element of $\mathcal{C}_{pair}$.

Let us in turn consider a countable class of *infinite* languages. Let p_i be the i-th prime number (for $i \geq 1$), and $S_i = \{n \mid n \text{ is a multiple of } p_i\}$. $\mathcal{C}_{prime} = (S_i)_{i \in \mathbb{N}-\{0\}}$ is finitely identifiable. Our learner will refrain from a natural number answer as long as it receives non-primes. At the step when a prime is seen, it outputs the corresponding index, and repeats the "$\uparrow$"-answer from that point on. Since every prime number is an element of only one language in $\mathcal{C}_{pair}$, our learner's only conjecture will be the right one.

For contrast, let us look at a very simple class $\mathcal{C}_{simp} = \{S_0 = \{0\}, S_1 = \{0, 1\}\}$. It should be immediately evident this class is not finitely identifiable, but let us give a simple argument, as this example inspires the more complex kind of identifiability we will discuss shortly. Assume, for contradiction, that there is a learner that finitely identifies $\mathcal{C}_{simp}$. Then, by definition, on a finite initial segment of the text for S_0 the learner will output the index of S_0, i.e., "0", and it will be its only natural number answer. So, if after that it turns out that the text was in fact for S_1, i.e., after that point the text will show 1, the learner will fail to identify S_1 on this text for S_1, because it has already used its "one-shot" conjecture. This contradicts the assumption that the learner finitely identifies $\mathcal{C}_{simp}$.

The class $\mathcal{C}_{simp}$ often inspires a (logical) objection: why is negative information disallowed? After all, if the learner is given an observation "not-1", they will be able to settle the problem straight away. In this paper we restrict our attention to learning from positive information only. There are several reasons for this, in particular our desire to stay as close as possible to unsupervised learning. In any case, it is important to mention that learning from the so-called *informant*, i.e., from streams enumerating all positive and negative information consistent with the unknown language, is widely studied in inductive inference, recently even within the paradigm of finite identifiability (see [18, 32]).

Note that since in finite identifiability the learner is only allowed "one shot" at a correct conjecture, its hypotheses should be based on certainty, requiring the learner to have eliminated all other possibilities by the time the conjecture is made. Let us now turn to a more relaxed learner, which is allowed to change its mind, but is still required to *stabilize* to a correct answer after finitely many steps.

▶ **Definition 5** (Identifiability in the limit [24]). *Let $\mathcal{C} = (S_i)_{i \in \mathbb{N}}$, and let L be a learner.*
1. *L identifies S_i in $\mathcal{C}$ in the limit on τ iff there is a $k \in \mathbb{N}$, s.t. for all $n \geq k$ $L(\tau \upharpoonright n) = i$;*
2. *L identifies S_i in $\mathcal{C}$ in the limit iff it identifies S_i in the limit on every τ for S_i;*
3. *L identifies $\mathcal{C}$ in the limit iff it identifies in the limit every S_i in $\mathcal{C}$.*

A class $\mathcal{C}$ is identifiable in the limit iff there is a learner L that identifies $\mathcal{C}$ in the limit.

▶ **Example 6.** Let us start with a class that is identifiable in the limit. Consider $\mathcal{C}_{init} = \{S_n = \{0, \ldots, n\} \mid n \in \mathbb{N}\}$, i.e., the class of finite initial segments of $\mathbb{N}$. The learner witnessing identifiability in the limit of this class, on the initial segment $\tau \upharpoonright n$ of a text τ for a language in $\mathcal{C}_{init}$, will output the largest n in $\text{set}(\tau \upharpoonright n)$. For every text for any language S_i in $\mathcal{C}_{init}$, i is the largest number that will ever be enumerated, so our learner will indeed stabilize on a correct language. For that to be possible, the learner must remain open-minded, i.e., the moment in which the right hypothesis is reached is not known to the learner.

Consider now a countable class of infinite sets contaning complements of the sets in $\mathcal{C}_{init}$, i.e., the class $\mathcal{C}_{coinit} = \{S_n = \mathbb{N} - \{0, \ldots, n\} \mid n \in \mathbb{N}\}$. This class is identified in the limit by a learner that always answers with the smallest natural number that was *not* eliminated so far. If we extend $\mathcal{C}_{coinit}$ to include also the set of all natural numbers, i.e., we consider the class $\mathcal{C}_{coinit^+} = \{S_0 = \mathbb{N}\} \cup \{S_{n+1} = \mathbb{N} - \{0, \ldots, n\} \mid n \in \mathbb{N}\}$ we will retain identifiability in the limit: our learner should now answer "$n + 1$" as long as the smallest number it hasn't encountered so far is n, and upon receiving 0 the learner should changes the conjecture to "0", which stands for $S_0 = \mathbb{N}$.

Sometimes adding $\mathbb{N}$ to an identifiable in the limit class can get is in trouble. Let's return to the class $\mathcal{C}_{init}$ and extend it with $\mathbb{N}$, i.e., consider $\mathcal{C}_{init^+} = \{S_0 = \mathbb{N}\} \cup \{S_{n+1} = \{0, \ldots, n\} \mid n \in \mathbb{N}\}$. This class is not identifiable in the limit. To see this we will construct an argument similar to the one for the class $\mathcal{C}_{simp}$ in Example 4. Assume that $\mathcal{C}_{init^+}$ is identified in the limit by some learner L. Then L identifies in the limit $S_0 \in \mathcal{C}_{init^+}$ on a text τ for S_0. This means that after some finite initial segment $\tau \restriction n$, L stabilizes to "0" (the index for $\mathbb{N}$). But $\tau \restriction n$ is also an initial segment of a text τ' for S_n such that n is the largest number in set$(\tau \restriction n)$. It follows that L does not identify in the limit S_n on the text τ', which contradicts our assumption that L identifies $\mathcal{C}_{init^+}$ in the limit.

The concept of identifiability, its various kinds and characterisations, is fascinating and complex (for an introductory overview see, e.g., [30]). The above simple examples were selected because they will be instructive for us later on. For now there are two takeaways. Firstly, it's important to realize that identifiability has a lot to do with inclusions among the sets in the class being learned. Secondly, the two types of identifiability differ with respect to their success conditions. Finite identifiability is "one-shot" learning, it results in (and requires) certainty about a correct conjecture. Identifiability in the limit does not – the learner is always allowed to change their conjecture, but (depending on the structure of the class) it can be successful in the sense that its conjecture will become "safe", i.e., it will never change, because after some finite time it will not be contradicted by new (truthful) incoming information anymore.

1.2 Epistemic modal logic

In both kinds of identifiability reaching the correct hypothesis can be seen as arriving at a kind of knowledge. While finite identifiability results in *certainty*, identifiability in the limit caries *safe* or *reliable belief*. Such knowledge states are the topic of study of (multi-agent) epistemic and doxastic modal logic. This line of research interprets modal box $\Box$ as a knowledge (or belief) operator, so that the intended meaning of the formula $\Box\varphi$ is "the agent knows that φ" (or "the agent believes that φ").

▶ **Definition 7.** *Let P be a countable set of propositions, with $p \in P$, and $\mathcal{A} = \{1, \ldots, n\}$ is a set of agents, with $i \in \mathcal{A}$ as the name of some agent. The syntax of (epistemic) modal logic is given by:*

$$\varphi := \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box_i\varphi$$

Most commonly, epistemic languages are interpreted over Kripke models.

▶ **Definition 8.** *A* Kripke model *M for n agents over P is a tuple $(W, v, (R_i)_{i\in\mathcal{A}})$, where: W is a non-empty set worlds; $v : P \to 2^W$ is a valuation; for each agent i, R_i is a binary accessibility relation on W. A* pointed Kripke model *is a pair (M, w), of a Kripke model M and a designated world w from its domain W.*

▶ **Definition 9.** *We write* $(M, w) \models \varphi$ *to express that* φ *is true at* w *in* M*, and* $(M, w) \not\models \varphi$ *that* φ *is not true at* w *in* M*. The semantics of our language is defined in the following way:*

$(M, w) \models \top$ *always*
$(M, w) \models p$ *iff* $w \in V(p)$
$(M, w) \models \neg\varphi$ *iff* $(M, w) \not\models \varphi$
$(M, w) \models \varphi \wedge \psi$ *iff* $(M, w) \models \varphi$ *and* $(M, w) \models \psi$
$(M, w) \models \Box_i\varphi$ *iff for all* v *with* $(w, v) \in R_i$, $(M, v) \models \varphi$

The last clause of this definition complies to the intuitive understanding of what it means to know φ, i.e., φ being true in all worlds considered possible by (or "accessible to") the agent. The properties of knowledge understood in this way will vary depending on the properties of the accessibility relations R (on the so-called frame conditions). The validities, when $\Box_i$ is interpreted as knowledge, give additional insights into the properties of knowledge. For instance, the formula $\Box_i\varphi \to \Box_i\Box_i\varphi$ is read as "if the agent i knows that φ, then i knows that it knows φ", hence the name of this property: *positive introspection* of knowledge. This formula is valid in the class of models with transitive accessibility relation. We say that an axiom system Ax is a *logic of a class of models* $\mathcal{M}$ iff Ax is sound and complete with respect to $\mathcal{M}$, see Table 1. Two systems are of special interest to us, the so-called $S5$ and $S4$. $S5$ (axioms K+T+4+5 in Table 1) is the logic of models with equivalence accessibility relations, while $S4$ (K+T+4), is the logic of locally connected, reflexive and transitive relations.

Table 1 Validities with their epistemic interpretations and properties of R_i, for $i \in \mathcal{A}$.

Rules		
(MP)	if $\vdash \varphi$ and $\vdash \varphi \to \psi$, then $\vdash \psi$	
(Necc)	if $\vdash \varphi$, then $\vdash \Box_i\varphi$	
Axioms		
(K)	$\Box_i(\varphi \to \psi) \to (\Box_i\varphi \to \Box_i\psi)$	(omniscience)
(T)	$\Box_i\varphi \to \varphi$	(truthfullness/reflexivity)
(D)	$\Box_i\varphi \to \neg\Box_i\neg\varphi$	(consistency/seriality)
(4)	$\Box_i\varphi \to \Box_i\Box_i\varphi$	(positive introspection/transitivity)
(5)	$\neg\Box_i\varphi \to \Box_i\neg\Box_i\varphi$	(negative introspection/Euclidean-ness)

Dynamic extension: Public Announcement Logic

Epistemic logic is made dynamic by adding action modalities. The first extension of this type was Public Announcement Logic (PAL, [31]), whose language is that of epistemic modal logic, enriched by the public announcement operator $[!\varphi]$.

$$\varphi := \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box_i\varphi \mid [!\varphi]\varphi$$

The semantics of epistemic modal logic given in Definition 9 is accordingly extended in the following way: $(M, w) \models [!\varphi]\psi$ iff $(M, w) \models \varphi$ implies $(M|\varphi, w) \models \psi$, where $M|\varphi = (W', v', (R'_i)_{i\in\mathcal{A}})$ is defined as follows: $W' := \{w \in W \mid (M, w) \models \varphi\}$, $v' := v$ restricted to W'; for each $i \in \mathcal{A}$, $R'_i := R_i \cap (W' \times W')$. Public announcement can be seen as incorporating new information in a radical way, by performing joint update – the accessibility relations of all agents are restricted to the set of worlds satisfying the incoming information φ, other worlds are simply disregarded.

Plausibility models and belief

Plausibility models (introduced in [9]) are a special kind of Kripke models in which the accessibility relations are *plausibility* relations. Plausibility satisfies the following conditions: it's reflexive, transitive, locally connected, and well-founded. It allows an interpretation of both certain knowledge (in the sense of $S5$), but also a weaker kind, called *safe belief* ($S4$), which is a belief that remains true under receiving true information.

We will use the symbol $\preceq_i$ to denote plausibility of the agent i, with $v \preceq_i w$ standing for v is at least as plausible as w, which means that the worlds minimal according to $\preceq_i$ will be seen as the most plausible ones for the agent i. The corresponding epistemic-doxastic language, called $K\Box$, includes two epistemic modalities, K_i interpreted as "hard information" possessed by the agent i, i.e., information that is true in all possible worlds in the connected component of agent i, while $\Box_i$ stands for safe belief, i.e., the information that is true in all worlds that are at least as plausible for agent i.

$$\varphi := \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi \mid \Box_i\varphi$$

This language is interpreted over a plausibility model, which is a tuple $M = (W, (\preceq_i)_{i\in\mathcal{A}}, v)$. We define the *connected component* of $\preceq_i$ as follows: $\mathtt{cc}_i(w) := \{v \in W \mid w(\preceq_i \cup \succeq_i)^* v\}$. The semantics for the propositional part is as usual, while the modal operators are defined in the following way:

$$\begin{array}{lll} (M,w) \models K_i\varphi & \text{iff} & \text{for all } v \in \mathtt{cc}_i(w) \text{ with } (M,v) \models \varphi \\ (M,w) \models \Box_i\varphi & \text{iff} & \text{for all } v \text{ with } v \preceq_i w,\ (M,v) \models \varphi \end{array}$$

We can also define regular (possibly false) belief in φ, as φ being true in all most plausible worlds.

$$(M,w) \models B_i\varphi \quad \text{iff} \quad \text{for all } v \in min_{\preceq_i}(W),\ (M,v) \models \varphi$$

This interpretation of belief is consistent with the standard approach of the belief revision theory (see, e.g., [36]). To know more about plausibility models and the logic $K\Box$ the reader is encouraged to consult adequate passages of [8].

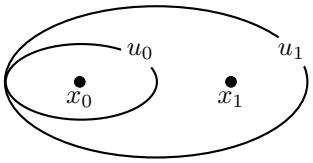
2 Classes of languages as epistemic spaces

To express our inductive inference problems in the setting of epistemic modal logic we need to establish common ground between the two paradigms. In what follows we will consider the case of a single-agent learning, and so we will simplify the framework of epistemic logic accordingly, by omitting the agent indices $i \in \mathcal{A}$. First, we will express the classes of languages from Section 1.1 as single-agent *epistemic spaces*.

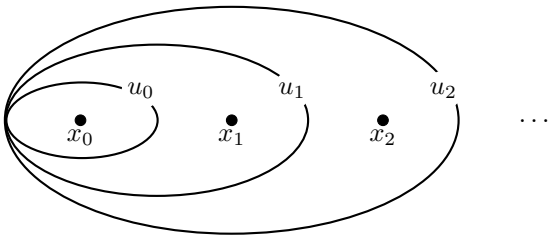
▶ **Definition 10.** *An* epistemic space *is* $\mathcal{S} = (X, \mathbb{U})$*, where* $\mathbb{U} \subseteq 2^X$ *and both* X *and* $\mathbb{U}$ *are at most countable.*

Given a class of languages $\mathcal{C} = \{S_i \mid i \in I\}$, we set its corresponding epistemic space to be $\mathcal{S}_\mathcal{C} = (X_\mathcal{C}, \mathbb{U}_\mathcal{C})$, with $X_\mathcal{C} = \{x_i \mid i \in I\}$ and for each $u_n \in \mathbb{U}_\mathcal{C}$, $x_k \in u_n$ iff $n \in S_k$.

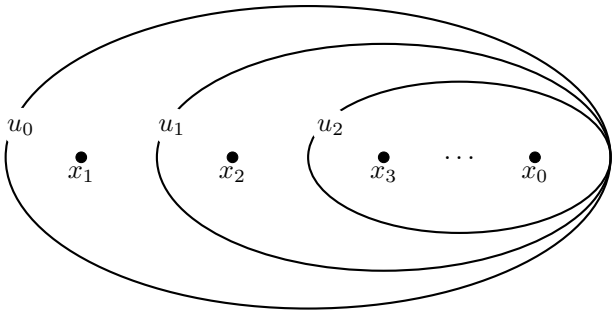
▶ **Example 11.** Recall the class $\mathcal{C}_{simp}$ from Example 4, the corresponding epistemic space is $\mathcal{S} = (X, \mathbb{U})$, $X = \{x_0, x_1\}$, and $\mathbb{U} = \{u_0, u_1\}$ with $u_0 = \{x_0\}$, $u_1 = \{x_0, x_1\}$. The space is depicted below.



▶ **Example 12.** The epistemic space for $\mathcal{C}_{coinit^+}$ from Example 6 is depicted below.



▶ **Example 13.** The epistemic space for $\mathcal{C}_{init^+}$ from Example 6 is depicted below.



Note that epistemic spaces can be seen as single-agent Kripke models with an equivalence accessibility relation – all worlds are considered equally plausible.

We will now reformulate the learning notions from Section 1.1 in terms of epistemic spaces.

▶ **Definition 14.** *Let $S = (X, \mathbb{U})$ be an epistemic space. A* data stream *is an infinite sequence $\vec{U} = (u_0, u_1, \ldots)$ from $\mathbb{U}$. A* data sequence *$\vec{U} \restriction n$ is a finite initial segment of $\vec{U}$ of length $n+1$. A data stream $\vec{U}$ is* sound *with respect to x if every element listed in $\vec{U}$ contains x, and it is* complete *with respect to x if every element containing x is listed in $\vec{U}$.*

▶ **Definition 15.** *Let $\mathcal{S} = (X, \mathbb{U})$ be an epistemic space and let σ be a data sequence. A* learner *L is a function that on a data sequence σ outputs a conjecture, a subset of X, $L(\sigma) \subseteq X$.*

Thus, a learner outputs a subset of an epistemic space, which is more general than in the case of the classical learner in Section 1.1. This is in order to allow the learner to make more general conjectures.

▶ **Definition 16.** *An epistemic space $\mathcal{S} = (X, \mathbb{U})$ is* finitely identified *by a learner L if for every $x \in X$ and every data stream $\vec{U}$ for x, L is once-defined and there is $k \in \mathbb{N}$ such that $L(\vec{U} \restriction n) = \{x\}$. $\mathcal{S}$ is* finitely identifiable *if it is finitely identified by a learner L.*

▶ **Definition 17.** *An epistemic space $\mathcal{S} = (X, \mathbb{U})$ is* identified in the limit *by a learner L if for every $x \in X$ and every data stream $\vec{U}$ for x, there is $k \in \mathbb{N}$ such that $L(\vec{U} \restriction n) = \{x\}$ for all $n \geq k$. $\mathcal{S}$ is* identifiable in the limit *if it is identified in the limit by a learner L.*

3 Learning by belief revision

Learning often proceeds by an underlying implicit order of possible worlds. This brings to mind the plausibility relation described in Section 1.2. Let us consider the epistemic space corresponding to $\mathcal{C}_{coinit^+}$ in Example 12. The successful learner, upon receiving information u, conjectures the *maximal*, according to the indices, hypothesis consistent with u. This can be interpreted as restricting the X to u, which leads the agent to "believe" everything that is true in the most plausible worlds in the restricted domain. This is how belief revision operators work in general, they adjust plausibility based on what is observed. In the contributions [22, 4, 6] it has been shown how belief revision methods can be seen as constructive strategies for converging to the right conjecture. An interesting question is how well various belief revision methods perform as learning engines. In order to capture this intuition we will now build up towards belief-revision based learners.

▶ **Definition 18.** *Let $\mathcal{S} = (X, \mathbb{U})$ be an epistemic space. A* plausibility assignment *PL is a map that assigns to $\mathcal{S}$ some plausibility (total, reflexive, and transitive) relation $\preceq$ on X, thus converting it into a plausibility space $PL(\mathcal{S}) = (X, \mathbb{U}, \preceq)$.*

Plausibility spaces are single-agent plausibility models from Section 1.2, and the language of $K\Box$ is interpreted on them in an analogous way. Note that the above definition allows the plausibility assignment to be non well-founded on X, i.e., there might be an infinite descending chain $s_0 \succ s_1 \succ s_2 \succ \ldots$, where $\prec$ is the strict plausibility relation, given by: $s \preceq t$ and $t \not\preceq s$). This requires a modification of the definition of belief so that we cover the case when the minimum of $\preceq$ does not exist. This is done in the following way: $(M, w) \models B_i\varphi$ iff there is a k such that for all $v \preceq_i k$, $(M, v) \models \varphi$.

Plausibility order will be revised upon receiving new information. The ways of revision vary, but they can be given a general definition.

▶ **Definition 19.** *A* one-step revision method *is a function R_1 that, for any plausibility space $\mathcal{B} = (X, \mathbb{U}, \preceq)$ and any $u \in \mathbb{U}$, outputs a new plausibility space $R_1(\mathcal{B}, u)$. A* (iterated) belief revision method *R is obtained by iterating a one-step revision method R_1: $R(\mathcal{B}, \lambda) = \mathcal{B}$, for the empty data sequence λ, and for a data sequence σ and $u \in \mathbb{U}$, $R(\mathcal{B}, \sigma^\frown u) = R_1(R(\mathcal{B}, \sigma), u)$.*

▶ **Definition 20.** *Every belief revision method R, together with a plausibility assignment PL, generates a learning method L_R^{PL}, called a* belief revision-based learning method*, given by:*

$$L_R^{PL}(\mathcal{S}, \sigma) := \min R(PL(\mathcal{S}), \sigma).$$

With this apparatus at hand, we can ask the question: how good are belief revision methods as learning methods? Among many be will consider three widely studied belief-revision policies: conditioning, the minimal, and the lexicographic revision (a DEL account of those is given in [10]). Conditioning removes possible worlds inconsistent with the incoming information (just like public announcements do); minimal revision selects the best worlds satisfying the incoming information and makes them the most plausible; and lexicographic revision puts all the worlds satisfying the incoming information to be better than all the ones that do not (within the two clusters the order remains the same). The relevant question here is: given a belief revision method, can the belief-revision based learner identify in the limit everything that's learnable by an arbitrary learning method? If it is so, we say that revision method generates a *universal* learning method. It turns out that under the condition of sound and complete streams conditioning and lexicographic revision are universal, while minimal revision *is not*. If we relax the condition of soundness on data streams (a stream is

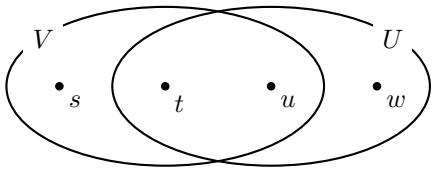
fair, if it allows finitely many errors, and each error must be accounted for truthfully in the future), conditioning loses its power – once a possible world is removed on the basis of an incorrect piece of data, it cannot be revived, and so conditioning is no longer an universal learning method. Lexicographical revision remains universal on fair streams.

Another important insight from this analysis is that reaching the full learning power of belief revision requires non-standard plausibility relations, which is important because in belief revision theory plausibility is typically assumed to be well-founded. Under the assumption of well-foundedness none of the revision methods generates a universal learning method. The epistemic space of the class $\mathcal{C}_{coinit^+}$ in Example 12 shows why – initially the learner must allow all possible worlds, and so must assume infinitely increasing plausibility, i.e., if $i < j$ then $x_j \prec x_i$. If a well-founded plausibility is initially assigned to this space, the learning process will inevitably overgeneralise while attempting to learn some $x \in X$.

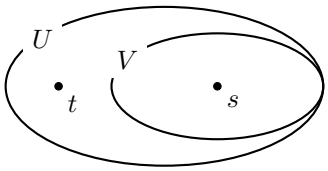
4 Topological perspective on inductive inference

Whether or not an epistemic space is learnable has a lot to do with possible worlds being *separable* from others by the incoming data. General topology [25] not only puts separability properties as first class citizens, but it has also been proposed as a logic of observation [37]. In this perspective the points of the space represent possible worlds and basic open neighborhoods of a point code information that comes about from executing finitely many observations or measurements (for examples, see [3]).

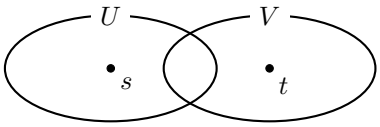
Figure 1 shows several basic examples of how points can be separated by observations.



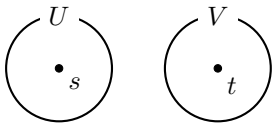
(a) t and u are not separable.



(b) T_0 space.



(c) T_1 space.



(d) T_2 space.

Figure 1 Some topological separation properties.

The necessary condition for identifying individual worlds is T_0: the space cannot contain two inseparable points, i.e., for any two worlds x and y, there must be an observation that separates them one way or the other, i.e., there must be an o such that $x \in o$ and $y \notin o$, or $y \in o$ and $x \notin o$. As a matter of fact, finite identifiability requires a stronger condition, in which every state has an observation that separates it from all other states in the space. Finding a condition for identifiability in the limit requires a more specific condition, concerning singleton worlds being *locally closed*.

▶ **Definition 21.** *A topological space* $(X, \mathbb{O})$ *is* T_d *iff* $\forall x \in X\ \exists U \in \mathbb{O}\ U \setminus \{x\} \in \mathbb{O}$.

In order to apply the tools of topology to epistemic spaces, we have to close them on finite intersections (corresponding to accumulating finitely many observations) and arbitrary unions. The latter addition does not disrupt our learning scenarios, since arbitrary unions do

not bring any extra information to the learning process. By considering a topology, we also allow our learners (in principle) to observe an $\emptyset$, i.e., contradictory datum – in the setting of learning from positive data, this will never happen.

▶ **Definition 22.** *Let $\mathcal{S} = (X, \mathbb{U})$ be an epistemic space.* A topology generated by $\mathcal{S}$ *is $\tau_{\mathcal{S}} = (X, \mathbb{O})$, where $\mathbb{O}$ is defined in the following way:*
1. *if $u \in \mathbb{U}$, then $u \in \mathbb{O}$;*
2. *$\emptyset \in \mathbb{O}$.*
3. *if O is a finite subset of $\mathbb{O}$, then $\cap O \in \mathbb{O}$;*
4. *if O is an arbitrary subset of $\mathbb{O}$, then $\cup O \in \mathbb{O}$;*
5. *$X \in \mathbb{O}$.*

We obtain the following characterisation identifiable in the limit epistemic spaces.

▶ **Theorem 23** ([5]). *An epistemic space $\mathcal{S}$ is identifiable in the limit iff its generated topology $\tau_{\mathcal{S}}$ is T_d.*

Topological analysis of inductive inference has been picked up in similar settings: [34] propose the so-called *positive information topology*. [17] analyzes concept spaces (structures like our classes of languages in Section 1.1) and also obtains a T_d characterisation. That last contribution also contains interesting results involving topological and algebraic characterisations that could without a doubt be of use in logical approaches to learning.

4.1 Topological semantics of modal logic

Having established the topological characterisation of identifiable in the limit spaces, we can now ask the question of a corresponding modal logic. It's only natural we consider topological semantics. We will now interpret standard modal language on topological models via the so-called topo-semantics.

▶ **Definition 24.** *A* topological model *(or a topo-model) $M = (X, \mathbb{O}, v)$ is a topology $(X, \mathbb{O})$ together with a valuation function $v : P \to 2^X$.*

▶ **Definition 25** (Topo-semantics). *Let $M = (X, \mathbb{O}, v)$ be a topological model and $x \in X$:*

$$
\begin{array}{lll}
M, x \models \top & & \textit{always} \\
M, x \models p & \textit{iff} & x \in v(p) \\
M, x \models \neg\varphi & \textit{iff} & \textit{not } M, x \models \varphi \\
M, x \models \varphi \wedge \psi & \textit{iff} & M, x \models \varphi \textit{ and } M, x \models \psi \\
M, x \models \Box\varphi & \textit{iff} & \textit{there is } U \in \tau(x \in U \textit{ and for all } y \in U\colon M, y \models \varphi)
\end{array}
$$

Under this interpretation of modal box, we get, for instance, that $S4$ is the topo-logic of all topological spaces [28]. Unfortunately, it is also known that T_d spaces, the identifiability-adequate class of models, is not topo-definable. We can however change the way we view $\Box$, allowing belief to be false (see Figure 2). This agrees with the kind of belief occurring in learning in the limit.

▶ **Definition 26** (d-semantics). *Given a topological model $M = (X, \mathbb{O}, v)$ and a state $x \in X$:*

$$
\begin{array}{lll}
M, x \models_d \top & & \textit{always} \\
M, x \models_d p & \textit{iff} & x \in v(p) \\
 & \cdots & \\
M, x \models_d \Box\varphi & \textit{iff} & \exists U \in \tau(x \in U \;\&\; \forall y \in U - \{x\}\; M, y \models_d \varphi)
\end{array}
$$

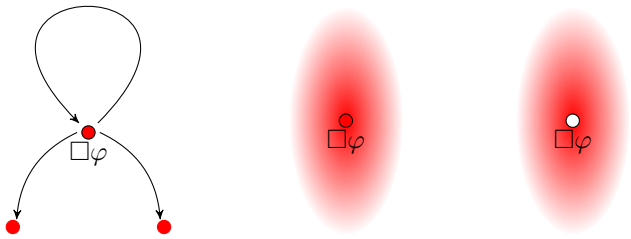


Figure 2 Relational (left), topo-semantic (center) and d-semantic (right) interpretations of □, with φ true in red areas.

Quite a lot is known about sound and complete *d*-axiomatisations (for an overview, see [11]): wKD45 is the d-logic of dense spaces; KD45 is the d-logic of DSO-spaces, where DSO stands for "derived sets are open"; GL is the d-logic of scattered spaces, where GL is the Grzegorczyk axiom $\Box(\Box\varphi \to \varphi) \to \Box\varphi$; weak K4 is the d-logic of all topological spaces. Finally, we also have that K4 is the d-logic of all T_d-spaces. This is a technically valuable result, but given the relative poverty of our modal language it gives us very little insight into the possible *epistemic* interpretations of this modal language characterising spaces that are identifiable in the limit. To be of use a logic of learning theory should involve a semantic notion of a learner that drives beliefs, and also dynamic modalities to talk about receiving new information. Dynamic Logic for Learning Theory (DLLT, [3]), built upon Subset Space Logic [16] overcomes both challenges.

4.2 Dynamic Logic of Learning Theory

With Subset Space Logic (SSL, [16]) we are back to an epistemic-doxastic language with two modalities, like the one in Section 1.2.

▶ **Definition 27** (SSL Syntax). *Let P be a countable set of propositional symbols and $p \in P$.*

$$\varphi := \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K\varphi \mid \Box\varphi$$

This logic is interpreted in topological models, in this context called *intersection models.*

▶ **Definition 28.** *An* intersection model $M = (X, \mathbb{O}, v)$ *is a topology* $(X, \mathbb{O})$ *together with a valuation function* $v : P \to 2^X$.

The meaning of formulas of SSL is defined in the following way.

▶ **Definition 29** (SSL Semantics). *Let $M = (X, \mathbb{O}, v)$, $U \in \mathbb{O}$, and $x \in U$.*

$$\begin{array}{lll} M, x, U \models p & \textit{iff} & x \in v(p) \\ M, x, U \models \neg\varphi & \textit{iff} & M, x, U \not\models \varphi \\ M, x, U \models \varphi \wedge \psi & \textit{iff} & M, x, U \models \varphi \textit{ and } M, x, U \models \psi \\ M, x, U \models K\varphi & \textit{iff} & \forall y \in U \;\; M, y, U \models \varphi \\ M, x, U \models \Box\varphi & \textit{iff} & \forall O \in \mathcal{O} \;\; \textit{if } x \in O \subseteq U \textit{ then } M, x, O \models \varphi \end{array}$$

In the above definition □ (also called effort modality) accounts for *observational effort* – making the epistemic effort to obtain more information about a possible world. It denotes how shrinking the open neighborhood gives a more accurate approximation of the actual state of the world. Note that the formulas of SSL are evaluated at triples, the novel addition

being the third component, i.e., a particular open neighbourhood of the point at which the formula is evaluated. This can be seen as evaluating formulas at some stage of inquiry – a starting actual world, together with some truthful information already received.

Dynamic Logic of Learning Theory (DLLT, [3]) extends SSL in the following way.

▶ **Definition 30** (DLLT Syntax). *Let p and o be drawn from countable sets of propositional and observational symbols, P and O respectively.*

$$\varphi := p \mid o \mid L(\vec{o}) \mid \neg\varphi \mid \varphi \wedge \varphi \mid K\varphi \mid \Box\varphi \mid [o]\varphi$$

Note that the language of DLLT allows talking directly about observations. The two remaining additions to the language are dynamic. The first one allows expressing Learner's conjectures after being given some finite sequence of observations. The second is a more basic "learning" known already from PAL. The learning model has to accordingly involve a semantics notion of the learner.

▶ **Definition 31.** *The structure $M_L = (X, \mathbb{O}, \mathbb{L}, v)$ is a* learning model *consisting of a topology $(X, \mathbb{O})$; the learner $\mathbb{L} : \mathbb{O} \to 2^X$ is such that $\mathbb{L}(O) \subseteq O$, and if $O \neq \emptyset$ then $\mathbb{L}(O) \neq \emptyset$. Additionally: $\mathbb{L}(\vec{O}) := \mathbb{L}(\bigcap \mathrm{set}(\vec{O}))$. Finally, $v : P \cup O \to 2^X$ is a valuation.*

▶ **Definition 32** (DLLT semantics). *Given a learning model M_L, $U \in \mathbb{O}$, and $x \in U$:*

$$\begin{array}{lll} M, x, U \models o & \textit{iff} & x \in v(o) \\ M, x, U \models L(o_1, \ldots, o_n) & \textit{iff} & x \in \mathbb{L}(U, v(o_1), \ldots, v(o_n)) \\ M, x, U \models [o]\varphi & \textit{iff} & x \in v(o) \textit{ implies } M, x, U \cap v(o) \models \varphi \end{array}$$

Clauses for p, $\neg$, $\wedge$, K, and $\Box$ are as in the semantics of SSL, see Definition 29.

The paper [3] provides a sound and complete axiomatisation of DLLT with respect to the learning models. The advantage of this logic is not only that it builds upon SSL, which with its effort modality $\Box$ has a very natural epistemic interpretation, but also that it allows expressing the conditions of learnability discussed in the beginning of this paper. We have that: $\mathcal{M}, x, U \models \Diamond Kp$ iff p is *finitely learnable* at x and $\mathcal{M}, x, U \models p \wedge \Diamond\Box Bp$ iff p is *learnable in the limit* by $\mathbb{L}$ at x (here B is an abbreviation $B^{\vec{o}}\varphi := K(L(\vec{o}) \to \varphi)$ and $B\varphi := B^{\lambda}\varphi$).

5 Knowledge in computable inductive inference

Computable inductive inference is concerned with learnability under the assumption of recursivity of the learner. It also often assumes that languages are, at least, recursively enumerable. Since the learner should be able to decide whether a given observation is consistent with a given hypothesis, indexed families of (non-empty) recursive languages are extensively studied as the background of learning. These are $\mathcal{C} = (S_i)_{i \in \mathbb{N}}$ for which a computable function $f : \mathbb{N} \times U \to \{0, 1\}$ exists that uniformly decides $\mathcal{C}$, i.e.:

$$f(i, w) = \begin{cases} 1 & \text{if } w \in S_i, \\ 0 & \text{if } w \notin S_i. \end{cases}$$

Even though the existence of such a computable function seems like a natural element of any agency, the logical approaches to knowledge representation, both belief revision and epistemic logic, do not consider this assumptions, or treat it as implicit. Belief revision operators are usually studied from the perspective of rationality postulates, and epistemic logic takes agents (learners) to be relations in a (dynamic) Kripke model, abstracting away from computability. The decision of whether or not $w \models p$ is taken to be an atomic step of computation which does not deserve much attention.

In this section we will assume the perspective of learning theory on certain aspects of knowledge that are usually taken for granted in epistemic logic and turn out to be restrictive when seen through the lenses of computable learning. We will start with finite identifiability and then move on to identifiability in the limit.

What are the conditions for reaching certainty? Intuitively, it seems that as soon as the agent receives information that is inconsistent with all the alternatives, she can be sure about the correct conjecture. Recall the concept of finite identifiability given in Definition 3. The necessary and sufficient condition for finite identifiability [29, 27] involves the *definite finite tell-tale set*, a more strict kind of tell-tale than the one involved in the characterisation of identifiability in the limit [2].

▶ **Definition 33** ([29, 27]). *Let $\mathcal{C}=(S_i)_{i\in\mathbb{N}}$ be an indexed family of recursive languages. A set D_i is a definite finite tell-tale set ($DFTT$) for S_i in $\mathcal{C}$ if*

1. *$D_i \subseteq S_i$,*
2. *D_i is finite, and*
3. *for any index j, if $D_i \subseteq S_j$ then $S_i = S_j$.*

▶ **Theorem 34** ([29, 27]). *An indexed family of recursive languages $\mathcal{C}=(S_i)_{i\in\mathbb{N}}$ is finitely identifiable from positive data iff there is an effective procedure $\mathcal{D} : \mathbb{N} \to \mathcal{P}^{<\omega}(\mathbb{N})$, given by $n \mapsto D_n$, that on input i produces a definite finite tell-tale of S_i in $\mathcal{C}$.*

Let us again take a finitely identifiable class $\mathcal{C}$, and S_i in $\mathcal{C}$. Now, consider the collection $\mathbb{D}_i$ of all DFTTs of S_i in $\mathcal{C}$.

▶ **Definition 35.** *Let $\mathcal{C}=(S_i)_{i\in\mathbb{N}}$ be an indexed family of recursive sets. $\mathcal{C}$ is* finitely identifiable in the fastest way *if and only if there is a learning function L such that, for each τ and for each $i \in \mathbb{N}$, $L(\tau\restriction n) = i$ iff $\exists D_i^j \in \mathbb{D}_i$ $(D_i^j \subseteq \mathrm{set}(\tau\restriction n))$ and $\neg\exists D_i^k \in \mathbb{D}_i$ $(D_i^k \subseteq \mathrm{set}(\tau\restriction n-1))$. We will call such L a* fastest learning function.

The fastest learning thus our desired learner, as soon as a definite finite telltale arrives, certainty is reached and the correct conjecture is issued. It turns out that, even under the assumptions of uniform recursivity of the class and recursivity of the learner, things are not that simple.

▶ **Theorem 36** ([23]). *There exists an indexed family of recursive sets $\mathcal{C}=(S_i)_{i\in\mathbb{N}}$ that is recursively finitely identifiable but is not recursively finitely identifiable in the fastest way.*

The proof of this theorem relies on recursively inseparable sets [35]. The the witness class still possesses the well-separated topological structure, but a recursive fastest learner would be able decide that a minimal DFTT has been observed, and so it would provide a recursive separating set for something that is recursively inseparable (for details, see [23]).

Results similar in spirit are very common in identifiability in the limit. Two properties of learners deserve our special attention because of their close connection to epistemic logic and rationality postulates in belief revision. They are *conservativity* and *consistency*.

▶ **Definition 37.** *A learning function L is* conservative *if, for each σ and x,* $\mathrm{set}(\sigma^\frown\langle x\rangle) \subseteq S_{L(\sigma)}$ *implies $L(\sigma^\frown\langle x\rangle) = L(\sigma)$.*

A property of this kind is a guiding principle of belief revision and dynamic epistemic logic. If the incoming information is already accounted for within the learner's conjecture, learning takes no effect. Under the assumptions of computability, conservativity turns out to be a restrictive property – there are uniformly recursive families that can be learned by a recursive learner, but not by a conservative recursive learner.

▶ **Theorem 38** ([39]). *There is a uniformly recursive family of languages $\mathcal{C}$, which is effectively identifiable in the limit, but which is not effectively identifiable by a conservative learner.*

Another crucial principle in learning is consistency, which in this case means consistency of the conjecture with the incoming sequence of information at every point of learning.

▶ **Definition 39.** *A learning function L is* consistent *if, for each σ,* $\text{set}(\sigma) \subseteq S_{L(\sigma)}$.

In AGM Belief Revision theory [1], this principle is called *success postulate* and is seen as one of the rationality postulate. Despite its very obvious character, for classes of recursive languages consistency turns out to be restrictive. For indexed families of recursive languages consistency is not limiting, but it turns out that sometimes issuing conjectures inconsistent with the information obtained so far can significantly speed up the learning process.

▶ **Theorem 40** ([38]). *There is a uniformly recursive class of languages that is polynomially effectively identifiable in the limit from informant but is not polynomially effectively consistently identifiable in the limit from informant.*

6 Conclusions

The standard notion of identifiability in the limit and its little cousin, finite identifiability, naturally correspond to safe belief and certain knowledge. Despite the clear intuitions, it is not easy to capture these analogies. Topological notion of epistemic effort and dynamic extensions of epistemic modal logic provide an adequate set of tools and an appropriate level of expressivity. The common ground between the theory of inductive inference and epistemic modal logic opens many opportunities for cross-fertilisation between the fields, e.g., understanding the limitations of computable learning in the context of belief revision, as outlined in Section 5. Another direction is looking into inductive inference of action models of dynamic epistemic logic [12, 13, 14]. Finally, (dynamic) epistemic modal logic is by default developed for multi-agent systems. It is interesting to investigate how that kind of multi-agency can be translated into the settings of computable inductive inference.

References

1 Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50(2):510–530, 1985.

2 Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.

3 Alexandru Baltag, Nina Gierasimczuk, Aybüke Özgün, Ana Lucia Vargas Sandoval, and Sonja Smets. A dynamic logic for learning theory. *Journal of Logical and Algebraic Methods in Programming*, 109:100485, 2019.

4 Alexandru Baltag, Nina Gierasimczuk, and Sonja Smets. Belief revision as a truth-tracking process. In K. Apt, editor, *TARK'11: Proceedings of the 13th Conference on Theoretical Aspects of Rationality and Knowledge, Groningen, The Netherlands, July 12-14, 2011*, pages 187–190. ACM, New York, NY, USA, 2011.

5 Alexandru Baltag, Nina Gierasimczuk, and Sonja Smets. On the solvability of inductive problems: A study in epistemic topology. In R. Ramanujam, editor, *Proceedings Fifteenth Conference on Theoretical Aspects of Rationality and Knowledge, TARK 2015, Carnegie Mellon University, Pittsburgh, USA, June 4-6, 2015*, volume 215 of *EPTCS*, pages 81–98, 2015.

6 Alexandru Baltag, Nina Gierasimczuk, and Sonja Smets. Truth-tracking by belief revision. *Studia Logica*, 107(5):917–947, 2019.

7 Alexandru Baltag, Lawrence S. Moss, and Slawomir Solecki. The logic of public announcements, common knowledge, and private suspicions. In Itzhak Gilboa, editor, *TARK'98: Proceedings of the 7th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 43–56, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

8 Alexandru Baltag and Bryan Renne. Dynamic Epistemic Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2016 edition, 2016.

9 Alexandru Baltag and Sonja Smets. A qualitative theory of dynamic interactive belief revision. In G. Bonanno, W. van der Hoek, and M. Wooldridge, editors, *LOFT'08: Proceedings of 8th Conference on Logic and the Foundations of Game and Decision Theory*, number 3 in Texts in Logic and Games, pages 9–58. Amsterdam University Press, 2008.

10 Johan van Benthem. Dynamic logic for belief revision. *Journal of Applied Non-Classical Logics*, 2:129–155, 2007.

11 Johan van Benthem and Guram Bezhanishvili. Modal logics of space. In Marco Aiello, Ian Pratt-Hartmann, and Johan Van Benthem, editors, *Handbook of Spatial Logics*, pages 217–298. Springer Netherlands, Dordrecht, 2007.

12 Thomas Bolander and Nina Gierasimczuk. Learning actions models: Qualitative approach. In Wiebe van der Hoek, Wesley H. Holliday, and Wen-Fang Wang, editors, *Logic, Rationality, and Interaction - 5th International Workshop, LORI 2015 Taipei, Taiwan, October 28-31, 2015, Proceedings*, volume 9394 of *Lecture Notes in Computer Science*, pages 40–52. Springer, 2015.

13 Thomas Bolander and Nina Gierasimczuk. Learning to act: qualitative learning of deterministic action models. *Journal of Logic and Computation*, 28(2):337–365, 2018.

14 Thomas Bolander, Nina Gierasimczuk, and Andrés Occhipinti Liberman. Learning to act and observe in partially observable domains. In Nick Bezhanishvili, Rosalie Iemhoff, and Fan Yang, editors, *Outstanding Contributions to Logic: Dick de Jongh on intuitionistic and provability logics*. Springer, 2023.

15 John Case and Timo Kötzing. Topological separations in inductive inference. *Theoretical Computer Science*, 620:33–45, 2016.

16 Andrew Dabrowski, Lawrence S. Moss, and Rohit Parikh. Topological reasoning and the logic of knowledge. *Annals of Pure and Applied Logic*, 78(1):73–110, 1996. Papers in honor of the Symposium on Logical Foundations of Computer Science, Logic at St. Petersburg.

17 Matthew de Brecht and Akihiro Yamamoto. Topological properties of concept spaces. *Information and Computation*, 208(4):327–340, 2010.

18 Dick de Jongh and Ana Lucia Vargas-Sandoval. Finite identification with positive and with complete data. In Alexandra Silva, Sam Staton, Peter Sutton, and Carla Umbach, editors, *Language, Logic, and Computation*, pages 42–63, Berlin, Heidelberg, 2019. Springer Berlin Heidelberg.

19 Hans van Ditmarsch, Wiebe Van der Hoek, and Barteld Kooi. *Dynamic Epistemic Logic*. Springer, The Netherlands, 2007.

20 Nina Gierasimczuk. Bridging learning theory and dynamic epistemic logic. *Synthese*, 169(2):371–384, 2009.

21 Nina Gierasimczuk. Learning by erasing in dynamic epistemic logic. In Adrian Horia Dediu, Armand Mihai Ionescu, and Carlos Martin-Vide, editors, *LATA'09: Proceedings of 3rd International Conference on Language and Automata Theory and Applications, Tarragona, Spain, April 2-8, 2009*, volume 5457 of *Lecture Notes in Computer Science*, pages 362–373. Springer, The Netherlands, 2009.

22 Nina Gierasimczuk. *Knowing One's Limits. Logical Analysis of Inductive Inference*. PhD thesis, Universiteit van Amsterdam, The Netherlands, 2010.

23 Nina Gierasimczuk and Dick de Jongh. On the complexity of conclusive update. *The Computer Journal*, 56(3):365–377, 2013.

24 E. Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

25 John L. Kelley. *General topology*. Springer-Verlag New York, 1975.

26 Steffen Lange, Rolf Wiehagen, and Thomas Zeugmann. Learning by erasing. In Setsuo Arikawa and Arun Sharma, editors, *ALT*, volume 1160 of *Lecture Notes in Computer Science*, pages 228–241. Springer, 1996.

27 Steffen Lange and Thomas Zeugmann. Types of monotonic language learning and their characterization. In *COLT'92: Proceedings of the 5th Annual ACM Conference on Computational Learning Theory, Pittsburgh, PA, USA, July 27-29, 1992*, pages 377–390. ACM, New York, NY, USA, 1992.

28 John C. C. McKinsey and Alfred Tarski. The algebra of topology. *Annals of Mathematics*, 1944.

29 Yasuhito Mukouchi. Characterization of finite identification. In Klaus Jantke, editor, *AII'92: Proceedings of the International Workshop on Analogical and Inductive Inference, Dagstuhl Castle, Germany, October 5-9, 1992*, volume 642 of *Lecture Notes in Computer Science*, pages 260–267. Springer, Berlin/Heidelberg, 1992.

30 Daniel Osherson, Michael Stob, and Scott Weinstein. *Systems that Learn*. M.I.T. Press, Cambridge, MA, USA, 1986.

31 Jan Plaza. Logics of public communications. In M.L. Emrich, M.S. Pfeifer, M. Hadzikadic, and Z.W. Ras, editors, *Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems*, pages 201–216, 1989.

32 Ana Lucía Vargas Sandoval. *On the Path to the Truth: Logical and Computational Aspects of Learning*. PhD thesis, Universiteit van Amsterdam, The Netherlands, 2020.

33 Oliver Schulte. Formal Learning Theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2022 edition, 2022.

34 Olivier Schulte and Cory Juhl. Topology as epistemology. *Monist*, 79(1):141–147, 1996.

35 Raymond M. Smullyan. Undecidability and recursive inseparability. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 4(7–11):143–147, 1958.

36 Wolfgang Spohn. Ordinal conditional functions: A dynamic theory of epistemic states. In Brian Skyrms and William L. Harper, editors, *Causation in Decision, Belief Change, and Statistics*, volume II, pages 105–134. Dordrecht: Kluwer, 1988.

37 Steven Vickers. *Topology Via Logic*. Cambridge University Press, 1996.

38 Rolf Wiehagen and Thomas Zeugmann. Ignoring data may be the only way to learn efficiently. *J. Exp. Theor. Artif. Intell.*, 6(1):131–144, 1994.

39 Thomas Zeugmann, Steffen Lange, and Shyam Kapur. Characterizations of monotonic and dual monotonic language learning. *Information and Computation*, 120(2):155–173, 1995.

A Positive Perspective on Term Representation

Dale Miller
Inria Saclay, Palaiseau, France
LIX, Institut Polytechnique de Paris, France

Jui-Hsuan Wu
LIX, Institut Polytechnique de Paris, France

Abstract

We use the focused proof system *LJF* as a framework for describing term structures and substitution. Since the proof theory of *LJF* does not pick a canonical polarization for primitive types, two different approaches to term representation arise. When primitive types are given the negative polarity, *LJF* proofs encode terms as tree-like structures in a familiar fashion. In this situation, cut elimination also yields the familiar notion of substitution. On the other hand, when primitive types are given the positive polarity, *LJF* proofs yield a structure in which explicit sharing of term structures is possible. Such a representation of terms provides an explicit method for sharing term structures. In this setting, cut elimination yields a different notion of substitution. We illustrate these two approaches to term representation by applying them to the encoding of untyped λ-terms. We also exploit concurrency theory techniques – namely traces and simulation – to compare untyped λ-terms using such different structuring disciplines.

2012 ACM Subject Classification Theory of computation → Proof theory

Keywords and phrases term representation, sharing, focused proof systems

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.3

Category Invited Talk

Related Version *Full Version*: `https://hal.inria.fr/hal-03843587`

Acknowledgements We thank Beniamino Accattoli and Kaustuv Chaudhuri for their valuable discussions and suggestions. We also thank anonymous reviewers for their comments on an earlier draft of this paper.

1 Introduction

The structure of terms and expressions are represented variously via labeled trees or directed acyclic graphs (DAGs). When such expressions contain bindings, additional devices are needed. We follow a familiar line of research in which the design of term representations is motivated by proof-theoretic considerations. Accordingly, we rely on proof theory in the hope that it will provide careful and formal descriptions of the term structures that serve as the foundation of theorem provers, semantic specifications, interpreters, parsers, and compilers.

Applications of structural proof theory usually start with either natural deduction or sequent calculus proof systems, both of which were introduced by Gentzen in [15]. For our purposes here, these two proof systems are inadequate: natural deduction (with or without *generalized elimination rules* [36]) seems too restrictive, and sequent calculus seems too unstructured. Instead, we use the *LJF* proof system [24], which results from applying the notions of *polarity* and *focusing* [1, 17] to Gentzen's *LJ* proof system as a framework for studying term structures with and without bindings and with and without explicit sharing constructs. By examining the dynamics of cut-elimination in *LJF*, we can also describe substitution into such term structures.

Focused proof theories have been successfully applied within the Curry-Howard correspondence: for example, call-by-value, call-by-name, and call-by-push-value evaluation strategies have been related to different choices in polarizing type expressions [34, 39, 42]. Instead of dealing with functional programming style evaluation, we address a more primitive notion: what is a *term* (used to encode programs), and what is *substitution* (used to describe various kinds of evaluation). Even this more narrow setting has been addressed using the focused proof systems *LJT* [12, 22] and *LJQ* [11]. As in those papers, we shall consider proofs-as-terms and not proofs-as-programs since we do not consider the evaluation of such structures in this paper.

2 The LJF proof system

Gentzen's sequent calculus proof system *LJ* [15] employs rather tiny and slippery inference rules: they are tiny since most of them deal with at most one logical connective at a time, and they are slippery since the order in which they are applied can often be freely reorganized. As a result, it is hard to use *LJ* to identify large-scale structures in proofs. For example, capturing the two phases of proof search in logic programming – *goal-reduction* and *backchaining* – with sequent calculus rules requires rather technical arguments about permutations of inference rules [30]. Andreoli's invention of a *focused proof system* for linear logic [1] provided means for adding more structure to sequent calculus proofs for linear logic. This notion of focusing was eventually moved to intuitionistic and classical logic as the *LJF* and *LKF* proof systems [24]. This paper considers only intuitionistic logic and only for formulas built using implication as the only logical connective.

2.1 LJF inference rules

There are two kinds of sequents in *LJF*: the $\Uparrow$-sequents $\Gamma \Uparrow \Theta \vdash \Delta \Uparrow \Delta'$ and the $\Downarrow$-sequents $\Gamma \Downarrow \Theta \vdash \Delta \Downarrow \Delta'$. Here, all four *zones* Γ, Θ, Δ, and Δ' are multisets of formulas. Given that we are in an intuitionistic proof system, we require that the multiset union $\Delta \cup \Delta'$ is always a singleton. The zones Γ and Δ' are called the left and right *storage zones*. The zones Θ and Δ are called the left and right *staging areas*. As these names suggest, formulas in the storage zones can persist during the construction of a proof, while formulas in the staging area are intended to have a limited impact on the long-term construction of such a proof. Occurrences of logical connectives introduced by the left and right introduction rules only appear in the staging area. As we shall see, the only sequents with empty staging areas are of the form $\Gamma \Uparrow \cdot \vdash \cdot \Uparrow \Delta$. Such sequents are called *border sequents*: when we define synthetic inference rules in Definition 6, these sequents form the borders (the conclusion and premises) of synthetic inference rules.

Notational conventions: We usually denote an empty zone by explicitly using the dot $\cdot$. Also, while every *LJF* sequent has two occurrences of either $\Uparrow$ or $\Downarrow$, we write fewer of these arrows by adopting the convention that we drop writing $\cdot \Downarrow$ and $\cdot \Uparrow$ when they appear on the right, and we drop writing $\Downarrow \cdot$ and $\Uparrow \cdot$ when they appear on the left. Finally, since the right side of sequents have exactly one formula, we replace writing $\Downarrow \cdot$ with $\Downarrow$ and $\Uparrow \cdot$ with $\Uparrow$. Thus, the sequent $\Gamma \Uparrow \cdot \vdash \cdot \Uparrow E$ can be written as $\Gamma \vdash E$ and the sequent $\Gamma \Downarrow \cdot \vdash E \Downarrow \cdot$ can be written as $\Gamma \vdash E \Downarrow$. As a result of these conventions, border sequents in *LJF* will resemble sequents in *LJ*, which is a completely desirable resemblance.

In the general setting, Gentzen's *LJ* proof system involves *unpolarized* formulas, while the *LJF* focused proof system involves *polarized formulas*. Since, in this paper, we are only interested in one logical connective, the implication $\supset$, and since the polarization of an

Decide, Release, and Store Rules

$$\frac{N, \Gamma \Downarrow N \vdash A}{N, \Gamma \vdash A}\, D_l \qquad \frac{\Gamma \vdash P \Downarrow}{\Gamma \vdash P}\, D_r \qquad \frac{\Gamma \Uparrow P \vdash A}{\Gamma \Downarrow P \vdash A}\, R_l \qquad \frac{\Gamma \vdash N \Uparrow}{\Gamma \vdash N \Downarrow}\, R_r \qquad \frac{\Gamma, C \Uparrow \Theta \vdash \Delta' \Uparrow \Delta}{\Gamma \Uparrow \Theta, C \vdash \Delta' \Uparrow \Delta}\, S_l \qquad \frac{\Gamma \Uparrow \Theta \vdash A}{\Gamma \Uparrow \Theta \vdash A \Uparrow}\, S_r$$

Initial Rules

$$\frac{\delta(A) = +}{A, \Gamma \vdash A \Downarrow}\, I_r \qquad \frac{\delta(A) = -}{\Gamma \Downarrow A \vdash A}\, I_l$$

Introduction Rules for implication

$$\frac{\Gamma \vdash B \Downarrow \quad \Gamma \Downarrow B' \vdash A}{\Gamma \Downarrow B \supset B' \vdash A}\, \supset L \qquad \frac{\Gamma \Uparrow \Theta, B \vdash B' \Uparrow}{\Gamma \Uparrow \Theta \vdash B \supset B' \Uparrow}\, \supset R$$

Figure 1 The rules of (cut-free) *LJF* for the implicational fragment of propositional intuitionistic logic.

implication is unambiguous (it is *negative*), the only distinction between unpolarized and polarized formulas falls on atomic formulas: in polarized formulas, we need to describe the polarity of atomic formulas explicitly. (We remind the reader that a logical connective is polarized negatively if its right introduction rule is invertible.)

▶ **Definition 1.** *An* atomic bias assignment *is a function, δ, that maps atomic formulas to either $+$ or $-$. A formula B is* negative *if it is either an implication or atomic and $\delta(B) = -$. A formula B is* positive *if it is atomic and $\delta(B) = +$.*

The *LJF* proof system for intuitionistic propositional logic over just implication is given in Figure 1. This figure uses the following schema variables: P is a positive formula, N is a negative formula, A is an atomic formula, and B, B', and C denote arbitrary formulas. Note that in our simplified setting, a positive formula is also atomic.

An $\Uparrow$-*phase* is a collection of occurrences of $\Uparrow$-sequents that are all connected via inference rules; similarly, a $\Downarrow$-*phase* is a collection of occurrences of $\Downarrow$-sequents that are all connected via inference rules. The *decide* rules D_l and D_r are the only inference rules (in a cut-free proof) that have a border sequent as a conclusion. Thus, the decide rules sit on top of an $\Uparrow$-phase while their premises are at the bottom of a $\Downarrow$-phase. The *store* rules S_l and S_r work within an $\Uparrow$-phase. The release rules R_l and R_r are dual to the decide rules in that they sit on top of a $\Downarrow$-phase while their premises are at the bottom of an $\Uparrow$-phase.

When searching for a proof in *LJF*, the only occurrence of *don't know nondeterminism* occurs with the decide rules. There is a possible choice to decide on the right or left (D_l or D_r), and if D_l is selected, then another important choice is which formula in the left storage should be selected.

The following soundness and completeness theorem for *LJF* can be found in [24], where these theorems are proved for full first-order intuitionistic logic. (That paper also proves that *LJF* captures the structure of *LJT* and *LJQ* as well as some hybrid forms of those two proof systems.)

▶ **Theorem 2.** *Let B be an unpolarized formula composed only of implications and atomic formulas.*

1. *If B is provable in LJ and if $\delta(\cdot)$ is any atomic bias assignment for the atoms in B, then the sequent $\cdot \Uparrow \cdot \vdash B \Uparrow \cdot$ is provable in LJF.*
2. *If $\delta(\cdot)$ is an atomic bias assignment for the atoms in B and if $\cdot \Uparrow \cdot \vdash B \Uparrow \cdot$ is provable in LJF then B is provable in LJ.*

The theorem above implies that polarization of atomic formulas does not affect *provability* in *LJF*: in particular, if $\cdot \Uparrow \cdot \vdash B \Uparrow \cdot$ is provable for some atomic bias assignment then that sequent is provable for all such polarizations. On the other hand, different choices of atomic

bias assignments can make a big difference in the shape and size of *LJF* proofs. We illustrate this difference in the next section. Before doing that, we define the *order of a formula* and state two technical results about the order of formulas within *LJF* proofs.

▶ **Definition 3** (Order of a formula). *The* order *of the formula B, written $ord(B)$, is defined as follows: $ord(A) = 0$ if A is atomic and $ord(B_1 \supset B_2) = \max(ord(B_1) + 1, ord(B_2))$.*

Note that if we claim that all formulas in a multiset must have an order that is less than 0, then that multiset must necessarily be empty. The following proposition is proved by a straightforward induction on the structure of formulas.

▶ **Proposition 4.** Let B be a formula such that $ord(B) \leq n$. There is an $\Uparrow$-phase that has $\cdot \Uparrow \cdot \vdash B \Uparrow \cdot$ as its conclusion and has premises that are border sequents. Those border sequents are of the form $\Gamma \Uparrow \cdot \vdash \cdot \Uparrow A$, where A is atomic (i.e., $ord(A) = 0$) and the formulas in Γ have order less than or equal to $n - 1$.

The following proposition is proved by a simple induction on the structure of proofs.

▶ **Proposition 5.** Let Ξ be a (cut-free) *LJF* proof of the border sequent $\Gamma \Uparrow \cdot \vdash \cdot \Uparrow A$, where A is atomic and the formulas in Γ have order less than or equal to n. Then every border sequent in Ξ is of the form $\Gamma, \Delta \Uparrow \cdot \vdash \cdot \Uparrow A'$ where A' is an atomic formula and all formulas in Δ are of order less than or equal to $n - 2$.

2.2 Synthetic inference rules

The following definitions are based on similar definitions in [27].

▶ **Definition 6** (Synthetic inference rule). *A* left synthetic inference rule *is a rule of the form*

$$\frac{\Gamma_1 \vdash \Delta_1 \quad \ldots \quad \Gamma_n \vdash \Delta_n}{\Gamma \vdash \Delta}\ B. \qquad \textit{This rule is justified by a derivation of the form} \qquad \frac{\begin{array}{c}\Gamma_1 \Uparrow \cdot \vdash \cdot \Uparrow \Delta_1 \quad \ldots \quad \Gamma_n \Uparrow \cdot \vdash \cdot \Uparrow \Delta_n. \\ \Pi \\ \Gamma \Downarrow B \vdash \Delta\end{array}}{\Gamma \Uparrow \cdot \vdash \cdot \Uparrow \Delta}\ D_l$$

Here, $B \in \Gamma$, $n \geq 0$, and within Π, a $\Downarrow$-sequent never occurs above an $\Uparrow$-sequent. The structure of LJF proofs also forces $B \in \Gamma_i$ for all $1 \leq i \leq n$. Given our use of only implications, it is the case that there is a unique left synthetic inference rule for a given formula. The formula B can be used to name this left synthetic inference rule, and we say that this is the left synthetic inference rule for B. *We can similarly define the notion of* right synthetic inference rule*: however, in our context where the only positive formulas that can be used in a D_r rule are atomic, the only inference rule that can be applied to the premise of the D_r rule is I_r. As a result, right synthetic inference rules in our setting have zero premises and appear only at the leaves of proofs. We often write just* synthetic inference rule *to mean the left variant.*

▶ **Definition 7** (Bipole). *A* bipole *is a (left) synthetic inference rule in which all formulas stored using the store rules within the derivation justifying this synthetic inference rule are atomic formulas.*

Bipoles are, therefore, synthetic inference rules in which the only difference between the concluding sequent and any one of its premises is the presence or absence of atomic formulas. Note that, since we are only admitting implications, the synthetic rule for B is a bipole if and only if $ord(B) \leq 2$.

The primary use of the *LJF* proof system in this paper is to build large-scale, synthetic rules that we can add to the unpolarized proof system *LJ*: focusing gives us a framework to create such rules from Gentzen's micro rules. We define such an extension of *LJ* below and then illustrate it with two examples.

▶ **Definition 8** (Rules from polarized theory). *Let $\mathcal{T}$ be a finite set of formulas of order two or less, and let δ be an atomic bias assignment. We define $LJ\lfloor\delta,\mathcal{T}\rfloor$ to be the two-sided proof system built as follows. The only sequents in the $LJ\lfloor\delta,\mathcal{T}\rfloor$ proof system are of the form $\Gamma \vdash A$ where A is atomic and Γ is a multiset of atomic formulas. The inference rules of $LJ\lfloor\delta,\mathcal{T}\rfloor$ correspond to synthetic inference rules in the following way. For every left synthetic inference rule*

$$\frac{\mathcal{T},\Gamma_1 \vdash A_1 \quad \ldots \quad \mathcal{T},\Gamma_n \vdash A_n}{\mathcal{T},\Gamma \vdash A}\, B,$$

where $B \in \mathcal{T}$ is a negative formula, then the rule

$$\frac{\Gamma_1 \vdash A_1 \quad \ldots \quad \Gamma_n \vdash A_n}{\Gamma \vdash A}\, B$$

is added to $LJ\lfloor\delta,\mathcal{T}\rfloor$. The right synthetic inference rules are of the form

$$\frac{}{\Gamma \vdash A}$$ *provided $A \in \mathcal{T}$ and $\delta(A) = +$,*

and these are added as well. The only other inference rule added to $LJ\lfloor\delta,\mathcal{T}\rfloor$ is the rule

$$\frac{}{\Gamma, A \vdash A}\ \mathit{init}$$

where A is atomic. If $\delta(A) = -$, this rule is justified by the D_l rule, while if $\delta(A) = +$, this rule is justified by the D_r rule.

▶ **Example 9.** Let $n \geq 0$ and let $a_0, a_1, \ldots, a_n$ be a sequence of distinct atomic (propositional) formulas. Let $\mathcal{T}$ be the multiset of formulas $\{d_0, \ldots, d_n\}$, where d_0 is a_0, d_1 is $a_0 \supset a_1$, and so on until d_n is $a_0 \supset \cdots \supset a_{n-1} \supset a_n$. Let $\delta^-(\cdot)$ be the atomic bias assignment that gives all atomic formulas the negative polarity and let $\delta^+(\cdot)$ be the atomic bias assignment that gives all atomic formulas the positive polarity. The inference rules in $LJ\lfloor\delta^-,\mathcal{T}\rfloor$ contains the *init* rule along with the following $n+1$ rules

$$\frac{}{\Gamma \vdash a_0}\, d_0 \qquad \frac{\Gamma \vdash a_0}{\Gamma \vdash a_1}\, d_1 \qquad \frac{\Gamma \vdash a_0 \quad \Gamma \vdash a_1}{\Gamma \vdash a_2}\, d_2 \qquad \cdots \qquad \frac{\Gamma \vdash a_0 \quad \cdots \quad \Gamma \vdash a_{n-1}}{\Gamma \vdash a_n}\, d_n.$$

Given these inference rules, there is a *unique* proof of $\vdash a_n$, and that proof has 2^n occurrences of these inference rules. The negative bias assignment to atomic formulas yields the *backchaining* interpretation of these formulas. In contrast, the inference rules in $LJ\lfloor\delta^+,\mathcal{T}\rfloor$ contains the *init* rule along with the following n rules

$$\frac{\Gamma, a_0, a_1 \vdash A}{\Gamma, a_0 \vdash A}\, d_1 \qquad \frac{\Gamma, a_0, a_1, a_2 \vdash A}{\Gamma, a_0, a_1 \vdash A}\, d_2 \qquad \cdots \qquad \frac{\Gamma, a_0, \ldots, a_{n-1}, a_n \vdash A}{\Gamma, a_0, \ldots, a_{n-1} \vdash A}\, d_n.$$

Given these inference rules, it is easy to note that there are an infinite number of proofs of $a_0 \vdash a_n$, and that the shortest of these proofs contains n occurrences of synthetic inference rules plus one occurrence of the initial rule. The positive bias assignment to atomic formulas yields the *forward-chaining* interpretation of these clauses.

There are several proofs in the literature that show how cut can be eliminated within focusing proofs. Some proofs, such as the one given by Liang & Miller in [24, 26], introduce different variants of the cut rule and follow a rather tedious argument detailing how these cut

rules move through individual inference rules within ⇑ and ⇓ phases. Bruscoli & Guglielmi [4] provided a different style of proof of cut elimination in a focused proof system for linear logic in which they showed how cuts could move through entire phases at a time. Other phase-based cut-elimination proofs appear in [6, 25, 35, 41, 42]. Part II of Graham-Lengrand's HdR dissertation [20] and Simmons's article [37] provide overviews of such cut-elimination proofs. While none of the proofs mentioned above deal directly with *synthetic* rules in the sense that we have defined them here, the paper [27] does have a cut-elimination theorem that directly deals with eliminating cuts in proofs with synthetic rules. The following theorem follows directly from [27, Theorem 16]. It can also be proved directly by simple induction. Let the *atomic cut rule* be the rule

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B}\ \mathit{Acut},$$

where A is atomic. When we add this rule to $LJ\lfloor\delta, \mathcal{T}\rfloor$, then B is also atomic.

▶ **Theorem 10** (Cut admissibility for $LJ\lfloor\delta, \mathcal{T}\rfloor$). *Let $\mathcal{T}$ be a set of formulas of order two or less and let $\delta(\cdot)$ be an atomic bias assignment. The atomic cut rule is admissible for the proof system $LJ\lfloor\delta, \mathcal{T}\rfloor$.*

3 Synthetic inference rules as term constructors

Many approaches to relating proofs and terms start with the assumption that terms have been defined, say, as λ-term, and that the notion of proof is derived from terms, e.g., proofs are often identified as dependently typed λ-terms [21]. As we motivated in the introduction, we plan to reverse this influence by starting with proof structures as given and then deriving term structures from them. In particular, we shall map synthetic inference rules directly to the constructors used to build terms.

The first step in translating proofs to terms is to annotate sequents properly. We shall annotate assumptions (i.e., formulas on the left-hand side of sequents) with variables (i.e., token of type *var*) and annotate the conclusion (i.e., the formula on the right-hand side of sequents) with a term of type *tm*. The term constructor that corresponds to the *init* rule (Definition 8) is written $\lceil\cdot\rceil$ and has syntactic type $\mathit{var} \to \mathit{tm}$. The corresponding annotated inference rule is simply

$$\frac{}{\Gamma \vdash \lceil x \rceil : A}\ \mathit{init}, \text{ provided that } x : A \in \Gamma.$$

The following example illustrates how the proofs in Example 9 can be seen as terms.

▶ **Example 11.** Consider changing the naming of the formulas used in Example 9 into typing assumptions, as follows:

$$d_0 : a_0, \quad d_1 : a_0 \supset a_1, \quad \ldots, \quad d_n : a_0 \supset \cdots \supset a_{n-1} \supset a_n.$$

Here, $d_0, \ldots, d_n$ are all of syntactic type *var*. Under the δ^- atomic bias assignment, the synthetic rules can be annotations as

$$\frac{\Gamma \vdash t_0 : a_0 \quad \cdots \quad \Gamma \vdash t_{i-1} : a_{i-1}}{\Gamma \vdash (E_i\ t_0\ \cdots t_{i-1}) : a_i}\ d_i,$$

where $0 \le i \le n$. The syntactic type of constructor E_i is $\mathit{tm} \to \cdots \to \mathit{tm} \to \mathit{tm}$ (where *tm* occurs $i+1$ times). On the other hand, if all atomic formulas are polarized positively, then the annotated synthetic rules are

$$\frac{\Gamma,\ x_0 : a_0,\ \ldots,\ x_{i-1} : a_{i-1},\ y : a_i \vdash t : A}{\Gamma,\ x_0 : a_0,\ \ldots,\ x_{i-1} : a_{i-1} \vdash (F_i\ x_0\ \ldots\ x_{i-1}\ (\lambda y.t)) : A} \quad \text{(provided } y \text{ is new)},$$

where $i \geq 1$. The syntactic type of F_i $(i \geq 1)$ is $var \to \cdots \to var \to (var \to tm) \to tm$ (where var occurs $i+1$ times). Below, we display the unique proof of a_4 using the E_n constructors and the shortest proof of a_4 using the F_n constructors: the structure on the right allows for explicit sharing of subterms while the structure on the left must repeat these subterms.

$$(E_4\ (E_3\ (E_2\ (E_1\ E_0)\ (E_1\ E_0))\ (E_2\ (E_1\ E_0)\ (E_1\ E_0)))\\ \quad (E_3\ (E_2\ (E_1\ E_0)\ (E_1\ E_0))\ (E_2\ (E_1\ E_0)\ (E_1\ E_0))))$$

$$\begin{array}{ll}(F_1\ d_0 & (\lambda x_1.\\ (F_2\ d_0\ x_1 & (\lambda x_2.\\ (F_3\ d_0\ x_1\ x_2 & (\lambda x_3.\\ (F_4\ d_0\ x_1\ x_2\ x_3 & (\lambda x_4.\ \lceil x_4 \rceil)))))))\end{array}$$

More generally, we can describe the structure of synthetic inference rules as follows. In order to compute the left synthetic inference rule that corresponds to using D_l on the indexed formula $f : B$, where $ord(B) \leq 2$, we must know how the atomic formulas in B are polarized. In what follows, we assume that there are exactly two atomic bias assignments, namely, δ^- and δ^+. We consider these two cases separately. In the following discussion, we use Δ as a schematic variable to range over multisets of atomic formulas and use $\bar{w}$ to denote a list of variables, say, $w_1, \ldots, w_n$ for some $n \geq 0$. The notation $\bar{w} : \Delta$ denote a type assignment, say, $w_1 : A_1, \ldots, w_n : A_n$, where $\bar{w}$ is $w_1, \ldots, w_n$ and Δ is $A_1, \ldots, A_n$. Finally, whenever we construct typing assignments for the left-hand side of sequents, we will always assume that the names assigned types are all distinct.

Consider first the case that we are using δ^-. Any formula B such that $ord(B) \leq 2$ can be written as

$$(\Delta_1 \supset A_1) \supset \cdots \supset (\Delta_m \supset A_m) \supset A_0 \qquad (m \geq 0)$$

where $\Delta_1, \ldots, \Delta_m$ are multisets of atomic formulas. (If Δ is empty then $\Delta \supset B$ is simply B.) If the displayed formula above has order 0 then $m = 0$. If that formula has order 1 then $m \geq 1$ and $\Delta_1, \ldots, \Delta_m$ are empty. If that formula has order 2 then $m \geq 1$ and at least one of $\Delta_1, \ldots, \Delta_m$ is nonempty. In any case, all of the formulas $\Delta_1 \supset A_1, \ldots, \Delta_m \supset A_m$ have negative polarity. In this case, the left synthetic inference rule corresponding to B can be written as

$$\frac{\Gamma, \bar{w}_1 : \Delta_1 \vdash t_1 : A_1 \quad \ldots \quad \Gamma, \bar{w}_m : \Delta_m \vdash t_m : A_m}{\Gamma \vdash (f\ (\lambda \bar{w}_1.t_1) \ldots (\lambda \bar{w}_m.t_m)) : A_0}$$

By $\bar{w} : \{D_1, \ldots, D_n\}$ we mean $w_1 : D_1, \ldots, w_n : D_n$, assuming that all the tokens $w_1, \ldots, w_n$ are all distinct and do not occur in Γ. If the list of variables $\bar{w}$ is empty, then we abbreviate $\lambda \bar{w}.t$ as simply t.

Now consider the second case where we are using δ^+. In this case, we note that any formula B such that $ord(B) \leq 2$ can be written as

$$\Delta_0 \supset (\Delta_1 \supset A_1) \supset \cdots \supset (\Delta_m \supset A_m) \supset A_0 \qquad (m \geq 0)$$

(We assume that all the atomic argument types are listed before the non-atomic argument types.) Here, $\Delta_0, \ldots, \Delta_m$ are all multisets of atomic formulas, and we assume that $\Delta_1, \ldots, \Delta_m$ are nonempty. If the displayed formula above has order 0 then $m = 0$ and Δ_0 is empty. If that formula has order 1 then $m = 0$ and Δ_0 is nonempty. If that formula has order 2 then $m \geq 1$. In this way of presenting the type of f, the (atomic) argument types in Δ_0 all have positive polarity, while the remaining argument types all have negative polarity. Thus, the left synthetic inference rule corresponding to B can be written as

$$\frac{\Gamma, \bar{w}_1 : \Delta_1 \vdash t_1 : A_1 \quad \ldots \quad \Gamma, \bar{w}_m : \Delta_m \vdash t_m : A_m \quad \Gamma, y : A_0 \vdash t : A}{\Gamma \vdash \mathbf{name}\ y = (f\ \bar{u}\ (\lambda \bar{w}_1.t_1) \ldots (\lambda \bar{w}_m.t_m))\ \mathbf{in}\ t : A} \text{ provided } \bar{u} : \Delta_0 \subseteq \Gamma$$

The proviso to this rule means that every type assumption $u_1 : D_1, \ldots, u_n : D_n$ is a member of Γ (here, $\bar{u}$ is $u_1, \ldots, u_n$ and Δ_0 is $\{D_1, \ldots, D_n\}$). We do not assume that the variables in $u_1, \ldots, u_n$ are distinct. The first m premises above construct abstracted terms, and the last premise continues to construct a term for type A but this time with an additional variable y that names the structure $(f\ \bar{u}\ (\lambda\bar{w}_1.t_1) \ldots (\lambda\bar{w}_m.t_m))$. Note also that the first arguments of f (written here as $\bar{u}$) must already be present on the left-hand side of the conclusion.

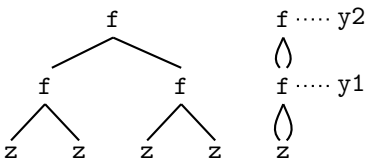
▶ **Example 12.** Using this notation, the last term in Example 11 can be written as follows.

name $x_1 = (F_1\ d_0)$ **in**
name $x_2 = (F_2\ d_0\ x_1)$ **in**
name $x_3 = (F_3\ d_0\ x_1\ x_2)$ **in**
name $x_4 = (F_4\ d_0\ x_1\ x_2\ x_3)$ **in** $\lceil x_4 \rceil$

To provide a simple, graphical illustration of how atomic bias assignments can affect term structure, consider the typing assignment $\{z : i,\ f : i \supset i \supset i\}$. The following two terms denote proofs built with synthetic inference rules in which i has a negative and positive polarity, respectively.

```
(f (f z z) (f z z))
name y1 = (f z z) in name y2 = (f y1 y1) in y2.
```

These terms could also be displayed as the following labeled tree and DAG. Here, we display the tree associated with a term with its root at the top and its leaves below: this is, of course, the opposite convention to how proof trees are commonly displayed.



In an appendix of the extended version of this paper [32], we show how these constructors – corresponding to synthetic inference rules – can be implemented using the $\lambda\kappa$-calculus, a term representation for *LJF* introduced by Brock-Nannestad et al. in [3].

In the following subsections, we note two applications of the term representations that we have described above.

3.1 Intermediate representation of programs

The `name` expressions used above resemble the more common `let` expressions, but they are different in at least two important ways. First, the let-expression "let $x = r$ in t" is often considered to be a non-normal term since it sometimes abbreviates the β-redex $((\lambda x.t)r)$. In contrast, the term "name $x = r$ in t" is normal if t is normal. (Note that the synthetic inference rules are built without the use of the cut inference rule.) Second, the term r in a let-expression can be an arbitrary term, whereas in the name-expression, the term r must have a particular structure: when we are considering only first-order signatures, r must be the application of a constructor to one or more variables (and not to general terms).

Given an expression E using negative bias syntax, there might be a subexpression, say E', that has many occurrences in E. If we let $F(x)$ denote the result of replacing every occurrence of E' in E with the variable x, then the expression $let\ x = E'\ in\ F(x)$ might be

a more appropriate presentation of E in which the subformula E' is named and explicitly shared. Such an operation in often called *common subexpression elimination*. The positive bias syntax that we have described here is orthogonal to this processing in the sense that the positive bias assignment syntax forces *all* function applications (not just those that are repeated) to be named while there is no guarantee that naming is not redundant (i.e., a given subterm might be given two names). Of course, if one takes care in building terms using positive bias assignment, it is possible to build terms where common subexpressions are explicitly shared.

A useful intermediate representation of programs in compilers of functional programming languages is the *administrative normal form (ANF)* [13] in which all arguments to functions are values, that is, either constants, variables, or λ-abstractions. Clearly, when we are using positive bias assignment syntax, the expressions that result are in ANF.

Various other term representations have been developed for focused proofs that contain more logical connectives and inference rules than we have considered here. See, for example, [5, 23, 37]. However, atomic formulas are given the negative polarity in these references.

3.2 Encoding functional expressions as relational queries

When a Prolog programmer needs to compute the value of a mathematical expression, such as $\sqrt{b^2 - 4ac}$, it is necessary to explicitly convert the calls to various functions (here, subtraction, addition, multiplication, and square root) into associated relations. For example, addition on real numbers is usually represented by the three-place predicate *plus* such that the atomic formula $(plus\ x\ y\ z)$ holds if and only if $x + y = z$. Now assume that relations are available to encode each primitive function. One way to organize the relations needed to compute the expression above involves converting that expression into positive bias syntax. For example, the function-based expression above can be written as

$$\textbf{name}\ n_1 = b \times b\ \textbf{in name}\ n_2 = 4 \times a\ \textbf{in name}\ n_3 = n_2 \times c\ \textbf{in}$$
$$\textbf{name}\ n_4 = n_1 - n_3\ \textbf{in name}\ n_5 = \sqrt{n_4}\ \textbf{in}\ \lceil n_5 \rceil.$$

As described in [16], it is straightforward to convert such an expression into a series of calls to predicates. In particular, we can rewrite this expression by replacing $[\textbf{name}\ n = f\ x_1\ \cdots\ x_i\ \textbf{in}\ \bullet]$ with $[\exists n.(R_f\ x_1\ \cdots\ x_i\ n) \wedge \bullet]$, where R_f is a relation that computes the function f. Assuming that *times*, *minus*, and *sqrt* are all relations that compute multiplication, subtraction, and the (positive) square root, then the relational presentation can be given as

$$\exists n_1.\, times\ b\ b\ n_1 \wedge \exists n_2.\, times\ 4\ a\ n_2 \wedge \exists n_3.\, times\ n_2\ c\ n_3 \wedge \exists n_4.\, minus\ n_1\ n_3\ n_4 \wedge \exists n_5.\, sqrt\ n_4\ n_5,$$

which is an expression that is easily written as a Prolog goal formula.

4 The untyped lambda-calculus

Let D be an atomic formula and let Γ_0 be the theory $\{(D \supset D) \supset D,\ D \supset (D \supset D)\}$. This theory is *inconsistent* in that every formula built from implications and D is provable from Γ_0. We choose to consider Γ_0 because cut-free proofs in *LJF* of $\Gamma_0 \vdash D$ correspond to closed untyped λ-terms. The following derivations result from applying D_l to these two formulas (assuming that $\Gamma_0 \subseteq \Gamma$).

$$\dfrac{\dfrac{\Xi_1 \quad \Xi_2 \quad \Xi_3}{\Gamma \Downarrow D \supset (D \supset D) \vdash D}\ \supset L \times 2}{\Gamma \vdash D}\ D_l \qquad \dfrac{\dfrac{\dfrac{\dfrac{\Xi_4}{\Gamma, D \vdash D}}{\Gamma \Uparrow D \vdash D \Uparrow}\ S_l, S_r}{\Gamma \vdash D \supset D \Uparrow}\ R_r, \supset R \quad \dfrac{\Xi_5}{\Gamma \Downarrow D \vdash D}}{\Gamma \Downarrow (D \supset D) \supset D \vdash D}\ \supset L}{\Gamma \vdash D}\ D_l$$

These derivations can only be extended to have border sequents as premises if we reveal the polarity assigned to D. If D is polarized negatively, then Ξ_1 and Ξ_2 are both R_r while Ξ_3 and Ξ_5 are I_l. In this case, the resulting synthetic inference rules are

$$\frac{\Gamma \vdash D \qquad \Gamma \vdash D}{\Gamma \vdash D} \quad \text{and} \quad \frac{\Gamma, D \vdash D}{\Gamma \vdash D} \ .$$

On the other hand, if D is polarized positively, then Ξ_1 and Ξ_2 are both I_r while Ξ_3 and Ξ_5 is R_l. In this case, the resulting synthetic inference rules are

$$\frac{\Gamma, D, D, D \vdash D}{\Gamma, D, D \vdash D} \quad \text{and} \quad \frac{\Gamma, D \vdash D \qquad \Gamma, D \vdash D}{\Gamma \vdash D} \ .$$

Without annotations, these inference rules are not illuminating. We provide such annotations next.

4.1 Using negative bias syntax for the untyped lambda-calculus

If we polarize D negatively, the (annotated) synthetic inference rules based on Γ_0 yield the standard tree-like representation for the untyped λ-calculus. In particular, consider an annotated LJF proof of a sequent of the form $\Gamma_0, x_1 : D, \ldots, x_n : D \vdash t : D$. The synthetic inference rules in such a proof are either the result of deciding on $x_i : D$ (for $1 \leq i \leq n$) followed by I_l or on one of the two formulas in Γ_0. The three inference rules annotate these three choices for using the D_l rule.

$$\frac{\Gamma \vdash t : D \qquad \Gamma \vdash s : D}{\Gamma \vdash \mathit{napp}\ t\ s : D} \qquad \frac{\Gamma, x : D \vdash t : D}{\Gamma \vdash \mathit{nabs}\ (\lambda x.t) : D} \qquad \frac{}{\Gamma \vdash \mathit{nvar}\ x_i : D} \text{ provided } x_i : D \in \Gamma$$

The constructors represented by these synthetic rules are listed below, along with their syntactic types (using typing declarations from λProlog [29]).

```
type napp   tm -> tm -> tm.
type nabs   (var -> tm) -> tm.
type nvar   var -> tm.
```

As we have assumed before, `var` and `tm` are primitive types: annotations on the left side of sequents have type `var` while annotations on the right side have type `tm`. The prefix `n` on these names is meant to remind us that we assigned D the negative polarity. As an example of using these constructors, the untyped λ-term $((\lambda x.xx)(\lambda x.xx))$ can be written as the following term of type tm:[1]

```
(napp (nabs x\ napp (nvar x) (nvar x))
      (nabs x\ napp (nvar x) (nvar x))).
```

Many computational logic systems, such as λProlog, Abella, LF, and Isabelle, encode the untyped λ-calculus in this fashion.

4.2 Using positive bias syntax for the untyped lambda-calculus

If we polarize D positively, we get different synthetic rules based on using Γ_0 and, hence, we get a different format for encoding the untyped λ-calculus. Again, there are exactly three synthetic inference rules for LJF proofs of sequents of the form $\Gamma_0, x_1 : D, \ldots, x_n : D \Uparrow \cdot \vdash \cdot \Uparrow t : D$, but this time there are two left synthetic inference rules and one right synthetic inference rules.

[1] The λ-abstraction of λProlog is written using an infix backslash.

$$\frac{\Gamma, y : D \vdash t : D}{\Gamma \vdash \mathit{papp}\ x_i\ x_j\ (\lambda y.t) : D} \text{ provided } \Gamma \text{ contains } x_i : D \text{ and } x_j : D$$

$$\frac{\Gamma, x : D \vdash t : D \qquad \Gamma, y : D \vdash s : D}{\Gamma \vdash \mathit{pabs}\ (\lambda x.t)\ (\lambda y.s) : D} \qquad \frac{}{\Gamma \vdash \mathit{pvar}\ x_i : D} \text{ provided } x_i : D \in \Gamma$$

The constructors represented by these synthetic rules can be given the following syntactic typing.

```
type papp  var -> var -> (var -> tm) -> tm.
type pabs  (var -> tm) -> (var -> tm) -> tm.
type pvar  var -> tm.
```

The prefix `p` on these names is meant to remind us that, in this case, we assigned D the positive polarity. Using these constructors, the untyped λ-term $((\lambda x.xx)(\lambda x.xx))$ can be written as the term

```
(pabs (x\ papp x x (y\ pvar y)) (u\ papp u u (z\ pvar z)))
```

Note that this untyped λ-term contains two occurrences of `papp` while in the previous representation, this term contains three occurrences of `napp`.

In order to make terms in this syntax easier to read, we will often use the `name`-expressions presented before. In particular, the expression `(papp u v (w\ Body))`, which denotes the application of the variable `u` to the variable `v`, and then *naming* that application as `w` in the scope of `Body`. Thus, this expression can also be written using the expression `name w = (app u v) in Body`. Similarly, the expression `(pabs R (w\ Body))` can be written as `name w = (abs R) in Body`. If we used this syntax, then the expression above denoting $((\lambda x.xx)(\lambda x.xx))$ is written as

```
  name u = (abs x\ name y = (app x x) in y) in
  name z = (app u u) in z
```

That is, `u` is used to name the encoding for the term $(\lambda x.xx)$, and then that name is used twice in building the final application that is named `z`. Extending this example slightly, we see that the expression $(((\lambda x.xx)(\lambda x.xx))(\lambda x.xx))$ can be written as

```
(pabs (x\ papp x x (y\ pvar y)) (u\ papp u u (z\ papp z u v\ pvar
    v)))
```

or, using the `name` syntax, as

```
  name u = (abs x\ name y = (app x x) in y) in
  name z = (app u u) in
  name v = (app z u) in v.
```

As this alternative syntax suggests, the syntax that results from making the primitive type D positive makes sharing explicit by its requirement that all applications are built from *named* structures and that that application is named itself. It is also clear that the following two expressions denote the same untyped λ-term.

```
  name u  = (abs x\ name y = (app x x) in y) in
  name z1 = (app u u) in
  name z2 = (app u u) in
  name v  = (app z1 u) in v
```

```
name u1 = (abs x\ name y = (app x x) in y) in
name z  = (app u1 u1) in
name u2 = (abs x\ name y = (app x x) in y) in
name v  = (app z u2) in v
```

The first of these terms illustrates that a named structure might not be used in its scope: we call this *vacuous naming*. The second of these terms illustrates that the same structure can be named twice: we call this *redundant naming*. The proof theory behind *LJF* allows for both vacuous and redundant naming: we currently see no proof-theoretic device that can cleanly eliminate these kinds of naming expressions.

This style of syntactic representation seems relatively low-level since it uses names to designate all constructors in a term. Such a representation of terms resembles, in fact, the use of pointers to encode terms in memory: a pointer (name) indicates a unit of memory that contains the name of a constructor followed by a vector of pointers to that constructor's arguments.

4.3 Tracing untyped lambda-terms

Given that we have two different formats for untyped λ-terms, it is a natural question whether or not two such expressions denote the same untyped λ-term. For example, it seems sensible to consider the last three expressions in Section 4.2 (based on positive bias assignment) as equivalent in some sense to each other and to the following expression (based on negative bias assignment).

```
(napp (napp (nabs x\ napp (nvar x) (nvar x))
            (nabs x\ napp (nvar x) (nvar x)))
      (nabs x\ napp (nvar x) (nvar x)))
```

Broadly speaking, there are two approaches to answering this question. The "white box" approach examines the actual syntax of proof expressions to see if they should be considered equal. For example, in the setting of natural deduction, two proofs are often considered equal if they reduce to the same normal form. Given that we are considering proofs built with difference sets of (synthetic) inference rules, a different approach needs to be taken, such as the approach described in [35], where proofs based on positive bias assignment to atomic formulas are systematically converted to proofs based on negative bias assignment.

Instead, we propose to use a "black box" approach in which we probe a term to describe *traces* within expressions. For example, we can ask whether or not the term denotes a top-level application or not. This check is easy to make for negative bias syntax by simply examining the top-level symbol of the expression. It is also easy to check for the expressions using the positive bias assignment: simply examine the top-level naming structure, say,

$$[\texttt{name}\ x_1 = E_1\ \texttt{in}\ \texttt{name}\ x_2 = E_2\ \texttt{in}\ \cdots\ \texttt{name}\ x_n = E_n\ \texttt{in}\ x_j]$$

and check if E_j is an application or not. If two expressions denote an application, we can continue to develop a trace by examining either the first or second argument of that application. Similarly, we can examine two expressions to see if they denote a λ-abstraction. If they are both λ-abstractions, then we can probe the body of those abstractions, taking appropriate care when descending under a binding.

A formal specification of such trace predicates is easy in a language such as λProlog. In particular, the following declarations define the datatype of traces through untyped λ-terms.

```
type ntrace, ptrace   tm -> trace -> o.
```

```
ntrace (napp M _) (left  P) :- ntrace M P.
ntrace (napp _ N) (right P) :- ntrace N P.
ntrace (nabs R)   (bnd   P) :- pi x\pi p\ ntrace (nvar x) p =>
                                          ntrace (R x) (P p).

ptrace (papp U V K) P :-
   pi x\ (pi P\ ptrace (pvar x) (left  P) :- ptrace (pvar U) P) =>
         (pi P\ ptrace (pvar x) (right P) :- ptrace (pvar V) P) =>
   ptrace (K x) P.
ptrace (pabs R K) P :-
   pi x\  (pi Q\ ptrace (pvar x) (bnd Q) :-
                 pi p\ pi u\ ptrace (pvar u) p => ptrace (R u) (Q
                     p))
          => ptrace (K x) P.
```

Figure 2 Traces through negative and positive bias syntax.

```
kind trace          type.
type left, right    trace -> trace.
type bnd            (trace -> trace) -> trace.
```

The specification of traces within both variants of expressions for (closed) untyped λ-terms is given in Figure 2. Note that the orders of the clauses for `ntrace` are 0, 1, 2 while for `ptrace` the orders are 0, 3, and 4.[2] If we wish to treat open expressions, we can add a constant, say `w`, to denote a free variable, along with the declarations

```
type w       var.
type wtrace  trace.
```

```
ntrace  (nvar w) wtrace.
ptrace  (pvar w) wtrace.
```

We say that two expressions denoting untyped λ-terms (using either positive or negative bias assignment) are *trace equivalent* if they both have the same traces.[3] It is easy to prove that two expressions using the negative bias syntax are trace equivalent if and only if they are α-equivalent.[4] This statement is not true for positive bias syntax: in particular, the examples at the end of Section 4.2 that illustrate vacuous and duplicate naming all have the same traces but are not α-convertible expressions.

We note that in λProlog, it is possible to synthesize an expression from a list of traces using, for example, a query such as

```
?- forall (ntrace T) [(bnd (u\ left (bnd (v\ left u)))),
                      (bnd (u\ left (bnd (v\ right v)))),
                      (bnd (u\ right u))].
T = nabs (u\ napp (nabs (v\ napp (nvar u) (nvar v))) (nvar u))
```

[2] Standard techniques can be used to rewrite the last two of these clauses to clauses of order 2 at the expense of adding new predicate constants. See Appendix A of [32] for such a specification.
[3] In concurrency theory, this notion is more often called *maximal trace equivalence*.
[4] See `http://abella-prover.org/examples/lambda-calculus/term-structure/path.html` for a short, formal proof of this claim in Abella.

```
Kind op       type.
Type app      val  -> val -> op.
Type abs      (val -> tm) -> op.

Kind pair     type.
Type pr       val -> op -> pair.

Kind node     type.
Type nd       list pair -> tm -> node.
```

Figure 3 An Abella specification of sharing simulation.

Here, `forall` is a higher-order predicate that applies its predicate argument (here, `(ntrace T)`) to all members in its second argument. A black box method of converting an expression using the positive bias assignment into an expression using negative bias assignment proceeds as follows: first, list all possible traces in the positive bias assignment expression, and second, synthesize the negative bias assignment expression using the technique illustrated above (see also [29, Section 7.4.2]).

4.4 Sharing bisimulation

Determining that two untyped λ-term expressions are trace equivalent by enumerating every trace in them has an exponential cost since all sharing structures are removed when listing traces. Condoluci et al. [9] developed a graphic representation of sharing in the untyped λ-calculus using *λgraphs*. When an appropriate bisimulation is defined on nodes in such λgraphs, it is possible to check the bisimilarity in such graphs in linear time. As is known from concurrency theory, bisimilarity implies (maximal) trace equivalence. In our setting, if two terms – represented as two nodes in a λgraph – are bisimilar, then those two terms are also trace equivalent.

To illustrate how one can manipulate positive bias syntax effectively, we used Abella [2] to specify a simulation relation (closely related to the bisimulation relation defined in [9]) that compares two expressions in such a way that unfolding of the sharing does not happen.

The top-level of an untyped λ-term expression based on the positive bias assignment for D can be described as a pair containing an association list of naming variables and operations (such as `app` and `abs` used above) and the name of a particular naming variable. For example, the last term displayed in Section 4.2 can be seen as a list of four pairs, namely,

$$[\ \langle \texttt{u1}, \texttt{(abs x name y = (app x x) in y)}\rangle,\ \langle \texttt{z}, \texttt{(app u1 u1)}\rangle,$$
$$\langle \texttt{u2}, \texttt{(abs x name y = (app x x) in y)}\rangle,\ \langle \texttt{v}, \texttt{(app z u2)}\rangle\ ].$$

along with the designation of one particular variable, here `v`. This presentation suggests encoding such terms using the Abella declarations (which are similar to λProlog declarations) found in Figure 3. For example, the example term displayed above can be written as the Abella term of type `name`

```
(nd ((pr n4 (app n2 n3)) ::
     (pr n3 (abs x\ papp x x y\ pvar y)) ::
     (pr n2 (app n1 n1)) ::
     (pr n1 (abs x\ papp x x y\ pvar y)) :: nil)
    (pvar n4))
```

(Here, the symbols `n1`, ..., `n4` are examples of nominal constants in Abella.) In this setting, expressions using positive bias syntax are encoded as terms of type `node`.

Structures of type `node` can be used to generate a labeled transition system in which some arcs are labeled. These labels, called *actions* here, are of the following three kinds (see the full specification in Figure 4).

1. Primitive actions are described using the predicate `paction`. Such actions name a variable.
2. Bound actions, specified using `baction`, carry an abstraction node to the body of that abstraction.
3. Application actions, specified using `faction`, carry an application node to either its left or right argument: this action names that direction.

The ∇-quantifier [14, 31] (written in Abella as `nabla`) is used to manage nominal constants and binding mobility [28]. The distinction between *free action* and *bound action* here is essentially the same as has been used to specify various simulations in the π-calculus [33]. Our specification of simulation in the presence of both free and bound actions follows the specification technique of Tiu & Miller [38] that also relied on the ∇-quantifier. Given our simulation specification, the bisimulation specification is also easy to write.

5 Cut elimination at the level of synthetic rules

Theorem 10 states that cut elimination holds for proofs built using synthetic inference rules. Our goal in this section is to use cut elimination to determine what substitution into terms should be. In particular, if we have the term t and we have the abstraction of x over term s, how do we compute the result of substituting t for x in s? Clearly, the answer will depend on which polarity assumption we are using for primitive types.

In order to see how cut-elimination can yield substitution, consider the following instance of the cut rule: here, E and E' are atomic formulas, and the LJF proofs Ξ_L and Ξ_R are cut-free.

$$\frac{\begin{array}{cc}\overset{\Xi_L}{\Gamma \vdash t : E'} & \overset{\Xi_R}{\Gamma, x : E' \vdash s : E}\end{array}}{\Gamma \vdash Cut_0(x.s, t) : E}\, Cut_0$$

While the term $Cut_0(x.s, t)$ is not a term, it denotes the result of substituting u for x in t. By performing cut-elimination on this proof, we will arrive at a cut-free proof and the term annotating that proof should denote the result of such a substitution.

We illustrate here how the cut-elimination procedure works on our two encodings of untyped λ-terms.[5] In particular, we will provide specifications (using λProlog code) to define the predicates

```
type nsubst, psubst    tm -> (var -> tm) -> tm -> o.
```

that have the following specification: given `S` and `T` of type `tm` and `R` of type `var -> tm` then `(nsubst T R S)` is provable if and only if `T`, `R`, and `S` use the constructors `napp`, `nabs`, and `nvar` and `S` is the result of substituting `T` into the bound variable of `R`. Similarly, we wish to have the same kind of specification for `(psubst T R S)` but with the arguments `S`, `T`, and `R` built using the constructors `papp`, `pabs`, and `pvar`.

[5] The cut-elimination procedure is given in Appendix B of [32].

```
Define paction : list val -> node -> val -> prop by
  paction Vs (nd C (pvar w)) w ;
  paction Vs (nd C (pvar V)) V := member V Vs.

Define baction : node -> (val -> node) -> prop by
  nabla n, baction (nd ((pr n (abs R)):: C)  (pvar n))
                   (u\ nd C (R u)) ;
  nabla n, baction (nd ((pr M (Op n)):: (C n)) (pvar n)) Nd :=
     nabla n, baction (nd (C n) (pvar n)) Nd.

Kind direction     type.
Type right, left   direction.

Define faction : node -> direction -> node -> prop by
  nabla n, faction (nd ((pr n (app U V)):: C) (pvar n)) right
                   (nd C (pvar V)) ;
  nabla n, faction (nd ((pr n (app U V)):: C) (pvar n)) left
                   (nd C (pvar U)) ;
  nabla n, faction (nd ((pr M (Op n)):: (C n)) (pvar n)) A T :=
     nabla n, faction (nd (C n) (pvar n)) A T.

Define sim  : list val -> node -> node -> prop,
       simm : list val -> node -> node -> prop by
 sim Vs (nd C (papp U V K)) Nd :=
    nabla n, sim Vs (nd ((pr n (app U V)):: C) (K n)) Nd ;
 sim Vs (nd C (pabs R K)) Nd :=
    nabla n, sim Vs (nd ((pr n (abs R)):: C) (K n)) Nd ;
 sim Vs (nd D (pvar T)) (nd C (papp U V K)) :=
    nabla n, sim Vs (nd D (pvar T)) (nd ((pr n (app U V)):: C) (K
       n)) ;
 sim Vs (nd D (pvar T)) (nd C (pabs R K)) :=
    nabla n, sim Vs (nd D (pvar T)) (nd ((pr n (abs R)):: C) (K
       n)) ;
 sim Vs (nd C (pvar U)) (nd D (pvar V)) :=
   simm Vs (nd C (pvar U)) (nd D (pvar V)) ;

 simm Vs P Q :=
  (forall N,   paction Vs P N -> paction Vs Q N)         /\
  (forall A R, faction P A R  -> exists S, faction Q A S /\
                                           sim Vs R S)   /\
  (forall R,   baction P R    -> exists S,
                                   baction Q S  /\
                                   nabla u, sim (u::Vs) (R u) (S
                                       u)).
```

Figure 4 An Abella specification of sharing simulation.

When terms are built using negative bias syntax, the cut always moves to the right branch, which means that substitution can be defined recursively on `R`. Moreover, substitution is applied to all the arguments of constructors recursively. Thus, we have the following λProlog specification of `nsubst`.

```
nsubst T (x\ nvar x) T.
nsubst T (x\ nvar Y) (nvar Y).
nsubst T (x\ napp (R x) (S x)) (napp R' S') :-
   nsubst T R R', nsubst T S S'.
nsubst T (x\ nabs y\ R x y) (nabs y\ R' y) :-
   pi y\ nsubst T (x\ R x y) (R' y).
```

Note that this substitution predicate moves recursively through its second (abstracted) argument. This style of substitution is, of course, the familiar one. Using the following functional equations, we can write this substitution operation as a postfix operator.

- $(\texttt{nvar}\ x)[x/t] = t$;
- $(\texttt{nvar}\ y)[x/t] = (\texttt{nvar}\ y)$, provided x and y are different;
- $(\texttt{napp}\ R\ S)[x/t] = (\texttt{napp}\ R[x/t]\ S[x/t])$, and
- $(\texttt{nabs}\ (\lambda y.R))[x/t] = (\texttt{nabs}\ \lambda y.\ (R[x/t]))$, provided that x and y are different and y is not free in t.

When terms are built using the positive polarity, the cut moves to the left branch, which means that the substitution can be defined recursively on the first argument.

```
psubst (papp U V K) R (papp U V H) :- pi x\ psubst (K x) R (H x).
psubst (pabs S K)   R (pabs S H)   :- pi x\ psubst (K x) R (H x).
psubst (pvar U)     R (R U).
```

Note that the last line of this specification uses a meta-level β-reduction but only to effect a variable renaming substitution. An example query using this last predicate is the following.

```
?- psubst (papp w w y\ papp y y z\ papp z z v\ pvar v)
          (x\ papp x x pvar) R.
R = papp w w (y\ papp y y (z\ papp z z (v\ papp v v u\ pvar u)))
```

Another example of substitution using this style of syntax is given graphically (and using the `name` notation) in Figure 5.

We can instead write this substitution operation as a prefix operator using the following functional equations. Below, the operation $t[x := u]$ denotes the replacement of every free occurrence of x in t by the variable u, provided that no free occurrence of x is in the scope of a binding on u.

- $[(\texttt{papp}\ u\ v\ (\lambda y.K))/x]R = (\texttt{papp}\ u\ v\ (\lambda y.[K/x]R)$, provided x and y are different;
- $[(\texttt{pabs}\ S\ (\lambda y.K))/x]R = (\texttt{pabs}\ S\ (\lambda y.[K/x]R)$, provided x and y are different; and
- $[(\texttt{pvar}\ u)/x]R = R[x := u]$, provided no free occurrence of x is in the scope of a binding on u.

Given our discussion of checking the simulation of two untyped λ-terms in Section 4.4, we know that such terms can be represented as a pair, say, $\langle \Gamma,\ u\rangle$ where Γ is an association list between a variable (of type `var`) and an operation (of type `op`) that indicates the kind of node that variable names (either an application or an abstraction). An equivalent description for substitution can then be given as follow: The result of substituting the term $\langle \Gamma,\ u\rangle$ for x in the term $\langle \Gamma',\ u'\rangle$ is the term $\langle (\Gamma'[x := u]) \sqcup \Gamma,\ u'\rangle$, where $\sqcup$ denotes appending of two lists.

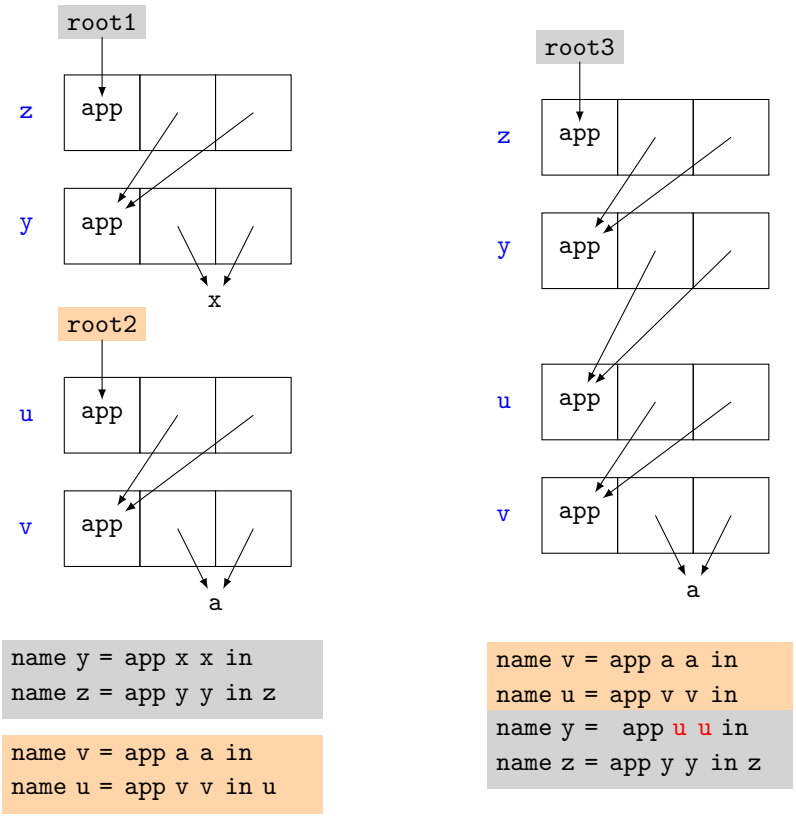


Figure 5 The term at `root3` is the result of substituting the term at `root2` for `x` into the term at `root1`. Operationally speaking, the pointers to `x` are redirected to point to `u` instead.

6 Related and future work

One goal of developing a logical framework, such as *LJF*, is to account for many other calculi within that framework. The literature contains other approaches to term representation that have also been motivated by focused proof systems [11, 12, 23]. Since *LJF* can easily account for several other focusing proof systems for intuitionistic logic [24], this framework should similarly account for such term calculi. Other frameworks have been used to justify term structures: for example, it would be interesting to see if there are any overlaps with the terms-as-graphs work of Grabmayer [19].

While many constructors for building term structures are only second-order, it is natural and occasionally important to be able to treat constructors of order greater than 2. Of course, the proof theory of *LJF* can treat formulas of all orders. However, the notion of synthetic inference rule (which is here limited to second order) would need to be generalized. In a setting with such higher-order constructors, cut elimination should be able to derive and generalize *hereditary substitutions* [40].

As mentioned in Section 4.4, the λgraphs in [9] represent sharing as DAG structures in the untyped λ-calculus. We conjecture that by using a multifocused version of the *LJF* proof system, we should be able to prove that maximal multifocused *LJF* proofs correspond to λgraphs. Maximal multifocused proofs have been shown elsewhere to correspond to graphical proof systems, such as proof nets [8], expansion proofs [7], and natural deduction proofs [35].

The black box methods of probing the structure of terms with sharing (see Sections 4.3 and 4.4) is closely related to well-established results in concurrency theory. However, these methods are not based on proof-theoretic principles, at least not that we have established here. We hope to find a way to describe trace equivalence and bisimilarity via proof-theoretic concepts. These notions seem related to Girard's Ludics project [18]. One promising approach might start with recognizing that simulations can be seen as winning strategies and that winning strategies can be related to focused proofs [10].

References

1 Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992. doi:10.1093/logcom/2.3.297.

2 David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu, and Yuting Wang. Abella: A system for reasoning about relational specifications. *Journal of Formalized Reasoning*, 7(2):1–89, 2014. doi:10.6092/issn.1972-5787/4650.

3 Taus Brock-Nannestad, Nicolas Guenot, and Daniel Gustafsson. Computation in focused intuitionistic logic. In Moreno Falaschi and Elvira Albert, editors, *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming, Siena, Italy, July 14–16, 2015*, pages 43–54. ACM, 2015. doi:10.1145/2790449.2790528.

4 Paola Bruscoli and Alessio Guglielmi. On structuring proof search for first order linear logic. *Theoretical Computer Science*, 360(1-3):42–76, 2006. doi:10.1016/j.tcs.2005.11.047.

5 Iliano Cervesato and Frank Pfenning. A linear spine calculus. *Journal of Logic and Computation*, 13(5):639–688, 2003. doi:10.1093/logcom/13.5.639.

6 Kaustuv Chaudhuri. Focusing strategies in the sequent calculus of synthetic connectives. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *LPAR: International Conference on Logic, Programming, Artificial Intelligence and Reasoning*, volume 5330 of *LNCS*, pages 467–481. Springer, November 2008. doi:10.1007/978-3-540-89439-1_33.

7 Kaustuv Chaudhuri, Stefan Hetzl, and Dale Miller. A multi-focused proof system isomorphic to expansion proofs. *J. of Logic and Computation*, 26(2):577–603, 2016. doi:10.1093/logcom/exu030.

8 Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin. Canonical sequent proofs via multi-focusing. In G. Ausiello, J. Karhumäki, G. Mauri, and L. Ong, editors, *Fifth International Conference on Theoretical Computer Science*, volume 273 of *IFIP*, pages 383–396. Springer, September 2008. doi:10.1007/978-0-387-09680-3_26.

9 Andrea Condoluci, Beniamino Accattoli, and Claudio Sacerdoti Coen. Sharing equality is linear. In *Proceedings of the 21st International Symposium on Principles and Practice of Declarative Programming*, pages 1–14, 2019. doi:10.1145/3354166.3354174.

10 Olivier Delande, Dale Miller, and Alexis Saurin. Proof and refutation in MALL as a game. *Annals of Pure and Applied Logic*, 161(5):654–672, February 2010. doi:10.1016/j.apal.2009.07.017.

11 Roy Dyckhoff and Stephane Lengrand. Call-by-value λ-calculus and LJQ. *J. of Logic and Computation*, 17(6):1109–1134, 2007. doi:10.1093/logcom/exm037.

12 José Espírito Santo. Completing herbelin's programme. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4583 of *LNCS*, pages 118–132. Springer, 2007. doi:10.1007/978-3-540-73228-0_10.

13 Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. *ACM SIGPLAN Notices*, 28(6):237–247, 1993. doi:10.1145/155090.155113.

14 Andrew Gacek, Dale Miller, and Gopalan Nadathur. Nominal abstraction. *Information and Computation*, 209(1):48–73, 2011. doi:10.1016/j.ic.2010.09.004.

15 Gerhard Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1935. Translation of articles that appeared in 1934-35. Collected papers appeared in 1969. doi:10.1007/BF01201353.

16 Ulysse Gérard and Dale Miller. Separating functional computation from relations. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *LIPIcs*, pages 23:1–23:17, 2017. doi:10.4230/LIPIcs.CSL.2017.23.

17 Jean-Yves Girard. A new constructive logic: classical logic. *Math. Structures in Comp. Science*, 1:255–296, 1991. doi:10.1017/S0960129500001328.

18 Jean-Yves Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3):301–506, June 2001. doi:10.1017/S096012950100336X.

19 Clemens Grabmayer. Modeling terms by graphs with structure constraints (two illustrations). In Maribel Fernández and Ian Mackie, editors, *TERMGRAPH@FSCD*, volume 288 of *EPTCS*, pages 1–13, 2018. doi:10.48550/arXiv.1902.02010.

20 Stéphane Graham-Lengrand. Polarities and focussing: a journey from realisability to automated reasoning, December 2014. Habilitation à diriger des recherches. URL: https://tel.archives-ouvertes.fr/tel-01094980.

21 Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993. doi:10.1145/138027.138060.

22 Hugo Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In *Computer Science Logic, 8th International Workshop, CSL '94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1995. doi:10.1007/BFb0022247.

23 Hugo Herbelin. *Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de lambda-termes et comme calcul de stratégies gagnantes*. PhD thesis, Université Paris 7, 1995. URL: https://tel.archives-ouvertes.fr/tel-00382528.

24 Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009. Abstract Interpretation and Logic Programming: In honor of professor Giorgio Levi. doi:10.1016/j.tcs.2009.07.041.

25 Chuck Liang and Dale Miller. A focused approach to combining logics. *Annals of Pure and Applied Logic*, 162(9):679–697, 2011. doi:10.1016/j.apal.2011.01.012.

26 Chuck Liang and Dale Miller. Focusing Gentzen's LK proof system. In Thomas Piecha and Kai Wehmeier, editors, *Peter Schroeder-Heister on Proof-Theoretic Semantics*, Outstanding Contributions to Logic. Springer, 2022. To appear. URL: https://hal.archives-ouvertes.fr/hal-03457379.

27 Sonia Marin, Dale Miller, Elaine Pimentel, and Marco Volpe. From axioms to synthetic inference rules via focusing. *Annals of Pure and Applied Logic*, 173(5):1–32, 2022. doi:10.1016/j.apal.2022.103091.

28 Dale Miller. Mechanized metatheory revisited. *Journal of Automated Reasoning*, 63(3):625–665, October 2019. doi:10.1007/s10817-018-9483-3.

29 Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, June 2012. doi:10.1017/CBO9781139021326.

30 Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51(1–2):125–157, 1991. doi:10.1016/0168-0072(91)90068-W.

31 Dale Miller and Alwen Tiu. A proof theory for generic judgments. *ACM Trans. on Computational Logic*, 6(4):749–783, October 2005. doi:10.1145/1094622.1094628.

32 Dale Miller and Jui-Hsuan Wu. A positive perspective on term representations: Extended paper. Technical report, Inria, 2022. URL: https://hal.inria.fr/hal-03843587.

33 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, Part II. *Information and Computation*, 100(1):41–77, 1992. doi:10.1016/0890-5401(92)90009-5.

34 Guillaume Munch-Maccagnoni and Gabriel Scherer. Polarised intermediate representation of lambda calculus with sums. In *30th Symp. on Logic in Computer Science*, pages 127–140. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.22.

35 Elaine Pimentel, Vivek Nigam, and João Neto. Multi-focused proofs with different polarity assignments. In Mario Benevides and Rene Thiemann, editors, *Proc. of the Tenth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2015)*, volume 323 of *ENTCS*, pages 163–179, July 2016. doi:10.1016/j.entcs.2016.06.011.

36 Jan von Plato. Natural deduction with general elimination rules. *Archive for Mathematical Logic*, 40(7):541–567, 2001. doi:10.1007/s001530100091.

37 Robert J. Simmons. Structural focalization. *ACM Trans. on Computational Logic*, 15(3):21, 2014. doi:10.1145/2629678.

38 Alwen Tiu and Dale Miller. Proof search specifications of bisimulation and modal logics for the π-calculus. *ACM Trans. on Computational Logic*, 11(2):13:1–13:35, 2010. doi:10.1145/1656242.1656248.

39 Philip Wadler. Call-by-value is dual to call-by-name. In *8th Int. Conf. on Functional Programming*, pages 189–201, New York, NY, 2003. ACM. doi:10.1145/944705.944723.

40 Keven Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: The propositional fragment. In *Post-proceedings of TYPES 2003 Workshop*, number 3085 in LNCS. Springer, 2003. doi:10.1007/978-3-540-24849-1_23.

41 Noam Zeilberger. Focusing and higher-order abstract syntax. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 359–369. ACM, 2008. doi:10.1145/1328897.1328482.

42 Noam Zeilberger. On the unity of duality. *Annals of Pure and Applied Logic*, 153(1), 2008. Special issue on classical logic and computation. doi:10.1016/j.apal.2008.01.001.

Enhanced Induction in Behavioural Relations

Davide Sangiorgi
University of Bologna, Italy
INRIA, Sophia Antipolis, France

Abstract

We outline an attempt at transporting the well-known theory of enhancements for the *coinduction* proof method, widely used on behavioural relations such as bisimilarity, onto the realms of *inductive* behaviour relations, i.e., relations defined from inductive observables, and discuss relevant literature.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases coinduction, induction, semantics, behavioural relations

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.4

Category Invited Talk

Related Version *Full Version*: https://dl.acm.org/doi/10.1145/3498679

1 Discussion

In this paper we discuss, informally, an attempt [35] at transporting the well-known theory of enhancements for the *coinduction* proof method, widely used on behavioural relations such as bisimilarity, onto the realms of *inductive* behaviour relations, i.e., relations defined from inductive observables. We also comment on the relevant literature. We refer to [35] for more details.

Behavioural relations (equalities, preorders) represent a one of the most basic elements for reasoning on programs or systems, because any transformation or property that we wish to prove is supposed to be in agreement with the behavioural relation adopted. A number of proposals for behavioural preorders and equalities have been made in the literature; see, e.g, van Glabbeek's spectrum [11,12]. Among the notions so formulated, bisimilarity has emerged as one of the most studied and used [19, 21]. While introduced in Concurrency Theory, bisimilarity has spread to other areas of Computer Science, as well as to other domains such as Mathematics and Cognitive Science.

Bisimilarity is the union of all bisimulations. A bisimulation is a relation on the terms of a language that is invariant under the observables of the language (i.e., what can be observed of the terms). Thus the definition itself immediately leads to a well-established proof technique: to prove two terms bisimilar, find a bisimulation relation containing the two terms as a pair. Furthermore, such a proof method can be enhanced, with the goal of making it more effective (easier to use, both in paper proofs and in tools for automated or semi-automated analysis) and more broadly applicable. Examples of enhancements are "up-to context", "up-to transitive closure", "up-to bisimilarity", "up-to environment", and so on [29]. Theories of enhancements have been proposed [25, 26, 31]. Most important, these theories allow one to *combine* enhancements so to obtain, for free, the soundness of more complex enhancements. Bisimilarity and the bisimulation proof method are instances of a coinductive definition and of the coinduction proof method. Analogously the bisimulation enhancements can be lifted to coinduction; see [28] for a presentation that follows fixed-point theory. Abstract formulations of the meaning of coinductive enhancements have also been given using category theory. The main technical tools are final semantics, coalgebras, spans of coalgebra homomorphisms, fibrations, and corecursion schemes. See, e.g., [4, 27, 30] (based on earlier works such as [3, 15, 17, 18, 37]), and [2, 5, 6, 16]. Enhancements of corecursion

schemes may also be examined using the generalised powerset construction [36]. For more details on coinductive enhancements, we refer to the technical survey [28] and to the historical review [29].

The bisimulation proof method and its enhancements are a major reason for the success of bisimilarity. Sometimes the enhancements seem essential to be able to carry out a proof: defining a full bisimulation, with all needed pairs, can be considerably hard, let alone carrying out the whole proof. This is frequent in languages for name mobility such as the π-calculus and its many dialects, and in languages including higher-order features such as λ-calculi. In these languages bisimilarity is hardly ever applied without enhancements.

As a behavioural equivalence, however, bisimilarity has also been criticised. One of the main arguments is that it may be regarded as too fine, discriminating processes that an external observer could not tell apart. For instance, in the CSP community failure equivalence [7] is used in place of bisimilarity. Another argument against bisimilarity is that it does not have a natural associated preorder. For instance, similarity – the "one-way" bisimilarity – or variants of it do not yield bisimilarity as their induced equivalence [33]. (Further, similarity as a behavioural preorder is often inadequate because it does not respect deadlocks.) Various inductive behavioural relations, both preorders and equivalences, have been put forward and studied that improve on such limitations: examples are preorders based on traces, failures, ready sets, refusals, may and must testing, ready and failure traces, e.g., [1, 7–9, 12, 13, 20, 22–24]. We call them "inductive" because resulting from inductively-defined observables, usually enriched forms of traces. Correspondingly, we sometimes call "inductive" the resulting enhancements.

At the heart of theories of bisimulation enhancements such as [25, 26, 31] is the notion of *progression*. A progression from a relation $\mathcal{R}$ to a relation $\mathcal{S}$, written $\mathcal{R} \rangle\!\rightarrow \mathcal{S}$, indicates that pairs of processes in $\mathcal{R}$ can match each other's actions and their derivatives (i.e., the processes resulting from performing such actions) are in $\mathcal{S}$. The progressions that are considered are of the form $\mathcal{R} \rangle\!\rightarrow \mathcal{F}(\mathcal{R})$, where $\mathcal{F}$ is a function on relations. Conditions on functions on relations guarantee *soundness* of the progressions, meaning that if $\mathcal{R} \rangle\!\rightarrow \mathcal{F}(\mathcal{R})$ then $\mathcal{R}$ only includes pairs of bisimilar processes. These are functorial-like conditions such as *respectfulness* [31] and *compatibility* [25]. Such conditions can then be extended to higher-order functions, also called *constructors*, so to be able to combine sound functions, that is, to derive sophisticated sound functions – and hence sophisticated proof techniques for bisimilarity – from simpler ones. As an example, "up-to bisimilarity" can be combined with "up-to context" yielding "up-to bisimilarity and context", in which one is allowed, in the bisimulation game, both to rewrite the derivative processes into bisimilar ones, and to remove, in the resulting terms, a common context. In fact, in this way even "up-to bisimilarity" is derivable from simpler functions, namely the identity function and the constant function mapping every relation onto bisimilarity itself [31].

To investigate the enhancements in an inductive setting, in [35] the observables for the inductive behavioural preorders and equivalences are described by means of *modal formulas*. The operators include the "diamond" $\langle\, \mu \,\rangle.\, \theta$, to detect the possibility of performing the action μ, a (possibly infinitary) 'and', to permit multiple observations, and a set of atomic observables. (Without the atomic observables, it is the positive fragment of Hennessy-Milner logic [14].) Progressions are maintained as the basic schema for studying enhancements. In fact, one-way progressions, called semi-progressions and written $\mathcal{R} \rangle\!\rightarrowtail \mathcal{S}$, are employed, aiming to capture also preorders (besides equivalences). The meaning of soundness, and the conditions on functions to guarantee soundness, are however modified. Moreover, everything is parametrised on a preorder, say $\precsim$, as the theory is supposed to be applicable to different

preorders. Thus, in such enhancements, a function $\mathcal{F}$ is *sound for* $\precsim$ if $\mathcal{R} \rightarrowtail \mathcal{F}(\mathcal{R})$ implies that $\mathcal{R}$ is included in the given preorder $\precsim$; and similarly for equivalences. The crux of this theory of inductive enhancements is the condition for functional soundness that should permit composition of enhancements. The condition hinges on the inductive definition of the observables. A weight is associated to each observable, intuitively expressing the depth of the nesting of actions in the behaviour of a process that may have be looked at when checking that observable. This yields a stratification of the preorder $\precsim$; at stage n of the preorder, $\precsim_n$, only the observables of weight less than or equal to n are taken into account. The condition for functional soundness, called *weight-preservation*, requires that if $\mathcal{R}$ complies with $\precsim_n$, i.e., $\mathcal{R} \subseteq \precsim_n$, then also $\mathcal{F}(\mathcal{R})$ should comply, i.e., $\mathcal{F}(\mathcal{R}) \subseteq \precsim_n$. In the case of equivalences, rather than preorders, one adds analogous converse requirements on pairs of related processes.

Common basic functions and constructors in the literature about bisimulation enhancements are shown to be weight-preserving, and may therefore be used also in inductive enhancements. Examples of basic functions and constructors are function composition, union, chaining (that gives us relational composition), and the context-closure function These closure properties allow one to derive sophisticated sound functions (and hence sophisticated proof techniques) from simpler ones. Examples of derived functions are the transitive-closure function, the closure under context and $\precsim$ (the analogous of the "up-to context and bisimilarity" enhancement for bisimilarity).

The inductive preorders and equivalences considered in [35] are the best-known relations in the literature, following [11, 12]. They include the trace, failure, failure trace, ready, and ready trace preorders (other preorders, like may and must testing and refusal, coincide with some of these, under mild conditions on the transitions performed by the processes), and their induced equivalences. In all cases, the soundness of the above functions and constructors is usually straightforward, the only exception being the context-closure function. As the theory of inductive enhancement is parametrised on a preorder, proofs can sometimes be made *parametric* on such a preorder, so to make them valid for a number of preorders. See [35] for examples.

The paper [35] develops its work for ordinary Labeled Transition Systems (LTSs). A theory is proposed both for strong semantics, where all actions are equally visible, and for weak semantics, in which a special action denoting internal activity may be partly or completely ignored. It is well-known that theories of weak coinductive enhancements tend to be rather more involved than the "strong" theories. For instance, a useful constructor, chaining, is sound only in the strong theories; to compensate for this, auxiliary relations such as *expansion* [32] and *contraction* [34] have been introduced. Similar issues show up in the inductive enhancements. In addition, some of the weak behavioural relations make use of state predicates such as *stability*, which do not appear in the strong case. Some of the technical solutions that are adopted for the inductive setting are inspired from those used in the coinductive setting, others are specific to the weight-based conditions for induction mentioned above. For instance, different forms of weak weight are considered (e.g., distinguishing the contribution of internal and visible actions), and their relative advantages and disadvantages are examined.

Another approach at transferring the coinductive enhancements to an inductive setting is based on the techniques of unique solutions of equations [10, 34]. Such proof techniques (for weak bisimilarity) employ equations as well as special inequations called *contractions*. The techniques give one the power of some bisimulation enhancements such as "up-to context", and can be transferred to relations such as trace equivalence and trace preorder. In general the techniques seem to have a limited applicability to preorders: they can only be used to show that a given process is related to the "syntactic solution of an equation", that is, the process whose syntactic definition is the equation itself.

2 Further work

We comment some directions of work to be explored. First, it would be interesting to see if and how the theory of inductive enhancements can be formulated in a more abstract setting, e.g., fixed-point theory or category theory. Proposals along these lines exist for the coinductive enhancements. Their meaning in the inductive setting is unclear, both because the observables are inductive and because of the coinductive flavour of the semi-progressions at the heart of the theory. Also, it is unclear how the theory could be lifted to a probabilistic setting and labelled Markov processes.

A powerful enhancement is up-to context. The paper [35] uses first-order LTSs and examines a a CCS-like language. Here an objective would be to examine general conditions that guarantee its soundness, more precisely the weight-preserving property. Such conditions could, for instance, look at the format of the rules defining the operational behaviour of the operators of the language. In coinduction, up-to-context has been shown very effective in *higher-order languages*, such as λ-calculi or languages enriched with functional features, and *nominal languages* such as the π-calculus. The objective could thus be extended towards the transfer of the inductive enhancements to these classes of languages.

Finally, it would be interesting to see if the theory can be lifted to behavioural relations that make use of both inductive and coinductive observables. Examples of observables that are naturally defined coinductively are infinite traces and divergence. Divergence in particular often appears in definition of *weak* behavioural relations (e.g., failure semantics and must testing).

References

1 Jos C. M. Baeten, Jan A. Bergstra, and Jan Willem Klop. Ready-trace semantics for concrete process algebra with the priority operator. *Comput. J.*, 30(6):498–506, 1987.

2 F. Bartels. Generalised coinduction. *Math. Struct. in Computer Science*, 13(2):321–348, 2003. doi:10.1017/S0960129502003900.

3 F. Bartels. *On generalised coinduction and probabilistic specification formats*. PhD thesis, CWI, Amsterdam, April 2004.

4 Henning Basold, Damien Pous, and Jurriaan Rot. Monoidal company for accessible functors. In *Proc. CALCO*, volume 72 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CALCO.2017.5.

5 Filippo Bonchi, Barbara König, and Daniela Petrisan. Up-to techniques for behavioural metrics via fibrations. In Sven Schewe and Lijun Zhang, editors, *CONCUR'18,*, volume 118 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.17.

6 Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. A general account of coinduction up-to. *Acta Inf.*, 54(2):127–190, 2017. doi:10.1007/s00236-016-0271-4.

7 Stephen D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984. doi:db/journals/jacm/BrookesHR84.html,10.1145/828.833.

8 Stephen D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In Stephen D. Brookes, A. W. Roscoe, and Glynn Winskel, editors, *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 281–305. Springer Verlag, 1984.

9 R. De Nicola and R. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

10 Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. Divergence and unique solution of equations. *Logical Methods in Computer Science*, 15(3), 2019. doi:10.23638/LMCS-15(3:12)2019.

11 R.J. van Glabbeek. The linear time—branching time spectrum II (the semantics of sequential systems with silent moves). In E. Best, editor, *Proc. CONCUR '93*, volume 715. Springer Verlag, 1993. doi:10.1007/3-540-57208-2_6.

12 R.J. van Glabbeek. The linear time—branching time spectrum I. In A. Ponse J. Bergstra and S. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001. doi:10.1016/b978-044482830-9/50019-9.

13 M. Hennessy. *Algebraic Theory of Processes*. The MIT Press, Cambridge, Mass., 1988.

14 M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.

15 Bart Jacobs. Distributive laws for the coinductive solution of recursive equations. *Information and Computation*, 204(4):561–587, 2006. doi:10.1016/j.ic.2005.03.006.

16 Marina Lenisa. From set-theoretic coinduction to coalgebraic coinduction: some results, some problems. *Electronical Notes in Computer Science*, 19:2–22, 1999. doi:10.1016/S1571-0661(05)80265-8.

17 Marina Lenisa, John Power, and Hiroshi Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. *Electronical Notes in Computer Science*, 33:230–260, 2000. doi:10.1016/S1571-0661(05)80350-0.

18 S. Milius, L. S. Moss, and D. Schwencke. Abstract GSOS rules and a modular treatment of recursive definitions. *Logical Methods in Computer Science*, 9(3), 2013. doi:10.2168/LMCS-9(3:28)2013.

19 R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

20 Ernst-Rüdiger Olderog and C. A. R. Hoare. Specification-oriented semantics for communicating processes. *Acta Inf.*, 23(1):9–66, 1986.

21 D. Park. A new equivalence notion for communicating systems. In G. Maurer, editor, *Bulletin EATCS*, volume 14, pages 78–80, 1981. Abstract of the talk presented at the Second Workshop on the Semantics of Programming Languages, Bad Honnef, March 16–20 1981. Abstracts collected in the Bulletin by B. Mayoh.

22 Iain Phillips. Refusal testing. *Theor. Comput. Sci.*, 50:241–284, 1987. A preliminary version in Proc. ICALP'86, Lecture Notes in Computer Science 226, Springer Verlag.

23 A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In W. Brauer, editor, *12th ICALP*, volume 194 of *Lecture Notes in Computer Science*, pages 15–32. Springer Verlag, 1985.

24 Lucia Pomello. Some equivalence notions for concurrent systems. an overview. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 381–400. Springer, 1985. doi:10.1007/BFb0016202.

25 D. Pous. Complete lattices and up-to techniques. In *Proc. APLAS '07*, volume 4807 of *Lecture Notes in Computer Science*, pages 351–366. Springer Verlag, 2007. doi:10.1007/978-3-540-76637-7_24.

26 Damien Pous. Coinduction all the way up. In *Proc. LICS*, pages 307–316. ACM, 2016. doi:10.1145/2933575.2934564.

27 Damien Pous and Jurriaan Rot. Companions, Codensity, and Causality. In *Proc. FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 106–123. Springer Verlag, 2017. doi:10.1007/978-3-662-54458-7_7.

28 Damien Pous and Davide Sangiorgi. Enhancements of the bisimulation proof method. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2012.

29 Damien Pous and Davide Sangiorgi. Bisimulation and coinduction enhancements: A historical perspective. *Formal Asp. Comput.*, 31(6):733–749, 2019. doi:10.1007/s00165-019-00497-w.

30 Jurriaan Rot, Filippo Bonchi, Marcello M. Bonsangue, Damien Pous, Jan Rutten, and Alexandra Silva. Enhanced coalgebraic bisimulation. *Mathematical Structures in Computer Science*, 27(7):1236–1264, 2017. doi:10.1017/S0960129515000523.

31 D. Sangiorgi. On the bisimulation proof method. *Journal of Mathematical Structures in Computer Science*, 8:447–479, 1998. doi:10.1017/S0960129598002527.

32 D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". In W.R. Cleveland, editor, *Proc. CONCUR '92*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer Verlag, 1992. doi:10.1007/BFb0084781.

33 Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012. doi:10.1017/CBO9780511777110.

34 Davide Sangiorgi. Equations, contractions, and unique solutions. *ACM Trans. Comput. Log.*, 18(1):4:1–4:30, 2017. doi:10.1145/2971339.

35 Davide Sangiorgi. From enhanced coinduction towards enhanced induction. *Proc. ACM Program. Lang.*, 6(POPL):1–29, 2022. doi:10.1145/3498679.

36 A. Silva, F. Bonchi, M. Bonsangue, and J. Rutten. Generalizing the powerset construction, coalgebraically. In *FSTTCS*, LIPIcs, pages 272–283. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.272.

37 Tarmo Uustalu, Varmo Vene, and Alberto Pardo. Recursion schemes from comonads. *Nord. J. Comput.*, 8(3):366–390, 2001. URL: http://www.cs.helsinki.fi/njc/References/uustaluvp2001:366.html.

A Cyclic Proof System for Full Computation Tree Logic

Bahareh Afshari
Institute for Logic, Language and Computation, University of Amsterdam, The Netherlands
Department of Philosophy, Linguistics and Theory of Science, University of Gothenburg, Sweden

Graham E. Leigh
Department of Philosophy, Linguistics and Theory of Science, University of Gothenburg, Sweden

Guillermo Menéndez Turata
Institute for Logic, Language and Computation, University of Amsterdam, The Netherlands

Abstract

Full Computation Tree Logic, commonly denoted CTL^*, is the extension of Linear Temporal Logic LTL by path quantification for reasoning about branching time. In contrast to traditional Computation Tree Logic CTL, the path quantifiers are not bound to specific linear modalities, resulting in a more expressive language. We present a sound and complete hypersequent calculus for CTL^*. The proof system is cyclic in the sense that proofs are finite derivation trees with back-edges. A syntactic success condition on non-axiomatic leaves guarantees soundness. Completeness is established by relating cyclic proofs to a natural ill-founded sequent calculus for the logic.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Proof theory

Keywords and phrases Full computation tree logic, Hypersequent calculus, Cyclic proofs

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.5

Funding Supported by the Knut and Alice Wallenberg Foundation [2020.0199]; Swedish Research Council [2017-05111, 2016-03502]; and the Dutch Research Council [OCENW.M20.048].

1 Introduction

Full Computation Tree Logic, CTL^*, is a well studied temporal logic in theoretical computer science. Its roots can be traced to Prior's stance of allowing time-distinctions (temporality) and future-alternatives (branching) in the assessment of truth. As such, it builds on a binary temporal operator *until*, written $\varphi \mathsf{U} \psi$ to express "φ is true until ψ becomes true", and a unary *next* modality, written $\mathsf{X}\varphi$, for denoting "φ is true at the next step". Additionally, path quantification is explicitly available via the *universal* and *existential* operators, $\mathsf{A}\varphi$ and $\mathsf{E}\varphi$ which, respectively, have the intended interpretation of *all* or *some* execution paths satisfy property φ. If these path quantifiers are only allowed as guards of an eventuality, namely in the form of $\mathsf{Q}(\varphi \mathsf{U} \psi)$ for $\mathsf{Q} \in \{\mathsf{A}, \mathsf{E}\}$, a strictly less expressive logic known as Computation Tree Logic, CTL, is realised. Dispensing with the path quantifiers altogether results in the most widely used temporal logic of all, Pnueli's Linear Temporal Logic, LTL.[1]

Much has already been achieved for the proof theory of CTL^* in terms of introducing finitary, infinitary, and cyclic tableaux systems (see e.g. [1, 22, 24, 25, 12, 13]). Most noteworthy is the complete (Hilbert-style) axiomatisation [22] provided by M. Reynolds in 2001. The system was later extended to include past modalities [23], a process which also simplified Reynolds' axioms and the completeness proof, though it remains a highly intricate analysis.

[1] For a comprehensive coverage of temporal logics in computer science we refer the reader to [10].

In this article we revisit axiomatisation of CTL* in the light of recent developments in the area of cyclic proof theory. We introduce a sound and complete cyclic hypersequent calculus for the logic based on an intuitive set of inference rules. Local soundness of the inferences is immediate. Global soundness is achieved by a correctness condition on cyclic proofs similar to the annotated mechanism introduced by Jungteerapanich [14] and Stirling [30] for the modal μ-calculus. There are two notable deviations from the Jungteerapanich–Stirling framework. First, it is sufficient to restrict attention to annotations of length ≤ 1, i.e., annotations that identify at most one temporal operator in a formula. Second, and in strong contrast with the modal μ-calculus, annotations may identify an occurrence of either the release operator (the dual operator to "until", corresponding to the ν quantifier in μ-calculus) *or* the until operator (corresponding to the μ quantifier). Annotations of until operators play a crucial role in ensuring soundness of cycles featuring the existential path quantifier, and allow the correctness condition on proofs to be determined by the simple cycles. In particular, recognising whether a cyclic derivation is a proof requires only linear time. Finally, our system is cut-free, a property which does not come naturally for temporal logics.[2]

Aside from furthering the study of temporal logics, the present work contributes to the development of cyclic proof systems beyond the traditional realm of Gentzen-style sequent calculi where the main focus of work in cyclic proof theory currently resides (see, e.g., [30, 15, 4, 11, 17, 29, 27, 2] for recent contributions to modal and temporal logics). Such a move seems inevitable as ever more structured fixpoint logics are analysed, as witnessed by Rooduijn's cyclic hypersequent calculi for modal logics with the master modality [26], and Das and Girlando's ill-founded hypersequent calculus for Transitive Closure Logic [9].

2 The full computation tree logic CTL*

Let Prop be a countably infinite set of propositional constants. Formulas of CTL* are given by the following grammar.

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \varphi \mid \varphi \mathsf{R} \varphi \mid \mathsf{A}\varphi \mid \mathsf{E}\varphi$$

where p ranges over Prop. We use Q to denote either the *universal (path) quantifier* A or the *existential (path) quantifier* E, and O to denote either the *until operator* U or the *release operator* R. A *literal formula* is either a propositional constant or the negation of one. Given a set of formulas Φ, we define $\mathsf{X}\Phi := \{\mathsf{X}\varphi : \varphi \in \Phi\}$. The notion of *subformula* is defined in the usual manner. We write $\psi \leq \varphi$ to denote that ψ is a subformula of φ.

A *labelled transition system (LTS)* is a triple $\mathcal{S} = (S, \to, \lambda)$ where S is a non-empty set of *states*, $\to$ is a binary relation on S, and $\lambda \colon S \to \mathcal{P}(\mathsf{Prop})$ is a *labelling map* which assigns to each state a set of propositional constants. $\mathcal{S}$ is *serial* if for every $s \in S$ there is some $t \in S$ such that $s \to t$. A *path* through a serial LTS $\mathcal{S}$ is an infinite sequence of states $\sigma = s_0 s_1 \cdots$ such that $s_i \to s_{i+1}$ for every $i < \omega$. The i-th state in σ is denoted by $\sigma(i)$, and (σ, i) denotes the path $s_i s_{i+1} \ldots$. Given paths σ, and σ', we write $\sigma \sim \sigma'$ if $\sigma(0) = \sigma'(0)$.

Satisfaction for CTL* formulas is defined relative to paths through serial labelled transition systems:

- $\mathcal{S}, \sigma \models p$ iff $p \in \lambda(\sigma(0))$;
- $\mathcal{S}, \sigma \models \neg p$ iff $p \notin \lambda(\sigma(0))$;
- $\mathcal{S}, \sigma \models \varphi \wedge \psi$ iff $\mathcal{S}, \sigma \models \varphi$ and $\mathcal{S}, \sigma \models \psi$;

[2] For a discussion on cut-free sequent systems for temporal logic see, e.g., [7].

- $\mathcal{S}, \sigma \models \varphi \vee \psi$ iff $\mathcal{S}, \sigma \models \varphi$ or $\mathcal{S}, \sigma \models \psi$;
- $\mathcal{S}, \sigma \models \mathsf{X}\varphi$ iff $\mathcal{S}, (\sigma, 1) \models \varphi$;
- $\mathcal{S}, \sigma \models \varphi \mathsf{U} \psi$ iff there is a j such that $\mathcal{S}, (\sigma, j) \models \psi$ and $\mathcal{S}, (\sigma, i) \models \varphi$ for every $i < j$;
- $\mathcal{S}, \sigma \models \varphi \mathsf{R} \psi$ iff for every j, either $\mathcal{S}, (\sigma, j) \models \psi$, or there is an $i < j$ such that $\mathcal{S}, (\sigma, i) \models \varphi$;
- $\mathcal{S}, \sigma \models \mathsf{A}\varphi$ iff $\mathcal{S}, \sigma' \models \varphi$ for every $\sigma' \sim \sigma$;
- $\mathcal{S}, \sigma \models \mathsf{E}\varphi$ iff $\mathcal{S}, \sigma' \models \varphi$ for some $\sigma' \sim \sigma$.

Explicit mention of $\mathcal{S}$ may be omitted when no ambiguity arises.

The *negation* of a formula φ, in symbols $\neg\varphi$, is defined inductively via the De Morgan dualities with $\neg\mathsf{X}\varphi := \mathsf{X}\neg\varphi$, $\neg(\varphi \mathsf{U} \psi) := \neg\varphi \mathsf{R} \neg\psi$, $\neg(\varphi \mathsf{R} \psi) := \neg\varphi \mathsf{U} \neg\psi$, $\neg\mathsf{A}\varphi := \mathsf{E}\neg\varphi$, and $\neg\mathsf{E}\varphi := \mathsf{A}\neg\varphi$. A formula φ is *satisfiable* if there is a serial LTS $\mathcal{S}$ and a path σ through $\mathcal{S}$ such that $\mathcal{S}, \sigma \models \varphi$, and *unsatisfiable* otherwise. We say that φ is *valid* if $\neg\varphi$ is unsatisfiable. We write $\varphi \equiv \psi$, and say that φ and ψ are *equivalent*, if for every serial LTS $\mathcal{S}$ and every path σ on $\mathcal{S}$, we have $\mathcal{S}, \sigma \models \varphi$ iff $\mathcal{S}, \sigma \models \psi$.

Note that $\varphi \mathsf{U} \psi \equiv \psi \vee [\varphi \wedge \mathsf{X}(\varphi \mathsf{U} \psi)]$ and dually $\varphi \mathsf{R} \psi \equiv \psi \wedge [\varphi \vee \mathsf{X}(\varphi \mathsf{R} \psi)]$. These equivalences exhibit the fixpoint nature of the until and release operators.

3 Ill-founded proofs

In this section we present a sound and complete, ill-founded, sequent-style proof system for the logic CTL^* inspired by Dam's syntax trees for an embedding of CTL^* into the modal μ-calculus [8]. Proofs are potentially infinite trees whose infinite branches satisfy a syntactic correctness condition that ensures soundness of the calculus.

A *tree* is a pair $(T, \leq_T)$ where T is a non-empty, possibly countably infinite set of *vertices*, and $\leq_T$ is a partial order on T such that $\{t \in T : t \leq_T s\}$ is well-ordered for every $s \in T$ and there is a *root* $r \in T$ satisfying $r \leq_T s$ for every vertex s. We abuse notation and write T in place of $(T, \leq_T)$. For vertices $s, t \in T$, we let $[s, t]_T = s \cdots t$ be the maximal (finite) path from s to t. If $s \not\leq_T t$, $[s, t]_T$ will be the empty sequence.

A *sequent* is an expression of the form $\mathsf{Q}\Phi$, where $\mathsf{Q} \in \{\mathsf{A}, \mathsf{E}\}$ and Φ is a finite set of formulas. We identify the sequent $\mathsf{Q}\{\varphi\}$ with the formula $\mathsf{Q}\varphi$, and write $\mathsf{Q}\{\Phi, \varphi\}$ as shorthand for $\mathsf{Q}(\Phi \cup \{\varphi\})$. To each sequent $\mathsf{Q}\Phi$ we associate a corresponding formula $(\mathsf{Q}\Phi)^*$ by setting $(\mathsf{A}\Phi)^* := \mathsf{A} \bigvee \Phi$ and $(\mathsf{E}\Phi)^* := \mathsf{E} \bigwedge \Phi$. We abuse notation and write $\sigma \models \mathsf{Q}\Phi$ for $\sigma \models (\mathsf{Q}\Phi)^*$. We define $\top := \mathsf{E}\varnothing$ and $\bot := \mathsf{A}\varnothing$. A *literal sequent* is either $\top$, $\bot$, or a sequent of the form $\mathsf{Q}\lambda$ where λ is a literal formula.

A *hypersequent* is a finite set of sequents. Hypersequents are denoted by symbols Γ, Δ, Σ. We extend the translation $(\cdot)^*$ to hypersequents by setting $\Gamma^* := \bigvee\{(\mathsf{Q}\Phi)^* : \mathsf{Q}\Phi \in \Gamma\}$. As with sequents, we abuse notation and write $\sigma \models \Gamma$ for $\sigma \models \Gamma^*$.

▶ **Definition 3.1** (Derivation). *A* CTL^*_∞ derivation *of a formula φ is a finite or infinite tree T whose vertices are labelled according to the rules in Figure 1 and such that:*

1. *The root of T has label $\mathsf{A}\varphi$.*
2. *Every infinite branch of T contains infinitely many applications of* AX *or* EX.

A vertex of a derivation is *modal* if it is the conclusion of an instance of AX or EX. Condition 2 in the definition requires that every infinite branch of a derivation contains infinitely many modal vertices. This prevents the construction of infinite branches by "degenerate" applications of rules, for instance applying rule ALit repeatedly to sequent $\mathsf{A}p$.

The rules of CTL^*_∞ are correct in the following sense.

▶ **Proposition 3.2.** *The sequent rules of* CTL^*_∞ *are sound: If* $\dfrac{\Gamma_1 \quad \cdots \quad \Gamma_n}{\Delta}$ *is an instance of a* CTL^*_∞ *rule and each of $\Gamma_1, \ldots, \Gamma_n$ is valid, then Δ is valid.*

$$\mathsf{ax}_p\ \frac{}{\mathsf{Q}p, \mathsf{Q}'\neg p, \Delta} \qquad \mathsf{ax}_\top\ \frac{}{\top, \Delta}$$

$$\mathsf{ALit}\ \frac{\mathsf{A}\Phi, \mathsf{A}\lambda, \Delta}{\mathsf{A}\{\Phi, \lambda\}, \Delta} \qquad \mathsf{ELit}\ \frac{\mathsf{E}\Phi, \Delta \qquad \mathsf{E}\lambda, \Delta}{\mathsf{E}\{\Phi, \lambda\}, \Delta}$$

$$\mathsf{A}\vee\ \frac{\mathsf{A}\{\Phi, \varphi, \psi\}, \Delta}{\mathsf{A}\{\Phi, \varphi \vee \psi\}, \Delta} \qquad \mathsf{E}\vee\ \frac{\mathsf{E}\{\Phi, \varphi\}, \mathsf{E}\{\Phi, \psi\}, \Delta}{\mathsf{E}\{\Phi, \varphi \vee \psi\}, \Delta}$$

$$\mathsf{A}\wedge\ \frac{\mathsf{A}\{\Phi, \varphi\}, \Delta \qquad \mathsf{A}\{\Phi, \psi\}, \Delta}{\mathsf{A}\{\Phi, \varphi \wedge \psi\}, \Delta} \qquad \mathsf{E}\wedge\ \frac{\mathsf{E}\{\Phi, \varphi, \psi\}, \Delta}{\mathsf{E}\{\Phi, \varphi \wedge \psi\}, \Delta}$$

$$\mathsf{AA}\ \frac{\mathsf{A}\Phi, \mathsf{A}\{\psi\}, \Delta}{\mathsf{A}\{\Phi, \mathsf{A}\psi\}, \Delta} \qquad \mathsf{EA}\ \frac{\mathsf{E}\Phi, \Delta \qquad \mathsf{A}\{\psi\}, \Delta}{\mathsf{E}\{\Phi, \mathsf{A}\psi\}, \Delta}$$

$$\mathsf{AE}\ \frac{\mathsf{A}\Phi, \mathsf{E}\{\psi\}, \Delta}{\mathsf{A}\{\Phi, \mathsf{E}\psi\}, \Delta} \qquad \mathsf{EE}\ \frac{\mathsf{E}\Phi, \Delta \qquad \mathsf{E}\{\psi\}, \Delta}{\mathsf{E}\{\Phi, \mathsf{E}\psi\}, \Delta}$$

$$\mathsf{AX}\ \frac{\mathsf{A}\Phi_i, \mathsf{E}\Psi_1, \ldots, \mathsf{E}\Psi_m}{\mathsf{AX}\Phi_1, \ldots, \mathsf{AX}\Phi_n, \mathsf{EX}\Psi_1, \ldots, \mathsf{EX}\Psi_m, \Sigma} \qquad \mathsf{EX}\ \frac{\mathsf{E}\Psi_1, \ldots, \mathsf{E}\Psi_m}{\mathsf{EX}\Psi_1, \ldots, \mathsf{EX}\Psi_m, \Sigma}$$

$$\mathsf{AU}\ \frac{\mathsf{A}\{\Phi, \varphi_1, \varphi_2\}, \Delta \qquad \mathsf{A}\{\Phi, \varphi_2, \mathsf{X}(\varphi_1 \mathsf{U} \varphi_2)\}, \Delta}{\mathsf{A}\{\Phi, \varphi_1 \mathsf{U} \varphi_2\}, \Delta} \qquad \mathsf{EU}\ \frac{\mathsf{E}\{\Phi, \varphi_2\}, \mathsf{E}\{\Phi, \varphi_1, \mathsf{X}(\varphi_1 \mathsf{U} \varphi_2)\}, \Delta}{\mathsf{E}\{\Phi, \varphi_1 \mathsf{U} \varphi_2\}, \Delta}$$

$$\mathsf{AR}\ \frac{\mathsf{A}\{\Phi, \varphi_2\}, \Delta \qquad \mathsf{A}\{\Phi, \varphi_1, \mathsf{X}(\varphi_1 \mathsf{R} \varphi_2)\}, \Delta}{\mathsf{A}\{\Phi, \varphi_1 \mathsf{R} \varphi_2\}, \Delta} \qquad \mathsf{ER}\ \frac{\mathsf{E}\{\Phi, \varphi_1, \varphi_2\}, \mathsf{E}\{\Phi, \varphi_2, \mathsf{X}(\varphi_1 \mathsf{R} \varphi_2)\}, \Delta}{\mathsf{E}\{\Phi, \varphi_1 \mathsf{R} \varphi_2\}, \Delta}$$

Figure 1 Rules of the system CTL^*_∞. Q and Q' are meta-variables for path quantifiers. In the rules ALit and ELit, λ ranges over literal formulas.

Let T be a CTL^*_∞ derivation. For vertices $s, t \in T$, we write $s \longrightarrow t$ if t is an immediate successor of s in T. We write $s \xrightarrow{\mathrm{R}} t$ to specify the rule R applied at s. A *path* through T is a finite or infinite sequence of vertices $s_0 \longrightarrow s_1 \longrightarrow \cdots$.

It is convenient to refer to the sequents labelling a vertex according to their role in the applied CTL^*_∞ rules. The distinguished sequent(s) in the conclusion of rules in Figure 1, such as the sequent $\mathsf{A}\{\Phi, \varphi \wedge \psi\}$ in A$\wedge$, are said to be *principal*. This includes all sequents in the conclusion of rules AX and EX. The distinguished sequent(s) in the premises of the rules, for instance $\mathsf{A}\{\Phi, \varphi\}$ and $\mathsf{A}\{\Phi, \psi\}$ in A$\wedge$, are called *active*. All sequents occurring in the hypersequent Δ are *side* sequents.

Let $\pi = (s_i)_{i<N}$ be a path through T where $N \in \omega \cup \{\omega\}$. For every i, we let $\rhd_i$ be the sequent trace relation between the labels of s_i and s_{i+1} given by $\mathsf{Q}\Xi \rhd_i \mathsf{Q}'\Psi$ if $\mathsf{Q}'\Psi$ arises from $\mathsf{Q}\Xi$ in the rule with conclusion s_i. That is, $\mathsf{Q}\Xi \rhd_i \mathsf{Q}'\Psi$ holds if $\mathsf{Q}\Xi = \mathsf{Q}'\Psi$ is a side sequent or $\mathsf{Q}\Xi$ is principal and $\mathsf{Q}'\Psi$ an active sequent arising from $\mathsf{Q}\Xi$. In particular, if s_i is a modal rule, we require $\Xi = \mathsf{X}\Psi$. A *sequent trace* on π is a sequence $(\mathsf{Q}_i\Xi_i)_{i<N}$ of sequents such that $\mathsf{Q}_i\Xi_i$ is in the label of s_i and $\mathsf{Q}_i\Xi_i \rhd_i \mathsf{Q}_{i+1}\Xi_{i+1}$. We drop the subscript from $\rhd_i$ when no ambiguity arises. A *context extraction* is a sequent trace of the form $\mathsf{Q}\{\Xi, \mathsf{Q}'\psi\} \rhd \mathsf{Q}'\psi$ or $\mathsf{Q}\{\Xi, \lambda\} \rhd \mathsf{Q}\lambda$ for a literal formula λ. A finite sequent trace $\mathsf{Q}_0\Xi_0 \rhd \cdots \rhd \mathsf{Q}_n\Xi_n$ is *stable* if $\mathsf{Q}_0 = \cdots = \mathsf{Q}_n$, and *circular* if $n > 0$ and $\mathsf{Q}_n\Xi_n = \mathsf{Q}_0\Xi_0$. When it is useful to specify the rule R by which $\mathsf{Q}_{i+1}\Xi_{i+1}$ results from $\mathsf{Q}_i\Xi_i$, we write $\mathsf{Q}_i\Xi_i \rhd^{\mathrm{R}} \mathsf{Q}_{i+1}\Xi_{i+1}$.

As with sequents, it is convenient to have a means of referring to the formulas in principal and active sequents according to their role in the CTL^*_∞ rules. Distinguished formulas in principal sequents in Figure 1 are called *principal*. Thus, $\varphi \wedge \psi$ is the principal formula of the rule $\mathsf{A}\wedge$. Every formula in the conclusion of rules AX and EX is principal. Formulas in the set Φ are *side* formulas. The distinguished formulas in active sequents are called *active*.

Let $(\mathsf{Q}_i \Xi_i)_{i<N}$ be a sequent trace on π. For each i, let $\triangleright_i$ denote the corresponding formula trace relation between Ξ_i and Ξ_{i+1}, given by $\varphi \triangleright_i \psi$ if formula ψ results from φ in the sense that either φ is principal and ψ an active formula corresponding to φ, or $\varphi = \psi$ is a side formula. A *formula trace* on $(\mathsf{Q}_i \Xi_i)_{i<N}$ is a sequence of formulas $(\varphi_i)_{i<N}$ such that $\varphi_i \in \Xi_i$ and $\varphi_i \triangleright_i \varphi_{i+1}$. As with sequent traces, we drop the subscript from $\triangleright_i$ when no ambiguity arises. When we want to specify the rule R by means of which φ_{i+1} results from φ_i, we write $\varphi_i \triangleright^{\mathrm{R}} \varphi_{i+1}$.

A *(fixpoint) unfolding* is a formula trace of the form $\psi \triangleright \mathsf{X}\psi$. Note that unfoldings can only be produced by rules AU, AR, EU, ER, and thus ψ is of the form $\psi_1 \mathsf{O} \psi_2$. Due to the presence of fixpoint unfoldings, the system CTL^*_∞ does not satisfy the subformula property: $\varphi \triangleright \psi$ does not imply $\psi \leq \varphi$. However, ψ does belong to the *closure* of φ, which is the natural replacement of the notion of subformula in this context:

▶ **Definition 3.3.** *The* closure *of a formula φ is the smallest set of formulas $\mathsf{Clos}(\varphi)$ satisfying:*

1. $\varphi \in \mathsf{Clos}(\varphi)$;
2. *If $\psi_1 \star \psi_2 \in \mathsf{Clos}(\varphi)$, for $\star \in \{\wedge, \vee\}$, then $\psi_1, \psi_2 \in \mathsf{Clos}(\varphi)$;*
3. *If $\psi_1 \mathsf{O} \psi_2 \in \mathsf{Clos}(\varphi)$, for $\mathsf{O} \in \{\mathsf{U}, \mathsf{R}\}$, then $\psi_1, \psi_2, \mathsf{X}(\psi_1 \mathsf{O} \psi_2) \in \mathsf{Clos}(\varphi)$;*
4. *If $\mathsf{X}\psi \in \mathsf{Clos}(\varphi)$, then $\psi \in \mathsf{Clos}(\varphi)$;*
5. *If $\mathsf{Q}\psi \in \mathsf{Clos}(\varphi)$, for $\mathsf{Q} \in \{\mathsf{A}, \mathsf{E}\}$, then $\psi \in \mathsf{Clos}(\varphi)$.*

It is easy to see that $\mathsf{Clos}(\varphi)$ is always finite. And, clearly, $\varphi \triangleright \psi$ implies $\psi \in \mathsf{Clos}(\varphi)$. Moreover, since $\mathsf{X}\varphi \triangleright \psi$ implies $\psi \in \{\mathsf{X}\varphi, \varphi\}$, we have:

▶ **Lemma 3.4.** *Let $\rho = (\varphi_i)_{i<N}$ be a finite or infinite formula trace. For every $\varphi \in \rho$, either $\varphi \leq \varphi_0$, or $\varphi = \mathsf{X}\psi$ for some $\psi \leq \varphi_0$.*

The following is a fundamental result about infinite formula traces that follows easily from Lemma 3.4.

▶ **Proposition 3.5.** *Let T be a CTL^*_∞ derivation of a formula φ, and let $(\varphi_i)_{i<\omega}$ be an infinite formula trace on an infinite branch of T. There is a formula $\psi = \psi_1 \mathsf{O} \psi_2 \in \mathsf{Clos}(\varphi)$ and some $j < \omega$ such that for every $k \geq j$ we have $\varphi_k \in \{\psi, \mathsf{X}\psi\}$. Moreover, both ψ and $\mathsf{X}\psi$ occur infinitely often in $(\varphi_{j+i})_{i<\omega}$.*

Another consequence of Lemma 3.4 is that there cannot be a formula trace of the form $\varphi \triangleright \cdots \triangleright \mathsf{Q}\varphi$, whence it follows that circular sequent traces must be stable.

▶ **Proposition 3.6.** *Let $(\mathsf{Q}_i \Phi_i)_{i<\omega}$ be an infinite sequent trace. There is some $j < \omega$ such that $\mathsf{Q}_{j+k} = \mathsf{Q}_j$ for every $k < \omega$.*

Given an infinite sequent trace $\tau = (\mathsf{Q}_i \Phi_i)_{i<\omega}$, we say that τ is *of type* A (E) if there is a $j < \omega$ such that $\mathsf{Q}_k = \mathsf{A}$ for all $k \geq j$ (resp., $\mathsf{Q}_k = \mathsf{E}$). Proposition 3.6 then says that every infinite sequent trace is either of type A or of type E. Similarly, an infinite formula trace $\rho = (\varphi_i)_{i<\omega}$ is *of type* U (R) if the operator O given by Proposition 3.5 is the until operator (resp., the release operator). We call $\psi_1 \mathsf{O} \psi_2$ the *dominating* formula in ρ.

We are now ready to identify the CTL^*_∞ derivations that constitute *proofs*. Informally, a derivation T is a proof if every leaf of T is axiomatic and every infinite branch of T contains a "good" sequent trace.

▶ **Definition 3.7** (Ill-founded proof). *A* CTL^*_∞ proof *of a formula* φ *is a* CTL^*_∞ *derivation* T *of* φ *satisfying:*

1. *Every leaf of* T *is labelled by an instance of an axiom.*
2. *On every infinite branch of* T *there is an infinite sequent trace* τ *such that either*
   a. τ *is of type* E *and contains no infinite formula trace of type* U*, or*
   b. τ *is of type* A *and contains an infinite formula trace of type* R*.*

The next section is devoted to showing that φ is valid iff there exists a CTL^*_∞ proof of φ.

4 Soundness and completeness of CTL^*_∞

To prove that the system CTL^*_∞ is sound we work with *signatures*, maps that assign natural numbers to until and release formulas. We follow [8], where signatures are called *indices*.

For every $n < \omega$ we define the *n-th approximation* $\varphi\mathsf{U}^n\psi$ of a formula $\varphi\mathsf{U}\psi$ by setting $\varphi\mathsf{U}^0\psi := \psi$ and $\varphi\mathsf{U}^{n+1}\psi := \psi \vee (\varphi \wedge \mathsf{X}(\varphi\mathsf{U}^n\psi))$. Dually, we define the *n-th approximation* $\varphi\mathsf{R}^n\psi$ of $\varphi\mathsf{R}\psi$ as $\varphi\mathsf{R}^n\psi := \neg(\neg\varphi\mathsf{U}^n\neg\psi)$. Clearly, $\sigma \models \varphi\mathsf{U}\psi$ iff $\sigma \models \varphi\mathsf{U}^n\psi$ for some $n < \omega$, and dually $\sigma \models \varphi\mathsf{R}\psi$ iff $\sigma \models \varphi\mathsf{R}^n\psi$ for all $n < \omega$.

An occurrence in φ of a subformula $\psi_1\mathsf{O}\psi_2$ is said to be an O-*eventuality* of φ. An O-eventuality of φ is *top-level* if it is not under the scope of U, R, A or E in φ. An O-eventuality of a set of formulas Φ is an O-eventuality of $\bigwedge\Phi$. We borrow this terminology from [8].

▶ **Definition 4.1.** *An* O-signature *of a formula* φ *is a map* ι *associating a natural number to each top-level* O*-eventuality of* φ*. An* O-signature *of a sequent* $\mathsf{Q}\Phi$ *is an* O*-signature of the formula* $\bigwedge\Phi$*.*

Given an O-signature ι of φ, the O-signature ι^- of φ is defined as $\iota^-(\psi_1\mathsf{O}\psi_2) := \max\{\iota(\psi_1\mathsf{O}\psi_2) - 1, 0\}$ for each top-level O-eventuality $\psi_1\mathsf{O}\psi_2$ of φ. We inductively define *signed formulas* $\varphi[\iota]$, with ι an O-signature of φ:

- $\lambda[\iota] := \lambda$ for every literal formula λ.
- $(\psi_1 \star \psi_2)[\iota] := (\psi_1[\iota]) \star (\psi_2[\iota])$ for $\star \in \{\wedge, \vee\}$.
- $(\mathsf{Q}\psi)[\iota] := \mathsf{Q}\psi$ for $\mathsf{Q} \in \{\mathsf{A}, \mathsf{E}\}$.
- $(\psi_1\mathsf{O}\psi_2)[\iota] := \psi_1\mathsf{O}^{\iota(\psi_1\mathsf{O}\psi_2)}\psi_2$.
- $(\psi_1\mathsf{O}'\psi_2)[\iota] := \psi_1\mathsf{O}'\psi_2$ for $\mathsf{O}' \neq \mathsf{O}$.
- $(\mathsf{X}\psi)[\iota] := \mathsf{X}(\psi[\iota^-])$.

A *signed sequent* is one of the form $\mathsf{Q}\Phi[\iota] := \mathsf{Q}\{\varphi[\iota] : \varphi \in \Phi\}$ where ι is a signature of $\mathsf{Q}\Phi$.

The following two fundamental results about existence of signatures are immediate.

▶ **Proposition 4.2.** *If* $\sigma \not\models \mathsf{A}\Phi$*, then there is an* R*-signature* ι *of* $\mathsf{A}\Phi$ *such that* $\sigma \not\models \mathsf{A}\Phi[\iota]$*.*

▶ **Proposition 4.3.** *The following hold, where* ι *is an* R*-signature of the appropriate sequent.*

1. *If* $\sigma \not\models \mathsf{A}\{\Phi, \lambda\}[\iota]$ *and* λ *is a literal formula, then* $\sigma \not\models \mathsf{A}\Phi[\iota]$ *and* $\sigma \not\models \lambda$*.*
2. *If* $\sigma \not\models \mathsf{A}\{\Phi, \varphi \vee \psi\}[\iota]$*, then* $\sigma \not\models \mathsf{A}\{\Phi, \varphi, \psi\}[\iota]$*.*
3. *If* $\sigma \not\models \mathsf{A}\{\Phi, \varphi \wedge \psi\}[\iota]$*, then either* $\sigma \not\models \mathsf{A}\{\Phi, \varphi\}[\iota]$ *or* $\sigma \not\models \mathsf{A}\{\Phi, \psi\}[\iota]$*.*
4. *If* $\sigma \not\models \mathsf{A}\{\Phi, \mathsf{Q}\psi\}[\iota]$*, for* $\mathsf{Q} \in \{\mathsf{A}, \mathsf{E}\}$*, then* $\sigma \not\models \mathsf{A}\Phi[\iota]$*.*
5. *If* $\sigma \not\models \mathsf{A}\{\Phi, \varphi\mathsf{R}\psi\}[\iota]$ *then there is an* R*-signature* ι' *which agrees with* ι *on all top-level* R*-eventualities of* $\Phi \cup \{\varphi\mathsf{R}\psi\}$ *and either* $\sigma \not\models \mathsf{A}\{\Phi, \psi\}[\iota']$ *or* $\sigma \not\models \mathsf{A}\{\Phi, \varphi, \mathsf{X}(\varphi\mathsf{R}\psi)\}[\iota']$*.*
6. *If* $\sigma \not\models \mathsf{A}\{\Phi, \varphi\mathsf{U}\psi\}[\iota]$ *then there is an* R*-signature* ι' *which agrees with* ι *on all top-level* R*-eventualities of* $\Phi \cup \{\varphi\mathsf{U}\psi\}$ *and either* $\sigma \not\models \mathsf{A}\{\Phi, \varphi, \psi\}[\iota']$ *or* $\sigma \not\models \mathsf{A}\{\Phi, \psi, \mathsf{X}(\varphi\mathsf{U}\psi)\}[\iota']$*.*
7. *If* $\sigma \not\models \mathsf{AX}\Phi[\iota]$*, then there is a* $\sigma' \sim \sigma$ *such that* $(\sigma', 1) \not\models \mathsf{A}\Phi[\iota^-]$*.*

We are now ready to prove that the system CTL^*_∞ is sound. Some details are omitted for brevity.

▶ **Proposition 4.4** (Soundness). *If there is a* CTL^*_∞ *proof of* φ*, then* φ *is valid.*

Proof. Let T be a CTL^*_∞ proof of φ. Towards a contradiction, suppose φ is not valid. Let $\mathcal{S}$ be an LTS and σ a path on $\mathcal{S}$ such that $\sigma \not\models \mathsf{A}\varphi$. We inductively find an infinite branch $\pi = (s_i)_{i<\omega}$ on T and paths $(\sigma_i)_{i<\omega}$ on $\mathcal{S}$ such that $\sigma_i \not\models \Gamma_i$ where Γ_i is the label of s_i. Propositions 4.2 and 4.3 associate to each sequent $\mathsf{A}\Phi \in \Gamma_i$ an R-signature ι such that $\sigma_i \not\models \mathsf{A}\Phi[\iota]$. Choices at $\{\mathsf{A}\wedge, \mathsf{AU}, \mathsf{AR}\}$-vertices are resolved by Proposition 4.3. Choices at $\{\mathsf{ELit}, \mathsf{EA}, \mathsf{EE}\}$-vertices are resolved by picking the premise with simpler active sequent whenever possible. Moreover, either $\sigma_{i+1} = \sigma_i$, if s_i is not modal, or else $\sigma_{i+1} = (\sigma', 1)$ for some $\sigma' \sim \sigma_i$.

For brevity we only consider the inductive case where the rule AX is applied at vertex s_i. So assume that at s_i we have:

$$\mathsf{AX}\ \frac{\mathsf{A}\Phi_j, \mathsf{E}\Psi_1, \dots, \mathsf{E}\Psi_m}{\mathsf{AX}\Phi_1, \dots, \mathsf{AX}\Phi_n, \mathsf{EX}\Psi_1, \dots, \mathsf{EX}\Psi_m, \Delta}$$

By the inductive hypothesis, σ_i refutes every sequent in the label of s_i. In particular, $\sigma_i \not\models \mathsf{AX}\Phi_j[\iota]$ where ι is the signature given by the inductive hypothesis. By Proposition 4.3, there is a path $\sigma' \sim \sigma_i$ such that $(\sigma', 1) \not\models \mathsf{A}\Phi_j[\iota^-]$. It is easy to see that $(\sigma', 1) \not\models \mathsf{E}\Psi_k$ for any $1 \le k \le m$. We then let s_{i+1} be the premise of s_i and $\sigma_{i+1} := (\sigma', 1)$. To the sequent $\mathsf{A}\Phi_j$ assign signature ι^-.

We claim that the existence of the infinite branch π contradicts the fact that T is a proof. The choice of signatures guarantees that there is no infinite sequent trace of type A on π containing an infinite formula trace of type R, as otherwise the signatures assigned to the occurrences of the dominating formula would yield an infinite descending chain of natural numbers.

Therefore, since T is a proof π must contain an infinite sequent trace $\tau = (\mathsf{Q}_i\Phi_i)_{i<\omega}$ of type E in which there is no infinite formula trace of type U. Let then $\rho = (\varphi_i)_{i<\omega}$ be an infinite formula trace through τ of type R (at least one exists by Kőnig's lemma). Let $\psi = \psi_1 \mathsf{R} \psi_2$ be the dominating formula in ρ and let $N < \omega$ be such that $\varphi_N = \psi$, $\varphi_{N+1} = \mathsf{X}\psi$, and $\varphi_i \in \{\psi, \mathsf{X}\psi\}$ for all $i \ge N$.

For each $i \ge N$, let $h(i) \ge i$ be least such that rule AX or EX is applied at $s_{h(i)}$. By construction, we have $\sigma_{h(i)+1} = (\sigma', 1)$ for some $\sigma' \sim \sigma_i$, so $\sigma_{h(i)+1}(0)$ is an immediate successor of $\sigma_i(0)$. Let σ^* be the path $\sigma_N(0)\, \sigma_{h(N)+1}(0)\, \sigma_{h(h(N)+1)+1}(0) \cdots$ through $\mathcal{S}$.

We inductively define a function $g\colon \{N, N+1, \dots\} \to \omega$ such that $\sigma_i \sim (\sigma^*, g(i))$ for every $i \ge N$. To that end, let $g(N) = g(N+1) = \cdots = g(h(N)) = 0$ and assume that g is defined on $N, N+1, \dots, h(i)$; set $g(h(i)+1) = \cdots = g(h(h(i)+1)) = g(h(i)) + 1$. The subtrace $(\varphi_i)_{i \ge N}$ then has the following form, where the numbers below the formulas are the indices and the ones below the braces are the g-images of the indices:

$$\underbrace{\underset{N}{\psi} \rhd \underset{N+1}{\mathsf{X}\psi} \rhd \cdots \rhd \underset{h(N)}{\mathsf{X}\psi}}_{0} \rhd \underbrace{\underset{h(N)+1}{\psi} \rhd \cdots \rhd \psi \rhd \mathsf{X}\psi \rhd \cdots \rhd \underset{h(h(N)+1)}{\mathsf{X}\psi}}_{1} \rhd \psi \rhd \cdots$$

Since $\sigma_N \not\models \mathsf{E}\Phi_N$ and $\sigma^* \sim \sigma_N$, we have $\sigma^* \not\models \bigwedge \Phi_N$. To reach a contradiction we show, by induction on χ, that if $\chi \in \Phi_i$, $i \ge N$, then $(\sigma^*, g(i)) \models \chi$. In particular, $\sigma^* \models \bigwedge \Phi_N$.

For brevity we present only the inductive case where $\chi = \chi_1 \mathsf{U} \chi_2$. We show the existence of an $n < \omega$ such that $(\sigma^*, g(i)+n) \models \chi_2$ and $(\sigma^*, g(i)+m) \models \chi_1$ for all $m < n$. The fact that there is no infinite formula trace of type U on τ and a context extraction is never

encountered along $(\varphi_k)_{k\geq N}$, means that there is a least $j \geq i$ such that $\chi_1 \mathsf{U} \chi_2$ is principal in s_j with unique active formula χ_2 in s_{j+1}. Let n be such that $g(j) = g(j+1) = g(i) + n$. By the inductive hypothesis, $(\sigma^*, g(i)+n) \models \chi_2$. Let $m < n$. Then, $g(i) \leq g(i) + m < g(j)$, so there is a least $i \leq k < j$ such that $g(k) = g(i) + m$. Since $g(k) < g(j)$, there is an instance of AX or EX in between s_k and s_j. So, by the minimality of j, there is a $k \leq k' < h(k)$ such that $\Phi_{k'} \ni \chi \rhd \chi_1, \mathsf{X}\chi \in \Phi_{k'+1}$, whence $(\sigma^*, g(k'+1)) \models \chi_1$ by the inductive hypothesis. And $g(k'+1) = g(k') = g(k) = g(i) + m$, so $(\sigma^*, g(i)+m) \models \chi_1$. ◀

Now we turn to the proof that the system CTL^*_∞ is complete. We argue game-theoretically and appeal to the soundness and completeness result for satisfiability of CTL^* formulas in [12].

▶ **Definition 4.5.** *A* proof-search tree *for a formula φ is a finite or infinite tree T whose vertices are labelled according to the rules on Figure 1 and such that*

1. *The root of T has label $\mathsf{A}\varphi$.*
2. *A vertex of T is a leaf iff it is either axiomatic or a set of literal sequents.*
3. *Every instance of rule EX in T is of the form*

$$\mathsf{EX}\ \frac{\mathsf{E}\Psi_1, \ldots, \mathsf{E}\Psi_m}{\mathsf{EX}\Psi_1, \ldots, \mathsf{EX}\Psi_m, \Lambda}$$

   where Λ is a non-axiomatic set of literal sequents.
4. *In place of rule AX, a branching rule AX_b is used in T:*

$$\mathsf{AX}_b\ \frac{\mathsf{A}\Phi_1, \mathsf{E}\Psi_1, \ldots, \mathsf{E}\Psi_m \qquad \cdots \qquad \mathsf{A}\Phi_n, \mathsf{E}\Psi_1, \ldots, \mathsf{E}\Psi_m}{\mathsf{AX}\Phi_1, \ldots, \mathsf{AX}\Phi_n, \mathsf{EX}\Psi_1, \ldots, \mathsf{EX}\Psi_m, \Lambda}$$

   where again Λ is a non-axiomatic set of literal sequents.
5. *Every infinite path of T contains infinitely many applications of AX or EX.*

Our aim is to show that any proof-search tree for φ contains either a CTL^*_∞ proof of φ or else a *refutation* of φ, a subtree from which the satisfiability of $\neg\varphi$ follows.

▶ **Definition 4.6** (Refutation). *A* refutation *of a formula φ is a subtree T' of a proof-search tree T for φ satisfying*

1. *Every leaf of T' is labelled by a non-axiomatic set of literal sequents.*
2. *If a vertex $s \in T'$ is obtained in T by an application of a rule other than AX_b, then s has exactly one immediate successor in T'.*
3. *If a vertex $s \in T'$ is obtained in T by an application of rule AX_b, then T' contains every immediate successor of s in T.*
4. *On every infinite sequent trace τ in T'*
   a. *if τ is of type A, then every infinite formula trace is of type U;*
   b. *if τ is of type E, then some infinite formula trace is of type U.*

The term *refutation*, which we borrow from [20], is justified by the following proposition, dual to the soundness and completeness result in [12]:

▶ **Proposition 4.7** ([12, Thm. 10]). *A formula φ is valid iff there is no refutation of φ.*

To finish the proof of completeness we set up a game for two players whose arena is a proof-search tree in which one of the players looks for a proof and the other one for a refutation. Determinacy of the game then yields completeness of CTL^*_∞.

▶ **Definition 4.8.** *Let T be a proof-search tree for φ. We define the game $\mathcal{G}(\varphi, T)$ with arena T and players* Prov *and* Ref *as follows.*

1. *The starting position is the root of T.*
2. Prov *owns all modal vertices.*
3. Ref *owns every branching non-modal vertex.*
4. *A finite play is won by* Prov *if it is maximal and the last position is labelled by an instance of an axiom; otherwise* Ref *wins.*
5. *An infinite play is won by* Prov *if it contains an infinite sequent trace τ such that either*
   a. *τ is of type* E *and contains no infinite formula trace of type* U*, or*
   b. *τ is of type* A *and contains an infinite formula trace of type* R*.*

   Otherwise Ref *wins.*

Informally, Prov attempts to find a (branch of a) proof within T, whereas Ref tries to find a (branch of a) refutation.

▶ **Proposition 4.9.** *There is a winning strategy for* Prov (Ref) *in $\mathcal{G}(\varphi, T)$ iff T contains a CTL^*_∞ proof of φ (resp., a refutation of φ).*

Proof. It is clear from Definition 4.8 that a winning strategy for Prov (Ref) determines a subtree of T which is a CTL^*_∞ proof of φ (resp., a refutation). Conversely, let T' be a CTL^*_∞ proof of φ (resp. a refutation) contained in T. Then, Prov (resp., Ref) has the following winning strategy: move so as to always remain inside T'. ◀

As the winning condition in $\mathcal{G}(\varphi, T)$ is Borel, Martin's theorem [18] implies the games are always determined, whence completeness of CTL^*_∞ follows.

▶ **Proposition 4.10** (Completeness)**.** *If φ is valid, then there is a CTL^*_∞ proof of φ.*

Proof. Let T be a proof-search tree for φ (one always exists). Since φ is valid, Proposition 4.9 and Proposition 4.7 ensure that Ref cannot have a winning strategy in $\mathcal{G}(\varphi, T)$. By determinacy, there is a winning strategy for Prov and thus φ is provable by Proposition 4.9. ◀

Combining Proposition 4.4 and Proposition 4.10:

▶ **Theorem 4.11.** *A formula φ is valid iff there is a CTL^*_∞ proof of φ.*

CTL^*_∞ was formulated without any internal (i.e., formula level) or external (sequent level) structural rules except for the implicit external weakening that may accompany applications of AX, EX. Contraction (at both levels) is implicit in our choice of sets rather than multisets or sequences. The external and internal weakening rules are, respectively,

$$\mathsf{eW}\ \frac{\mathsf{Q}\Phi, \Delta}{\mathsf{Q}\Phi, \mathsf{Q}'\Psi, \Delta} \qquad \mathsf{iW}\ \frac{\mathsf{A}\Phi, \Delta}{\mathsf{A}\{\Phi \cup \Psi\}, \Delta}$$

Note, the internal weakening rule is only sound for A sequents. Soundness and completeness confirms both rules are admissible in CTL^*_∞. When working with cyclic proofs, however, it will be convenient to have at our disposal the external weakening as an explicit inference; internal weakening can be included, but is not necessary for completeness.

In the rule eW above, both $\mathsf{Q}\Phi$ and $\mathsf{Q}'\Psi$ are considered principal in the conclusion and $\mathsf{Q}\Phi$ is the only active sequent in the premise. As in the modal rules we impose that $\mathsf{Q}'\Psi \not\trianglerighteq \mathsf{Q}''\Xi$ for any sequent $\mathsf{Q}''\Xi$ in the premise.

We denote by $\mathsf{CTL}^*_\infty + \mathsf{eW}$ the proof system resulting by adding eW to CTL^*_∞.

▶ **Proposition 4.12.** *There is a CTL^*_∞ proof of φ iff there is a $\mathsf{CTL}^*_\infty + \mathsf{eW}$ proof of φ.*

5 Cyclic proofs

We now introduce a cyclic version of the system CTL^*_∞. Formulas are annotated in the style of [14, 30] to keep track of fixpoint unfoldings and determine the existence of "good" traces on cycles. Whereas a CTL^*_∞ derivation is represented as a possibly infinite tree, a $\mathsf{CTL}^*_\circ$ derivation is a finite *tree with back-edges*, that is, a pair $(T, l \mapsto c_l)$ where T is a finite tree and $l \mapsto c_l$ is a partial function defined on a subset Rep_T of the leaves of T such that each $l \in \mathsf{Rep}_T$ is mapped to a vertex $c_l <_T l$. We call c_l the *companion* of l, and leaves in Rep_T are called *repeats*. For repeats l and l', we say that l' is *reachable* from l, in symbols $l \lhd l'$, if $c_l <_T l'$. Given a tree with back-edges T, we denote by T° the result after adding an edge from each repeat $l \in \mathsf{Rep}_T$ to its companion c_l. The sequence of vertices visited by an infinite branch on T° is always of the form

$$[r, l_0]_T + [c_0, l_1]_T + \cdots + [c_n, l_{n+1}]_T + \cdots$$

where r is the root of T, each l_i is a repeat with companion c_i, and "+" denotes sequence concatenation. The following result ensures that a path through T° which visits a repeat l more than once also passes through every vertex in $[c_l, l]_T$.

▶ **Proposition 5.1.** *Let $(T, l \mapsto c_l)$ be a tree with back-edges and π a path through T°. If there are $0 \leq m < n$ such that $\pi(m) = l$ and $\pi(n) = l'$ for repeats $l, l' \in \mathsf{Rep}_T$ such that $l \lhd l'$, then $[c_l, l']_T \subseteq \{\pi(k) : m \leq k \leq n\}$.*

Fix a formula φ. To each eventuality $\psi_1 \mathsf{O} \psi_2$ in φ we associate a unique *identifier* X and write $\psi_1 \mathsf{O}^X \psi_2$. We say that X is an O-*identifier* if the eventuality corresponding to X is an O-formula. For each identifier X we assume a countably infinite set $\mathsf{N}_X = \{x_0, x_1, \dots\}$ of *names* for X. A name $x \in \mathsf{N}_X$ is an O-*name* if X is an O-identifier. In the sequel, eventualities with different identifiers will be considered as different subformulas.

An *annotated formula* is a pair (ψ, u), henceforth written ψ^u, where ψ is a formula and u is either the empty string or a name for an identifier in ψ. We call u an *annotation*. An *annotated sequent* is an expression of the form $\mathsf{Q}\Phi$, where $\mathsf{Q} \in \{\mathsf{A}, \mathsf{E}\}$ and Φ is a finite set of annotated formulas. An *annotated hypersequent* is an expression of the form $\Theta : \Gamma$, where Γ is a finite set of annotated sequents and Θ is a linear ordering of the names occurring in Γ. We call Θ the *control* of $\Theta : \Gamma$.

Given a finite sequence of names Θ, we define the following strict linear order $\prec_\Theta$ on the collection of annotations contained in Θ: $u \prec_\Theta v$ if either $v = \varnothing$ and $u \neq \varnothing$, or both u and v are non-empty and the name in u occurs in Θ strictly before the name in v.

▶ **Definition 5.2** (Cyclic derivation). *A $\mathsf{CTL}^*_\circ$ derivation of a formula φ is a finite tree with back-edges $(T, l \mapsto c_l)$ whose vertices are labelled according to the rules in Figures 2 and 3 and satisfying*

1. *The root of T has label $\mathsf{A}\varphi$.*
2. *Every leaf not labelled by an instance of an axiom is a repeat and has the same label as its companion.*
3. *For every $l \in \mathsf{Rep}_T$, there is an instance of rule AX or EX in $[c_l, l]_T$.*
4. *In rules AR and EU, if $u = \varnothing$ then x is the first name for X not already occurring in Θ. In ER_0, either $u = \varnothing$, in which case $\Theta u = \Theta$, or u is the first name for X not already occurring in Θ.*
5. *Rule iThin has priority over other rules: If a hypersequent in T can be witnessed as the conclusion of an application of iThin, then it is.*

$$\mathsf{ax}_p\ \frac{}{\Theta : \mathsf{Q}p, \mathsf{Q}'\neg p, \Delta} \qquad \mathsf{ax}_\top\ \frac{}{\Theta : \top, \Delta}$$

$$\mathsf{ALit}\ \frac{\Theta : \mathsf{A}\Phi, \mathsf{A}\lambda, \Delta}{\Theta : \mathsf{A}\{\Phi, \lambda\}, \Delta} \qquad \mathsf{ELit}\ \frac{\Theta : \mathsf{E}\Phi, \Delta \qquad \Theta' : \mathsf{E}\lambda, \Delta}{\Theta : \mathsf{E}\{\Phi, \lambda\}, \Delta}$$

$$\mathsf{A}\vee\ \frac{\Theta : \mathsf{A}\{\Phi, \varphi, \psi\}, \Delta}{\Theta : \mathsf{A}\{\Phi, \varphi \vee \psi\}, \Delta} \qquad \mathsf{E}\vee\ \frac{\Theta : \mathsf{E}\{\Phi, \varphi\}, \mathsf{E}\{\Phi, \psi\}, \Delta}{\Theta : \mathsf{E}\{\Phi, \varphi \vee \psi\}, \Delta}$$

$$\mathsf{A}\wedge\ \frac{\Theta : \mathsf{A}\{\Phi, \varphi\}, \Delta \qquad \Theta : \mathsf{A}\{\Phi, \psi\}, \Delta}{\Theta : \mathsf{A}\{\Phi, \varphi \wedge \psi\}, \Delta} \qquad \mathsf{E}\wedge\ \frac{\Theta : \mathsf{E}\{\Phi, \varphi, \psi\}, \Delta}{\Theta : \mathsf{E}\{\Phi, \varphi \wedge \psi\}, \Delta}$$

$$\mathsf{AA}\ \frac{\Theta : \mathsf{A}\Phi, \mathsf{A}\{\psi\}, \Delta}{\Theta : \mathsf{A}\{\Phi, \mathsf{A}\psi\}, \Delta} \qquad \mathsf{EA}\ \frac{\Theta : \mathsf{E}\Phi, \Delta \qquad \Theta' : \mathsf{A}\{\psi\}, \Delta}{\Theta : \mathsf{E}\{\Phi, \mathsf{A}\psi\}, \Delta}$$

$$\mathsf{AE}\ \frac{\Theta : \ \mathsf{A}\Phi, \mathsf{E}\{\psi\}, \Delta}{\Theta : \mathsf{A}\{\Phi, \mathsf{E}\psi\}, \Delta} \qquad \mathsf{EE}\ \frac{\Theta : \mathsf{E}\Phi, \Delta \qquad \Theta' : \mathsf{E}\{\psi\}, \Delta}{\Theta : \mathsf{E}\{\Phi, \mathsf{E}\psi\}, \Delta}$$

$$\mathsf{AX}\ \frac{\Theta' : \mathsf{A}\Phi_i, \mathsf{E}\Psi_1, \dots, \mathsf{E}\Psi_m}{\Theta : \mathsf{AX}\Phi_1, \dots, \mathsf{AX}\Phi_n, \mathsf{EX}\Psi_1, \dots, \mathsf{EX}\Psi_m, \Sigma} \qquad \mathsf{EX}\ \frac{\Theta' : \mathsf{E}\Psi_1, \dots, \mathsf{E}\Psi_m}{\Theta : \mathsf{EX}\Psi_1, \dots, \mathsf{EX}\Psi_m, \Sigma}$$

$$\mathsf{iThin}\ \frac{\Theta' : \mathsf{Q}\{\Phi, \varphi^x\}, \Delta}{\Theta : \mathsf{Q}\{\Phi, \varphi^x, \varphi^v\}, \Delta}\ x \prec_\Theta v \qquad \mathsf{eW}\ \frac{\Theta' : \mathsf{Q}\Phi, \Delta}{\Theta : \mathsf{Q}\Phi, \mathsf{Q}'\Psi, \Delta}$$

Figure 2 Non-fixpoint rules of system $\mathsf{CTL}^*_\circ$. In ALit and ELit, λ is a literal formula. In all rules, Θ' is the result of removing from Θ all names not occurring in the associated hypersequent.

For clarity of presentation the ER inference is split into two rules, depending on whether the principal formula is annotated. We refer to the two rules ER_0 and ER_1 jointly as ER. Sequent traces, as well as principal and active sequents and formulas, follow the definition from the unannotated calculus. In the case of iThin, both φ^x and φ^v are principal and we let $\varphi^x \rhd \varphi^x$ and $\varphi^v \rhd \varphi^x$.

The prioritisation of iThin in the definition above is not necessary and the notion of a cyclic proof given below is sound and complete without this restriction. However, it ensures that all sequents in a derivation are bounded in size. Judicious use of external weakening in the form of an external *thinning* rule can be used to ensure a bound on the size of hypersequents also. We isolate a particular form of the rule eW that will be useful in proving completeness. This is rule

$$\mathsf{eThin}\ \frac{\Theta' : \mathsf{Q}\{\varphi_0^{u_0}, \dots, \varphi_n^{u_n}\}, \Delta}{\Theta : \mathsf{Q}\{\varphi_0^{u_0}, \dots, \varphi_n^{u_n}\}, \mathsf{Q}\{\varphi_0^{v_0}, \dots, \varphi_n^{v_n}\}, \Delta}$$

with the restriction that some name u_i precedes all the names $v_0, \dots, v_n$, i.e., for some $i \leq n$, we have $u_i \prec_\Theta v_j$ for every $j \leq n$. The condition ensures that the first name in Θ distinguishing the sequent $\mathsf{Q}\{\varphi_0^{u_0}, \dots, \varphi_n^{u_n}\}$ from $\mathsf{Q}\{\varphi_0^{v_0}, \dots, \varphi_n^{v_n}\}$ occurs in $\{u_0, \dots, u_n\}$ and is preserved in the premise.

Prioritising applications of eThin alongside iThin ensures that sequents and hypersequents never grow past a finite bound. More precisely, given a $\mathsf{CTL}^*_\infty + \mathsf{eW}$ derivation T, let $\mathsf{ann}(T)$ be the result after annotating the hypersequents in T according to the rules of $\mathsf{CTL}^*_\circ$ and applying rules iThin and eThin whenever possible. We then have

$$\mathsf{AU}\ \frac{\Theta : \mathsf{A}\{\Phi, \varphi_1, \varphi_2\}, \Delta \qquad \Theta : \mathsf{A}\{\Phi, \varphi_2, \mathsf{X}(\varphi_1 \mathsf{U} \varphi_2)\}, \Delta}{\Theta : \mathsf{A}\{\Phi, \varphi_1 \mathsf{U} \varphi_2\}, \Delta}$$

$$\mathsf{AR}\ \frac{\Theta' : \mathsf{A}\{\Phi, \varphi_2\}, \Delta \qquad \Theta x : \mathsf{A}\{\Phi, \varphi_1, \mathsf{X}(\varphi_1 \mathsf{R}^X \varphi_2)^x\}, \Delta}{\Theta : \mathsf{A}\{\Phi, (\varphi_1 \mathsf{R}^X \varphi_2)^u\}, \Delta}\ u \in \{\varnothing, x\}$$

$$\mathsf{EU}\ \frac{\Theta x : \mathsf{E}\{\Phi, \varphi_2\}, \mathsf{E}\{\Phi, \varphi_1, \mathsf{X}(\varphi_1 \mathsf{U}^X \varphi_2)^x\}, \Delta}{\Theta : \mathsf{E}\{\Phi, (\varphi_1 \mathsf{U}^X \varphi_2)^u\}, \Delta}\ u \in \{\varnothing, x\}$$

$$\mathsf{ER}_0\ \frac{\Theta u : \mathsf{E}\{\Phi, \varphi_1, \varphi_2\}, \mathsf{E}\{\Phi, \varphi_2, \mathsf{X}(\varphi_1 \mathsf{R}^X \varphi_2)^u\}, \Delta}{\Theta : \ \mathsf{E}\{\Phi, \varphi_1 \mathsf{R}^X \varphi_2\}, \Delta}\ u \in \{\varnothing\} \cup \mathsf{N}_X$$

$$\mathsf{ER}_1\ \frac{\Theta' : \mathsf{E}\{\Phi, \varphi_1, \varphi_2\}, \mathsf{E}\{\Phi, \varphi_2, \mathsf{X}(\varphi_1 \mathsf{R}^X \varphi_2)^u\}, \Delta}{\Theta : \ \mathsf{E}\{\Phi, (\varphi_1 \mathsf{R}^X \varphi_2)^x\}, \Delta}\ u \in \{\varnothing, x\}$$

Figure 3 Fixpoint rules of system $\mathsf{CTL}^*_\circ$. In all rules, Θ' denotes the result of removing from Θ all names not occurring in the associated hypersequent. The control Θx denotes Θ if $x \in \Theta$ and the concatenation of Θ and x otherwise. Rule ER_0 is subject to the restriction: u is either $\varnothing$ or a name for X not occurring in Θ.

▶ **Proposition 5.3.** *There are only finitely many distinct annotated hypersequents in* $\mathsf{ann}(T)$.

A name is *fixed* on a path π through a $\mathsf{CTL}^*_\circ$ derivation if it occurs in every control in π. Similarly, a name is *fixed* on a sequent (formula) trace if it occurs in each sequent (resp., formula) on the trace.

▶ **Definition 5.4** (Good trace). *Let T be a $\mathsf{CTL}^*_\circ$ derivation, and let τ be a (finite) stable sequent trace on a path through T. We say that τ is* good *if the following hold.*

1. *There is an* R*-name fixed on τ.*
2. *No* U*-name is fixed on τ.*

Otherwise τ is said to be bad.

▶ **Definition 5.5** (Successful repeat). *Let T be a $\mathsf{CTL}^*_\circ$ derivation. A repeat $l \in \mathsf{Rep}_T$ is* successful *if the following hold.*

1. *There is a good sequent trace on $[c_l, l]_T$.*
2. *No* R*-name is fixed on a bad trace on $[c_l, l]_T$.*

In other words, l is successful if there is a good trace on $[c_l, l]_T$ and, moreover, the path contains no E-trace on which an R-name and a U-name are both fixed. Observe that the rule EU always annotates the active U operator and the only rules eliminating U-names along sequent traces are the thinning rules; on the contrary, the rule ER_1 provides a mechanism for eliminating R-names along a path. Thus, on a successful repeat, if EU affects an E-trace between companion and leaf, then all R-names on the trace must eventually be eliminated (cf. Lemma 6.6 below).

▶ **Definition 5.6** (Cyclic proof). *A* $\mathsf{CTL}^*_\circ$ proof *of a formula φ is a $\mathsf{CTL}^*_\circ$ derivation T of φ each of whose leaves is either axiomatic or else a successful repeat.*

Figure 4 contains a CTL^*_∞ proof of the valid formula $(\neg p \mathsf{U} p) \vee (\bot \mathsf{R} \neg p)$ together with its cyclic version. The unique infinite branch in the ill-founded proof corresponds to the unique cycle in the cyclic one.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\mathsf{X}(\bot\mathsf{R}\neg p)\},\mathsf{A}p,\mathsf{A}\neg p}}{\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\neg p,\mathsf{X}(\bot\mathsf{R}\neg p)\},\mathsf{A}p} \quad \dfrac{\vdots}{\mathsf{A}\{\neg p\mathsf{U}p,\bot\mathsf{R}\neg p\}}}{\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\mathsf{X}(\bot\mathsf{R}\neg p)\},\mathsf{A}p}}{\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),p,\mathsf{X}(\bot\mathsf{R}\neg p)\},\mathsf{A}p}}{\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\bot,\mathsf{X}(\bot\mathsf{R}\neg p)\},\mathsf{A}p}}{\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\bot\mathsf{R}\neg p\},\mathsf{A}p}}{\mathsf{A}\{p,\mathsf{X}(\neg p\mathsf{U}p),\bot\mathsf{R}\neg p\}}}{\mathsf{A}\{\neg p\mathsf{U}p,\bot\mathsf{R}\neg p\}}}{\mathsf{A}\{\neg p\mathsf{U}p\vee\bot\mathsf{R}\neg p\}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x:\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\mathsf{X}(\bot\mathsf{R}\neg p)^x\},\mathsf{A}p,\mathsf{A}\neg p}}{x:\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\neg p,\mathsf{X}(\bot\mathsf{R}\neg p)^x\},\mathsf{A}p} \quad \dfrac{(\dagger)\quad x:\mathsf{A}\{\neg p\mathsf{U}p,(\bot\mathsf{R}\neg p)^x\}}{x:\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\mathsf{X}(\bot\mathsf{R}\neg p)^x\},\mathsf{A}p}}{x:\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),p,\mathsf{X}(\bot\mathsf{R}\neg p)^x\},\mathsf{A}p}}{x:\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\bot,\mathsf{X}(\bot\mathsf{R}\neg p)^x\},\mathsf{A}p}}{x:\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),(\bot\mathsf{R}\neg p)^x\},\mathsf{A}p}}{x:\mathsf{A}\{p,\mathsf{X}(\neg p\mathsf{U}p),(\bot\mathsf{R}\neg p)^x\}}}{(\dagger)\quad x:\mathsf{A}\{\neg p\mathsf{U}p,(\bot\mathsf{R}\neg p)^x\}}}{x:\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\mathsf{X}(\bot\mathsf{R}\neg p)^x\},\mathsf{A}p}}{x:\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\bot,\mathsf{X}(\bot\mathsf{R}\neg p)^x\},\mathsf{A}p}}{\mathsf{A}\{\mathsf{X}(\neg p\mathsf{U}p),\bot\mathsf{R}\neg p\},\mathsf{A}p}}{\mathsf{A}\{\neg p\mathsf{U}p\vee\bot\mathsf{R}\neg p\}}$$

Figure 4 Top: A CTL^*_∞ proof of the valid formula $(\neg p\mathsf{U}p)\vee(\bot\mathsf{R}\neg p)$ where $\bot := p\wedge\neg p$. Bottom: A $\mathsf{CTL}^*_\circ$ proof of the same formula. The unique repeat in the cyclic proof and its companion are marked with †. Double line indicates that some vertices are omitted for brevity.

In the next section we prove that φ is valid iff there exists a $\mathsf{CTL}^*_\circ$ proof of φ. We do so by showing that ill-founded proofs can be seen as unravellings of cyclic proofs.

6 Soundness and completeness of $\mathsf{CTL}^*_\circ$

Every successful repeat l in a $\mathsf{CTL}^*_\circ$ derivation T has an associated *invariant*, denoted $\mathsf{inv}(l)$, the shortest sequence of names wx such that x is an R-name witnessing condition 1 of Definition 5.5 and wx is a prefix of every control in $[c_l, l]_T$. The existence of invariants follows from the fact that new names are always appended to the right of the controls.

Invariants induce the following (reflexive) quasi-order on repeats of a proof: $l \preccurlyeq l'$ if $\mathsf{inv}(l)$ is a prefix of $\mathsf{inv}(l')$. The orders $\lhd$ and $\preccurlyeq$ are related in the sense of the following propositions, both of which are easily verified.

▶ **Proposition 6.1.** *For every infinite reachability sequence $l_0 \lhd l_1 \lhd \cdots$ there exists $k \geq 0$ such that $l_k \preccurlyeq l_j$ for all $j \geq k$.*

▶ **Proposition 6.2.** *If $l_0 \lhd l_1 \lhd \cdots \lhd l_m \lhd l_0$ and w is a prefix of $\mathsf{inv}(l_i)$ for each $i \leq m$, then w is a prefix of each control on $[c_{l_m}, l_0]_T$.*

The following lemmas concerning names fixed on paths through cyclic derivations are used to show soundness and completeness for $\mathsf{CTL}^*_\circ$. We omit the proofs for brevity.

▶ **Lemma 6.3.** *Let $\pi = (s_i)_{i\leq n}$ be a finite path through a $\mathsf{CTL}^*_\circ$ derivation. If names $x_1, \ldots, x_m$ are fixed on π and there is a sequent $\mathsf{Q}\Phi$ in the label of s_n such that $x_1, \ldots, x_m$ all occur in $\mathsf{Q}\Phi$, then there is a sequent trace τ on π such that $x_1, \ldots, x_m$ are all fixed on τ.*

▶ **Lemma 6.4.** *Let $\pi = (s_i)_{i<\omega}$ be an infinite path through a $\mathsf{CTL}^*_\circ$ derivation. If names $x_1, \ldots, x_m$ are fixed on π and for every $i < \omega$ there is a $j > i$ and a sequent $\mathsf{Q}\Phi$ in the label of s_j such that $x_1, \ldots, x_m$ all occur in $\mathsf{Q}\Phi$, then there is an infinite sequent trace τ on π fixing $x_1, \ldots, x_m$.*

The next lemmas are immediate consequences of the success condition on $\mathsf{CTL}^*_\circ$ proofs.

▶ **Lemma 6.5.** *Let T be a $\mathsf{CTL}^*_\circ$ proof, and let π be an infinite path through T°. There is an R-name x and an infinite sequent trace on π containing an infinite formula trace on which x is fixed.*

▶ **Lemma 6.6.** *Let T be a $\mathsf{CTL}^*_\circ$ derivation, τ an infinite sequent trace on T° of type E, and ρ an infinite formula trace on τ of type U. There is a U-name eventually fixed on ρ.*

Soundness of $\mathsf{CTL}^*_\circ$ now follows easily.

▶ **Proposition 6.7** (Soundness). *If there is a $\mathsf{CTL}^*_\circ$ proof of φ, then φ is valid.*

Proof. Let T be a $\mathsf{CTL}^*_\circ$ proof of φ. We show that the trace conditions on Definition 3.7 hold for T°. By the priority assigned to rule iThin, this suffices to ensure that the (possibly infinite) tree of paths on T°, once stripped of the annotations, is a $\mathsf{CTL}^*_\infty + \mathsf{eW}$ proof of φ.

Let π be an infinite path on T°. By Lemma 6.5 there is an infinite sequent trace τ on π containing an infinite formula trace ρ on which an R-name x is eventually fixed. If τ is of type A we are done, so suppose τ is of type E. Towards a contradiction, assume that τ contains an infinite formula trace ξ of type U. By Lemma 6.6, there is a U-name y which is eventually fixed on ξ. Then, there is a tail τ' of τ such that both x and y are fixed on τ'. Let π' be the tail of π corresponding to τ', and let l be a repeat encountered infinitely often on π'. By Proposition 5.1, every vertex on $[c_l, l]_T$ occurs infinitely often on π', so x and y are fixed on $[c_l, l]_T$ and some sequent in the label of l contains both x and y. By Lemma 6.3, there is a bad trace on $[c_l, l]_T$ on which x is fixed, contradicting the fact that l is successful. ◀

We now turn to the completeness proof for $\mathsf{CTL}^*_\circ$. An application of rule iThin is *trivial* if, using the same notation as in Figure 2, $v = \varnothing$. Due to the priority given to iThin in derivations, non-trivial instances of iThin satisfy the following.

▶ **Lemma 6.8.** *Let T be a $\mathsf{CTL}^*_\circ$ derivation, and let $s \in T$ be labelled by the premise of a non-trivial instance of iThin, say:*

$$\frac{\Theta' : \mathsf{Q}\{\Phi, \varphi^{x_1}\}, \Delta}{\Theta : \mathsf{Q}\{\Phi, \varphi^{x_1}, \varphi^{x_2}\}, \Delta}$$

Then, $\varphi = \mathsf{X}\psi$ for some $\psi = \psi_1 \mathsf{O}^X \psi_2$ and there is a vertex $t <_T s$ such that there is a sequent trace on $[t, s]_T$ of the form

$$\mathsf{Q}\{\Psi, \mathsf{X}\psi^{x_1}, \psi\} \rhd \mathsf{Q}\{\Psi', \mathsf{X}\psi^{x_1}, \mathsf{X}\psi^{x_2}\} \overset{\mathsf{iThin}\ \textit{only}}{\rhd} \cdots \rhd \mathsf{Q}\{\Phi, \mathsf{X}\psi^{x_1}, \mathsf{X}\psi^{x_2}\} \rhd \mathsf{Q}\{\Phi, \mathsf{X}\psi^{x_1}\}.$$

Proof. By the priority given to iThin and the fact that x_1 occurs before x_2 in Θ. ◀

▶ **Proposition 6.9** (Completeness). *If φ is valid, then there is a $\mathsf{CTL}^*_\circ$ proof of φ.*

Proof. Let T be a CTL^*_∞ proof of φ, and let $\mathsf{ann}(T)$ be the result of annotating T in accordance with the rules of $\mathsf{CTL}^*_\circ$, applying iThin and eThin whenever possible and always choosing $u \neq \varnothing$ when applying either ER_0 or ER_1. It suffices to show that every infinite branch on $\mathsf{ann}(T)$ contains a vertex satisfying the requirements of a successful repeat. Proposition 5.3 implies each branch of $\mathsf{ann}(T)$ contains only finitely many distinct annotated hypersequents.

Let $\mathsf{ann}(\pi)$ be an infinite branch on T, where $\pi = (s_i)_{i<\omega}$ is the corresponding branch on T. Let T^+ be the collection of all infinite sequent traces τ on $\mathsf{ann}(\pi)$ such that either τ is of type A and contains an infinite formula trace of type R, or τ is of type E and contains no infinite formula trace of type U. Since T is a proof, $T^+ \neq \varnothing$. Let T^- be the collection of all infinite sequent traces on $\mathsf{ann}(\pi)$ of type E that contain an infinite formula trace of type U.

We modify the annotations on $\mathsf{ann}(\pi)$ by applying the following procedure to every trace $\tau \in T^-$. Let $\tau = (\mathsf{Q}_i\Phi_i)_{i<\omega}$, and let $\rho = (\varphi_i^{u_i})_{i<\omega}$ be an infinite formula trace on τ of type U, say with dominating formula ψ^y. Let $n < \omega$ be such that $\varphi_i^{u_i} \in \{\psi^y, \mathsf{X}\psi^y\}$ for all $i \geq n$. For every application of ER on $\pi_{\geq n} = (s_i)_{i\geq n}$ with principal sequent in τ, we remove the annotation (using instead the rule instance with $u = \varnothing$) and propagate this change upwards appropriately. Let $\tilde{\pi} = (\tilde{s}_i)_{i<\omega}$ be the result after applying this procedure to every $\tau \in T^-$.

We claim that after all these changes the result remains an (annotated) derivation. This is clear for all rules except iThin and eThin. Each application of eThin will either vanish (because the two principal sequents now coincide), or remain an instance of eW. And every trivial instance of iThin will remain so or else vanish. Finally, consider a non-trivial instance of iThin in $\mathsf{ann}(\pi)$:

$$\frac{\Theta' : \mathsf{Q}\{\Psi, \psi^{x_1}\}, \Delta}{\Theta : \mathsf{Q}\{\Psi, \psi^{x_1}, \psi^{x_2}\}, \Delta}\, x_1 \prec_\Theta x_2 \tag{1}$$

The only way for this rule application to cease to be an instance of a $\mathsf{CTL}^*_\circ$ rule after the changes performed is if x_1 has been removed and x_2 remains. We claim this is impossible. Let s be the vertex on $\mathsf{ann}(\pi)$ corresponding to the premise of (1). By Lemma 6.8, $\psi = \mathsf{X}\chi$ for some formula $\chi = \chi_1 \mathsf{R}^X \chi_2$ (since x_1 is an R-name) and there is a vertex t below s such that there is a sequent trace on $[t, s]_{\mathsf{ann}(\pi)}$ of the form

$$\mathsf{Q}\{\Sigma, \mathsf{X}\chi^{x_1}, \chi\} \rhd \mathsf{Q}\{\Sigma', \mathsf{X}\chi^{x_1}, \mathsf{X}\chi^{x_2}\} \overset{\text{iThin only}}{\rhd} \cdots \rhd \mathsf{Q}\{\Psi, \mathsf{X}\chi^{x_1}, \mathsf{X}\chi^{x_2}\} \rhd \mathsf{Q}\{\Psi, \mathsf{X}\chi^{x_1}\}.$$

So by the time x_2 is introduced x_1 has already been removed, whence by construction of $\tilde{\pi}$ we also removed x_2 and thus (1) simply vanishes on $\tilde{\pi}$.

Finally, we show that $\tilde{\pi}$ passes through a successful repeat. Fix a trace $\tau = (\mathsf{Q}_i\Phi_i)_{i<\omega} \in T^+$ (note that $\tau \notin T^-$), and let $\rho = (\varphi_i^{u_i})_{i<\omega}$ be an infinite formula trace on τ of type R, say with dominating formula ψ^x. Let $n < \omega$ be such that

1. $\varphi_i^{u_i} \in \{\psi^x, \mathsf{X}\psi^x\}$ for every $i \geq n$, and
2. every pair $(\Theta_i : \Gamma_i, \mathsf{Q}_i\Phi_i)$ for $i \geq n$ is encountered infinitely often on $\{(\Theta_j : \Gamma_j, \mathsf{Q}_j\Phi_j)\}_{j<\omega}$, where $\Theta_j : \Gamma_j$ is the label of $\tilde{s}_j$.

Let $S = \{(\mathsf{E}\Psi_i, y_i, z_i)\}_{i\in I}$ be the collection of all triples $(\mathsf{E}\Psi_i, y_i, z_i)$ where $\mathsf{E}\Psi_i$ is a sequent in the label of $\tilde{s}_n$, y_i is a U-name occurring in $\mathsf{E}\Psi_i$, and z_i is an R-name occurring in $\mathsf{E}\Psi_i$. Note that S is finite. For every $i \in I$ there is a $j_i \geq n$ such that there is no sequent trace on $[\tilde{s}_n, \tilde{s}_{j_i}]_{\tilde{\pi}}$ starting from $\mathsf{E}\Psi_i$ and where y_i and z_i are both fixed. Otherwise by Lemma 6.4 there would be an infinite sequent trace τ' on $\tilde{\pi}$ of type E such that y_i and z_i are both fixed on τ', contradicting the construction of $\tilde{\pi}$. Let $m > \max\{n, \max\{j_i : i \in I\}\}$ be such that $\tilde{s}_n$ and $\tilde{s}_m$ have identical labels. We claim that $\tilde{s}_m$ is a successful repeat with companion $\tilde{s}_n$.

Let $\tau_n^m := (\mathsf{Q}_i\Phi_i)_{n\leq i\leq m}$. We know that x is fixed on τ_n^m, and by the choice of m no U-name is fixed on τ_n^m, so τ_n^m is a good trace on $[\tilde{s}_n, \tilde{s}_m]_{\tilde{\pi}}$. Towards a contradiction, suppose there is a bad trace ξ on $[\tilde{s}_n, \tilde{s}_m]_{\tilde{\pi}}$ such that an R-name z is fixed on ξ. By definition, there is a U-name y fixed on ξ. Let $\mathsf{E}\Psi$ be the first sequent on ξ. Then, $(\mathsf{E}\Psi, y, z) \in S$, say $(\mathsf{E}\Psi, y, z) = (\mathsf{E}\Psi_k, y_k, z_k)$. So ξ restricted to $[\tilde{s}_n, \tilde{s}_{j_k}]_{\tilde{\pi}}$ is a sequent trace starting from $\mathsf{E}\Psi_k$ and on which both y_k and z_k are fixed, contradicting the choice of j_k. ◀

Combining Proposition 6.9 and Proposition 6.7 we have

▶ **Theorem 6.10.** *A formula φ is valid iff there is a $\mathsf{CTL}^*_\circ$ proof of φ.*

7 A decision procedure for the universal fragment of CTL*

The completeness proofs for the cyclic and ill-founded calculi provide a deterministic proof-search procedure which always yields a proof if the initial formula is valid. The argument leaves open the question of whether validity can be decided via proof-search, i.e., that no proof exists if none has been found within sufficiently many steps. In this section we provide a positive answer for the *universal* fragment of CTL^*, that is, for formulas containing no existential quantifier.

Given a universal formula φ, we build a finite proof-search tree for φ, annotate it according to the rules of the cyclic calculus, and show that if it does not contain a $\mathsf{CTL}^*_\circ$ proof of φ then φ is not valid.

▶ **Definition 7.1.** *An* annotated proof-search tree *for a universal formula φ is a finite tree T whose vertices are labelled according to the rules in Figure 2 and Figure 3 and such that*

1. *The root of T has label $\mathsf{A}\varphi$.*
2. *A vertex $u \in T$ is a leaf iff either*
   a. *u is axiomatic,*
   b. *u is labelled by literal sequents only, or*
   c. *there is a vertex $v <_T u$ such that u and v have identical labels.*
3. *In place of rule AX, the branching rule AX^*_b is used in T:*

$$\mathsf{AX}^*_b\ \frac{\Theta_1 : \mathsf{A}\Phi_1 \qquad \cdots \qquad \Theta_n : \mathsf{A}\Phi_n}{\Theta : \mathsf{AX}\Phi_1, \ldots, \mathsf{AX}\Phi_n, \Lambda}$$

   where Λ is a non-axiomatic set of literal sequents and Θ_i is the result of removing from Θ all names not occurring in $\mathsf{A}\Phi_i$.
4. *In every application of rule ALit in T, say with principal formula λ and principal sequent $\mathsf{A}\{\Phi, \lambda\}$, we have $\Phi \setminus \{\lambda\} \neq \varnothing$.*
5. *All instances of eW in T are instances of eThin.*
6. *The thinning rules iThin and eThin are prioritised over the rest in T.*

As a consequence of Proposition 5.3, we have:

▶ **Proposition 7.2.** *For every universal formula φ there exists an annotated proof-search tree for φ.*

In the absence of existential quantifiers, the success condition for repeats drastically simplifies.

▶ **Proposition 7.3.** *A repeat l in an annotated proof-search tree T is successful iff there is an R-name fixed on $[c_l, l]_T$.*

Proof. Since unfoldings of U-formulas under a universal quantifier are never annotated, there are no U-names in T, whence l is successful iff there is a sequent trace on $[c_l, l]_T$ where an R-name is fixed. By Lemma 6.3, this is the case iff there is an R-name fixed on $[c_l, l]_T$. ◀

Finally, we show that if an annotated proof-search tree for φ does not contain a $\mathsf{CTL}^*_\circ$ proof of φ, then φ is not valid.

▶ **Proposition 7.4.** *Let T be an annotated proof-search tree for a universal formula φ. If T cannot be pruned at AX^*_b-vertices down to a $\mathsf{CTL}^*_\circ$ proof of φ, then φ is not valid.*

Proof. Starting bottom-up, let T' be the result of pruning T at A∧-, AU- and AR-vertices by keeping the subtree that fails to produce a proof. Note that branching in T' is due solely to AX^*_b, that there are no axiomatic leaves in T', and that every repeat in T' is unsuccessful. So, by Proposition 7.3, for every repeat $l \in T'$ the following holds: there is no R-name fixed on $[c_l, l]_{T'}$. It follows that there cannot be an infinite sequent trace of type A containing an infinite formula trace of type R on $(T')^\circ$, because unfoldings of R-formulas under a universal quantifier are always annotated. Therefore, stripping $(T')^\circ$ of the annotations yields a refutation of φ, whence φ is not valid by Proposition 4.7. ◀

Proposition 7.4 yields a decision procedure for validity of universal formulas: given a universal formula φ, build an annotated proof-search tree for φ and check whether it contains a $\mathsf{CTL}^*_\circ$ proof of φ. This procedure fails in the presence of existential quantifiers. Using the notation from the proof of Proposition 7.4, T' may contain an unsuccessful repeat l with a good sequent trace of type A on $[c_l, l]_{T'}$. Thus, we cannot guarantee that $(T')^\circ$, once stripped of the annotations, is a refutation.

8 Conclusion

We introduce a sound and complete cyclic hypersequent calculus for Full Computation Tree Logic CTL^* and a decision procedure for validity of the universal fragment. Hypersequents – sets of sets of formulas – offer a natural framework for accommodating the existential and universal path quantifiers of the logic. Each "sequent" in a hypersequent is a labelled set of formulas, either $\mathsf{A}\Phi$ or $\mathsf{E}\Phi$, interpreted as *along all paths* $\bigvee \Phi$ and *along some path* $\bigwedge \Phi$ respectively. Through this interpretation, a natural system of ill-founded proofs arises wherein every infinite path of a proof must contain either an infinite sequent trace of type A through which *some* infinite formula trace stabilises (on a release operator), or an infinite trace of type E in which *all* infinite formula traces stabilise. Correctness conditions of the latter kind are rare in ill-founded (and cyclic) proof calculi. Indeed, together with [9], which employs a similar trace condition, these appear to be the only examples of cyclic systems that fall outside the scope of the category theoretic notion of cyclic proof introduced in [5].

In contrast to the ill-founded calculus, correctness of cyclic proofs is determined by the simple cycles only, i.e., the shortest path between leaf and companion. Soundness is ensured by annotating formulas in the cyclic calculus in a manner similar to [2, 3, 14, 17, 30]. As a result, proof-checking a cyclic derivation is linear time, in contrast to the trace condition along all paths which is PSPACE complete in general [21, 5].[3] Even so, the annotation mechanism necessary for CTL^* is significantly simpler than those developed for the μ-calculus and related logics. For example, each formula is annotated by at most one name and vice versa.

[3] It should be noted, however, that the trace condition for regular ill-founded CTL^* proofs in Definition 3.7 does not directly fit within the known PSPACE completeness results.

Future directions include investigating whether an analytic calculus for CTL* can assist with proving completeness of Hilbert-style calculi. Also of interest is the development of robust proof systems for related logics such as hybrid [28], graded [19], memory-full [16], and multi-agent [6] extensions of CTL*.

References

1 Pietro Abate, Rajeev Goré, and Florian Widmann. One-pass tableaux for computation tree 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 32–46. Springer, 2007. doi:10.1007/978-3-540-75560-9_5.

2 Bahareh Afshari, Sebastian Enqvist, and Graham E Leigh. Cyclic proofs for the first-order μ-calculus. *Logic Journal of the IGPL*, 2022. doi:10.1093/jigpal/jzac053.

3 Bahareh Afshari and Graham E. Leigh. Circular proofs for the modal mu-calculus. *PAMM*, 16(1):893–894, 2016. doi:10.1002/pamm.201610435.

4 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 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005088.

5 Bahareh Afshari and Dominik Wehr. Abstract cyclic proofs. In Agata Ciabattoni, Elaine Pimentel, and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information, and Computation - 28th International Workshop, WoLLIC 2022, Iaşi, Romania, September 20-23, 2022, Proceedings*, volume 13468 of *Lecture Notes in Computer Science*, pages 309–325. Springer, 2022. doi:10.1007/978-3-031-15298-6_20.

6 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.

7 Kai Brünnler and Martin Lange. Cut-free sequent systems for temporal logic. *J. Log. Algebraic Methods Program.*, 76(2):216–225, 2008. doi:10.1016/j.jlap.2008.02.004.

8 Mads Dam. Translating CTL* into the modal μ-calculus. *Tech. rep. ECS-LFCS-90-123, University of Edinburgh, Department of Computer Science, Laboratory for Foundations of Computer Science*, 1990.

9 Anupam Das and Marianna Girlando. Cyclic proofs, hypersequents, and transitive closure logic. In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings*, volume 13385 of *Lecture Notes in Computer Science*, pages 509–528. Springer, 2022. doi:10.1007/978-3-031-10769-6_30.

10 Stéphane Demri, Valentin Goranko, and Martin Lange. *Temporal Logics in Computer Science: Finite-State Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016. doi:10.1017/CBO9781139236119.

11 Sebastian Enqvist. A circular proof system for the hybrid μ-calculus. In Nicola Olivetti, Rineke Verbrugge, Sara Negri, and Gabriel Sandu, editors, *13th Conference on Advances in Modal Logic, AiML 2020, Helsinki, Finland, August 24-28, 2020*, pages 169–188. College Publications, 2020.

12 Oliver Friedmann, Markus Latte, and Martin Lange. A decision procedure for CTL* based on tableaux and automata. In *International Joint Conference on Automated Reasoning*, pages 331–345. Springer, 2010.

13 Dov M. Gabbay and Amir Pnueli. A sound and complete deductive system for CTL* verification. *Logic Journal of the IGPL*, 16(6):499–536, 2008.

14 Natthapong Jungteerapanich. A tableau system for the modal μ-calculus. In Martin Giese and Arild Waaler, editors, *Automated Reasoning with Analytic Tableaux and Related Methods, 18th International Conference, TABLEAUX 2009, Oslo, Norway, July 6-10, 2009. Proceedings*, volume 5607 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2009.

15 Ioannis Kokkinis and Thomas Studer. Cyclic proofs for linear temporal logic. *Ontos Mathematical Logic*, 6:171–192, 2016.

16 Orna Kupferman and Moshe Y. Vardi. Memoryful branching-time logic. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 265–274. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.34.

17 Johannes Marti and Yde Venema. A focus system for the alternation-free μ-calculus. In Anupam Das and Sara Negri, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 30th International Conference, TABLEAUX 2021, Birmingham, UK, September 6-9, 2021, Proceedings*, volume 12842 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2021. doi:10.1007/978-3-030-86059-2_22.

18 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

19 Faron Moller and Alexander Moshe Rabinovich. Counting on CTL*: On the expressive power of monadic path logic. *Inf. Comput.*, 184(1):147–159, 2003. doi:10.1016/S0890-5401(03)00104-4.

20 Damian Niwiński and Igor Walukiewicz. Games for the μ-calculus. *Theoretical Computer Science*, 163(1-2):99–116, 1996.

21 Rémi Nollet, Alexis Saurin, and Christine Tasson. PSPACE-Completeness of a thread criterion for circular proofs in linear logic with least and greatest fixed points. In S. Cerrito and A. Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, Lecture Notes in Computer Science, pages 317–334. Springer International Publishing, Cham, 2019. doi:10.1007/978-3-030-29026-9_18.

22 Mark Reynolds. An axiomatization of full computation tree logic. *The Journal of Symbolic Logic*, 66(3):1011–1057, 2001.

23 Mark Reynolds. An axiomatization of PCTL*. *Inf. Comput.*, 201(1):72–119, 2005. doi:10.1016/j.ic.2005.03.005.

24 Mark Reynolds. A tableau for CTL*. In Ana Cavalcanti and Dennis R. Dams, editors, *Formal Methods*, pages 403–418. Springer Berlin Heidelberg, 2009.

25 Mark Reynolds. A tableau-based decision procedure for CTL*. *Formal aspects of computing*, 23(6):739–779, 2011.

26 Jan Rooduijn. Cyclic hypersequent calculi for some modal logics with the master modality. In Anupam Das and Sara Negri, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 30th International Conference, TABLEAUX 2021, Birmingham, UK, September 6-9, 2021, Proceedings*, volume 12842 of *Lecture Notes in Computer Science*, pages 354–370. Springer, 2021. doi:10.1007/978-3-030-86059-2_21.

27 Jan Rooduijn and Lukas Zenger. An analytic proof system for common knowledge logic over S5. In *Proceedings of Advances in Modal Logic (AiML 2022)*, (to appear).

28 Ulrike Sattler and Moshe Y. Vardi. The hybrid μ-calculus. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2001.

29 Yury Savateev and Daniyar S. Shamkanov. Non-well-founded proofs for the Grzegorczyk modal logic. *Rev. Symb. Log.*, 14(1):22–50, 2021. doi:10.1017/S1755020319000510.

30 Colin Stirling. A tableau proof system with names for modal mu-calculus. In Andrei Voronkov and Margarita V. Korovina, editors, *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.

Functorial String Diagrams for Reverse-Mode Automatic Differentiation

Mario Alvarez-Picallo
Programming Languages Laboratory, Huawei Research Centre, Cambridge, UK

Dan Ghica
Department of Computer Science, University of Birmingham, UK
Programming Languages Laboratory, Huawei Research Centre, Cambridge, UK

David Sprunger
Department of Computer Science, University of Birmingham, UK

Fabio Zanasi
Department of Computer Science, University College London, UK

Abstract

We formulate a reverse-mode automatic differentiation (RAD) algorithm for (applied) simply typed lambda calculus in the style of Pearlmutter and Siskind [27], using the graphical formalism of string diagrams. Thanks to string diagram rewriting, we are able to formally prove for the first time the soundness of such an algorithm. Our approach requires developing a calculus of string diagrams with hierarchical features in the spirit of functorial boxes, in order to model closed monoidal (and cartesian closed) structure. To give an efficient yet principled implementation of the RAD algorithm, we use *foliations* of our hierarchical string diagrams.

2012 ACM Subject Classification Mathematics of computing → Automatic differentiation; Theory of computation → Categorical semantics

Keywords and phrases string diagrams, automatic differentiation, hierarchical hypergraphs

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.6

Funding The last three authors acknowledge support from EPSRC grant EP/V002376/1.

1 Introduction

Interest in incorporating differential structure to programming languages has grown with their use in machine learning, especially to support gradient-based training algorithms [4]. A central piece of these algorithms is reverse-mode automatic differentiation (RAD, also called backpropagation), which has proven to be an efficient way to compute gradients. For programs constructed entirely of differentiable operations on (tuples of) real numbers, automatic differentiation amounts to a mechanisation of the chain rule from (finite dimensional) multivariable calculus. However, in many programming languages, programs can themselves consume and produce other programs. Since spaces of functions on non-trivial real vector spaces are infinite dimensional, the differentiation of these higher-order programs cannot rely on the same justifications for correctness. Moreover, a notion of differentiation for higher-order programs can be useful even in situations where it is not necessary. For example, mapping a function over a list can always be expressed by an equivalent first-order program, but rewriting a program to fully remove higher-order functions is often not as convenient or efficient as a higher-order approach would be.

In this paper, we propose a differentiation algorithm for higher-order code which incorporates the fine-grained structure induced by defining and reusing programs within programs. In a nutshell, our algorithm extends every closure in the code with a back-propagator which computes the gradient of the original function along a given vector. It

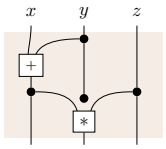
accomplishes this by executing a reversed version of the original computation graph. In order to differentiate higher-order functions, we also collect the partial derivatives of a function's output with respect to any captured variables/terms in its environment and pass those as the tangent value corresponding to an input of function type – in a similar style to Pearlmutter and Siskind's automatic differentiation algorithm [27].

```
1.  x += y;
2.  y := x * z;
```

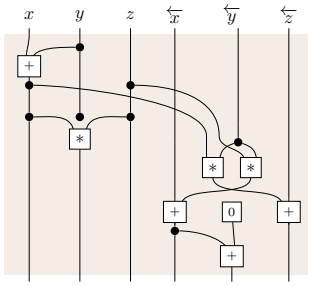
(a) Original.

```
1.  x += y;
2.  y := x * z;
3.  x̄ += z * ȳ;
4.  z̄ += x * ȳ;
5.  ȳ := 0;
6.  ȳ += x̄;
```

(b) Transformed.



(c) String diagram.



(d) Transformed diagram.

Figure 1 Approaches to RAD: program transformation and graphical.

In a typical program transformation approach to reverse-mode AD, each line of a program is paired with a small program computing that line's derivative. These derivatives are then assembled in the reverse order of the corresponding lines to obtain a transformed program which computes the original program (the "primal") together with its derivative. An example two-line primal program and its transformed version are shown in Figure 1a and Figure 1b. Note the first two lines of the original program are repeated (the "forward pass"). Lines 3–5 compute the derivative of line 2 and line 6 computes the the derivative of line 1, together forming the "reverse pass".

A good explanation of such a transformation should be detailed enough to serve as a complete specification for implementation and verification while also conveying the high-level ideas to facilitate necessary extensions. For example, the transformation of [27] assumes the program has been preprocessed to straight-line single-assignment code with unary and binary operations only. These assumptions rule out in-place assignments like line 1, so without high-level guidance one may not discover that destructive assignments like line 2 must have a zeroing line (line 5) in their reverse pass while nondestructive assigments should not.

The main novelty of our approach is the use of *category theory* to provide a firm semantic foundation to the algorithm. Our categorical environment features both the axioms of *cartesian closed categories*, modelling the operations of a generic functional language, and of *reverse differential categories* [12], accounting for first-order differentiability. We extract a detailed RAD transformation from the reverse differential operator; the axioms governing that operator provide a flexible high-level description one could use, for example, to understand why destructive and nondestructive assignment must be treated differently.

Roughly speaking, programs in these functional languages correspond with morphisms in these categories. To facilitate automated manipulation of these morphisms, we represent them as *string diagrams*, which are a graphical, yet completely formal language to represent morphisms [29]. For example, the program of Figure 1a corresponds with the string diagram of Figure 1c and its RAD transformed version (Figure 1b) corresponds to the string diagram of Figure 1d. To represent higher-order functions, we rely on a more sophisticated string diagram, namely **hierarchical string diagrams**, which in turn use *functorial boxes* [25].

A second ingredient that we need to introduce is the notion of **foliation**. A foliation is a "maximally spread out" representation of a string diagram; **leaves** in these foliations roughly correspond to lines in a program. This enables us to reason by *induction* on diagrams.

We formulate our algorithm as a rewriting system which, when applied to a foliation of a string diagram, produces its reverse-mode derivative. Thanks to the underlying reverse differential structure, using equational reasoning on string diagrams we are able to formulate and prove correctness of the algorithm (Theorem 17).

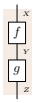
At a more fundamental level, these developments show how the dual nature of string diagrams – which may be interpreted both as (hyper)graphs and as a formal syntax – offers new perspective in the study of AD. On the one hand, string diagrams are akin to lower-level structures standardly found in AD implementations, such as computation graphs [1], and can be manipulated combinatorially. At the same time, their syntactic specification and algebraic properties enable methodologies typical of more abstract formalism, such as the approaches based on functional programming [13, 27]. Particularly, the use of induction, rewriting, and reverse derivative categories provide a basis for arguing soundness we do not see in computation graphs alone. In this sense, our approach ideally acts as a bridge between different formalisms to reason on AD algorithms.

Synopsis. In Section 2 we build towards the definition of hierarchical string diagrams. In Section 3 we develop the tools necessary to compute with these diagrams: foliations, hierarchical hypergraphs, and reverse differentiation. In Section 4 we present our algorithm for automatic differentiation and prove its correctness. Finally in Sections 5 and 6, we discuss related work and future work.

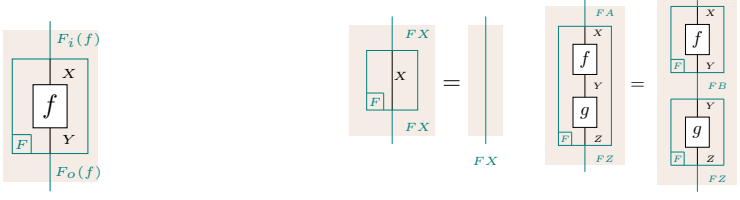
2 Hierarchical string diagrams

In this section we give a diagrammatic introduction to the basics of monoidal closed categories. String diagrams in this work are to be read from top to bottom.

2.1 Functorial string diagrams

Given a category, we denote the identity on an object X by , a morphism $f\colon X \to Y$ by  , and write  for the composition $f;g$ of morphisms $f\colon X \to Y$ and $g\colon Y \to Z$. Note the string diagrammatic notation neatly absorbs associativity and identity of composition.

Following [25], we extend string diagrams with *labelled frames* which indicate mappings between categories. The application of a mapping F to a morphism f can be seen in Figure 2a. Often such a mapping is a *functor*, respecting composition and identity as depicted in Figure 2b. We write $1_{\mathcal{C}}$ for the identity functor on category $\mathcal{C}$.

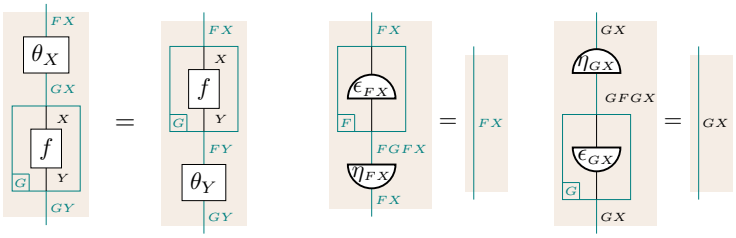


(a) Labelled frame. **(b)** Functor axioms.

Figure 2 Labelled frames and functors.

A natural transformation from a functor F to a parallel functor G is an object-indexed family of morphisms $\theta_A : FX \to GX$ satisfying the property depicted in this diagrammatic language in Figure 3a.

We may also express an *adjunction* between functors $F : \mathcal{D} \to \mathcal{C}$ (left adjoint) and $G : \mathcal{C} \to \mathcal{D}$ (right adjoint), via natural transformations $\epsilon : FG \to 1_{\mathcal{C}}$ and $\eta : 1_{\mathcal{D}} \to GF$, which satisfy unit and counit axioms found in Figure 3b. We will write the counit of an adjunction as a lower semicircle  , the unit as an upper semicircle  , and omit the label when the map is clear from context.

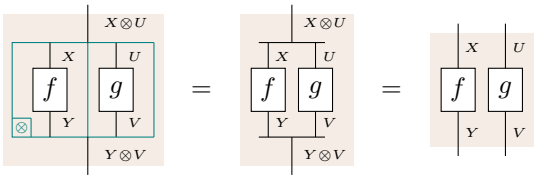


(a) Naturality. **(b)** Unit and counit axioms.

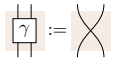
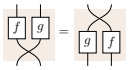
Figure 3 String diagrams for natural transformations and adjunctions.

2.2 String diagrams for monoidal categories

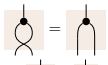
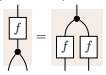
The diagrammatic notation can be generalised to bifunctors by drawing a two-place frame. One bifunctor that plays a special role in string diagrams is the *tensor product* or *monoidal product*, in particular when it is *strict*. The strict tensor is represented diagrammatically as:



The diagram above contains three representations. In the first one we see tensor as a bifunctor, with the two separate boxes indicating the two arguments of the bifunctor. The second one is special notation for the tensor, essentially hiding the functorial box and using a graphical convention (the horizontal line) to represent the "unravelling" of the tensored-labelled stem into components. Finally, the third is special notation for strict monoidal tensor, in which the tensor $X \otimes U$ is represented as the list of its components $[X, U]$.

The category is *symmetric* monoidal when there is a "symmetry" natural transformation $\gamma_{X,Y} : X \otimes Y \to Y \otimes X$ which is involutive, i.e. $\gamma_{X,Y}; \gamma_{Y,X} = \mathrm{id}_X \otimes \mathrm{id}_Y$. Diagrammatically, we represent symmetry by  := . As a consequence of the naturality of γ, we can "push" morphisms through symmetry:  .

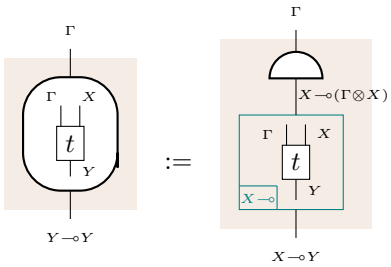
A *cartesian (monoidal) category* is a monoidal category where the tensor operation and unit are given by categorical products and the terminal object. One standard recipe to enforce this condition (see e.g. [19, Theorem 6.13]) on a monoidal category is to add monoidal natural transformations $\delta_X : X \to X \otimes X$ (contraction) and $\omega_X : X \to I$ (weakening) making I terminal, making every object into a cocommutative comonoid, and

interacting nicely with the product structure, i.e. $\delta_{X\otimes Y} = (\delta_X \otimes \delta_Y); (\mathrm{id}_X \otimes \gamma_{X,Y} \otimes \mathrm{id}_Y)$. We represent contraction as  and weakening as . The cocommutative comonoid equations include  and  . Copying and discarding are both consequences of naturality, i.e.  and  .

2.3 Hierarchical string diagrams

Monoidal closed categories and cartesian closed categories are fundamental in the construction of categorical models for the linear and simply-typed λ-calculus, respectively [6]. A *symmetric monoidal closed category* is a symmetric monoidal category where for every object Y, the (endo)functor $F_Y(X) = X \otimes Y$ has a right adjoint $G_Y(X) = X \multimap Y$. The equations defining this adjunction are presented diagrammatically in Appendix A.

We will not be using these here, but rather a hom-set presentation of the adjunction, consisting of a natural bijection between $\mathcal{C}(\Gamma \otimes X, Y)$ and $\mathcal{C}(\Gamma, X \multimap Y)$. This bijection is known as "currying", and is a more germane presentation for the λ-calculus. We define *abstraction*, the composition of the unit of the adjunction with the functorial box for G, as syntactic sugar denoted by a plain box with rounded corners.



This structure for abstraction yields our notion of a **hierarchical string diagram**, which is to say a string diagram which may contain other string diagrams in these boxes.

A cartesian monoidal category which is also a monoidal closed category is called a *cartesian closed category*. In the sequel we will restrict ourselves to cartesian closed categories, and so we will write $\times$ for $\otimes$ and $\Rightarrow$ for $\multimap$.

For computational purposes, it is important to have a data structure efficiently interpreting string diagrams. For string diagrams in symmetric monoidal categories, such a data structure is provided by hypergraphs with interfaces [8]. This can be extended to our closed context with *hierarchical hypergraphs* [2]. We also point out that foliations for hierarchical string diagrams (see Section 3.2) can be found efficiently by interpreting the string diagram as a hypergraph, then topologically sorting the edges.

3 Foliations and reverse differentiation

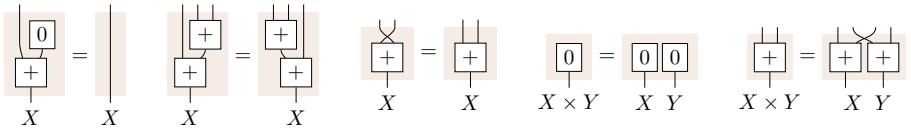
Next we focus on cartesian closed categories freely generated from a signature. In order to define differential structure, these signatures must include the vector space-like structure of (cartesian closed) left additive categories [7]. For our developments, we are especially interested in using *foliations*, special presentations of morphisms in these categories which enable the use of inductive definitions and proofs.

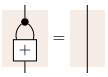
3.1 Freely generated categories

▶ **Definition 1.** *A* signature $(\Sigma_0, \Sigma_1, i, o)$ *consists of a set* Σ_0 *of* types*, a set* Σ_1 *of* operations*, and functions* $i, o : \Sigma_1 \to \Sigma_0^*$ *giving* input and output types *for each operation.*

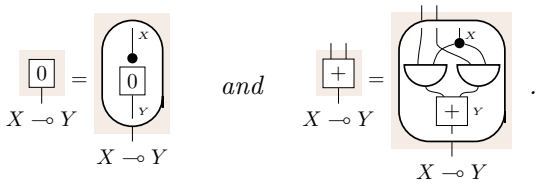
Derivatives are usually defined to be the linear map best approximating a function near a point. For us, this notion is provided by left additive categories, which introduce a monoid structure on homsets generalizing vector space addition.

▶ **Definition 2** ([12])**.** *A cartesian category* $\mathcal{C}$ *is* cartesian left additive *if each object* X *in* $\mathcal{C}$ *is a commutative monoid, meaning there are morphisms* $0_X : 1 \to X$ *and* $+_X : X \times X \to X$ *satisfying the first three equations depicted below. Furthermore, these monoids are required to be compatible with the cartesian structure in the sense of the last two equations.*



It is equivalent to say that every hom-set $\mathcal{C}(X, Y)$ is a commutative monoid, compatible with the cartesian structure; a full account of this notion can be found in [12]. Note that we do not assume any interaction between the additive structure and the comonoid structure given by counit and comultiplication;e.g. equations such as  do not hold in most models.

▶ **Definition 3** ([7, 1.4])**.** *A* cartesian closed left additive category *is both a cartesian left additive category and a cartesian closed category which respects the additive structure:*



Now we can give the setting for our automatic differentiation algorithm.

▶ **Definition 4.** *Suppose* Σ *is a signature. We let* $\mathcal{A}_\Sigma$ *be the free (strict) cartesian left additive category generated by* Σ*, and we let* $\mathcal{C}_\Sigma$ *be the free (strict) cartesian closed left additive category generated by* Σ*. Naturally,* $\mathcal{A}_\Sigma$ *is included in* $\mathcal{C}_\Sigma$*.*

The construction of these free categories is standard.

3.2 Foliations

Freely generated categories provide inductive structures with which we can describe all objects and morphisms. Being able to factorise morphisms gives us access to *foliations*, which allow us to treat morphisms as almost syntactic objects that we can perform induction on. Foliation-like objects, such as the more general polygraphs, are well-known in the literature, see for example [11, 18, 34]. Of particualr relevance is the use of foliations in [35] to enable the programmatic manipulation of string diagrams. We fix here the relevant definitions as they relate to hierarcical string diagrams.

▶ **Definition 5.** *Fix a set of string diagrams Σ_E, which we call* atomic.[1] *A* foliation *(with respect to Σ_E) is a sequential composition of string diagrams, which we call* leaves. *A* leaf *consists of an atomic morphism or an abstraction, tensored with any number of identities on either side. A* maximal foliation *is a foliation comprising only leaves. A* maximal hierarchical foliation *is a maximal foliation which is either abstraction free, or in which all abstracted diagrams are also maximal hierarchical foliations.*

For instance, the maximal foliations of  are  and  , if f and g are atomic. The following is an obvious generalisation of a folklore theorem.

▶ **Lemma 6.** *Any hierarchical string diagram can be written as a (non-unique) maximal hierarchical foliation.*

The proof is straightforward: intuitively, whenever two terms are "level" in a diagram one of them can be "shifted" using identities, then tensors and compositions can be reorganised using the functoriality of the tensor.

Foliations are inductive, allowing syntactic transformations defined recursively. Then instead of defining "big" rules for general sequential and parallel composites, we only need "small" rules for composing a morphism with a leaf, which is the format presenting the transformation in Figure 8. This also makes for simpler inductive proofs about the folations.

3.3 Reverse differential categories

Finally, we present a graphical treatment of basic reverse-mode differentiation without abstraction or function evaluation. The *reverse-mode derivative* of a function $f : \mathcal{R}^m \to \mathcal{R}^n$ is the function $(Jf)^\top : \mathcal{R}^m \to (\mathcal{R}^n \to \mathcal{R}^m)$ taking a base point in the domain of the function to the (linear map represented by the) transpose of the Jacobian of the function at that point. We denote the reverse mode derivative of a function by $\mathsf{R}f$.

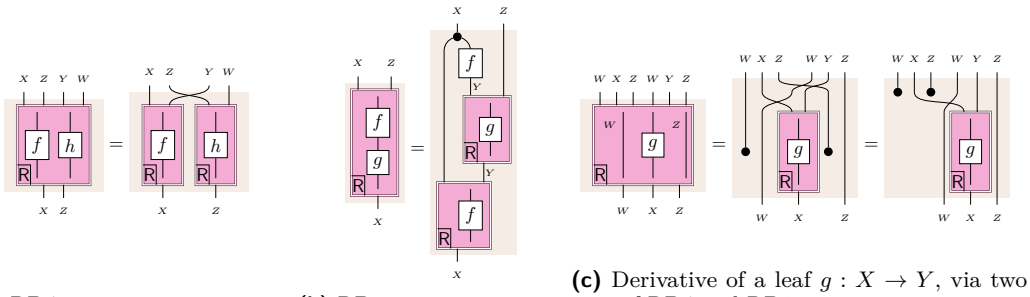
▶ **Example 7.** Consider binary multiplication, $m : \mathcal{R}^2 \to \mathcal{R}$ given by $m(x, y) = x \cdot y$. The Jacobian of this function at (x_0, y_0) is $\begin{bmatrix} y_0 & x_0 \end{bmatrix}$. Recall this means that given a small change in the inputs $\begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$ to m, the output will change by approximately $\begin{bmatrix} y_0 & x_0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \Delta x \cdot y_0 + x_0 \cdot \Delta y$. Thus, the reverse-mode derivative of m is the function $\mathsf{R}m : \mathcal{R}^2 \to (\mathcal{R} \to \mathcal{R}^2)$ given by $(\mathsf{R}m)(x_0, y_0) = [\Delta z \mapsto (y_0 \cdot \Delta z, x_0 \cdot \Delta z)]$.

The idea of this "reverse-mode derivative" is captured in the formalism of reverse differential categories [12].

▶ **Definition 8** ([12, Def. 13])**.** *A* reverse differential category*, or* reverse derivative category*, (RDC) is a cartesian left additive category endowed with a combinator* R *sending each morphism $f : X \to Y$ to a morphism $\mathsf{R}[f] : X \times Y \to X$. This combinator satisfies seven axioms (RD1-7) which can be found in both equational and diagrammatic form in Appendix B.*

We will introduce the axioms of the reverse derivative combinator necessary for understanding our algorithm gradually. We will use the a pink frame  to denote the reverse derivative $\mathsf{R}f$ of a morphism f in diagrammatic form. Note we are using the morphism frame notation for a *non-functorial* operation on morphisms. Despite this, this operator does have interesting compositional properties, as shown by axioms RD4 and RD5.

[1] Typically, we will take Σ_E to be the morphisms of a signature, together with some set of structural morphisms in a category.



(a) RD4. **(b)** RD5. **(c)** Derivative of a leaf $g : X \to Y$, via two uses of RD4 and RD2.

Figure 4 RDC axioms as string diagrams.

Next we consider an important class of morphisms: the linear morphisms.

▶ **Definition 9.** *A morphism $f : X \to Y$ in a reverse differential category is* linear *whenever there is a morphism $g : Y \to X$ such that $\mathsf{R}[f] = \omega_X \times g$ and $\mathsf{R}[g] = \omega_Y \times f$.*

Note that if f is linear, then so is g. We call g the dual *of f and denote it by $f^\dagger$.*

The subcategory of linear maps in an RDC is a dagger category [12, Prop. 24].

▶ **Lemma 10.** *When f and g are linear maps in a reverse differential category, so are $f;g$ and $f \times g$ and their duals are $(f;g)^\dagger = g^\dagger; f^\dagger$ and $(f \times g)^\dagger = f^\dagger \times g^\dagger$.*

The axioms of reverse differential categories require the structural morphisms (id_X, $\gamma_{X,Y}, \delta_X, \omega_X, 0_X$, and $+_X$) to be linear, and further require that 0_X is dual to ω_X, $+_X$ is dual to δ_X, $\gamma_{X,Y}$ is dual to $\gamma_{Y,X}$, and id_X is dual to itself.

In order to equip categories generated from a signature with reverse differential structure, it is sufficient to designate a suitable derivative for each of the basic operations by the following lemma.

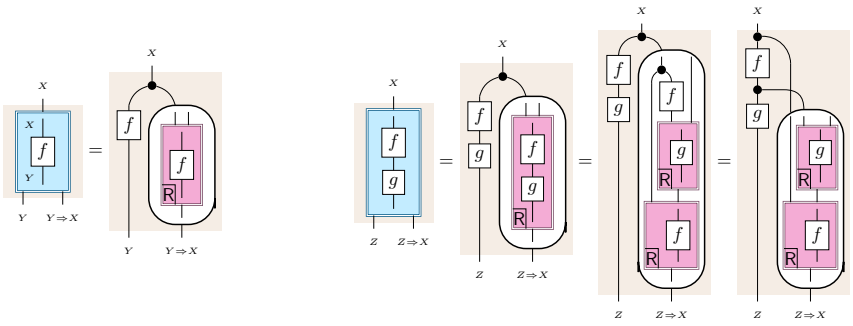
▶ **Lemma 11.** *Suppose Σ is a signature and suppose that for every operation $g : X \to Y$ in Σ_1, there is a well-defined reverse derivative $\mathsf{R}[g]$ morphism in $\mathcal{A}_\Sigma$ satisfying RD2, RD6, and RD7. Then $\mathcal{A}_\Sigma$ is a reverse differential category.*

In such a case, we will call Σ a differentiable signature.

It is an exercise for the reader to check that with the reverse-mode derivative of multiplication defined in Example 7, the string diagram of Figure 1d is the tupling of the string diagram of Figure 1c with its reverse-mode derivative. In this exercise, one will find that the zeroing of $\overleftarrow{y}$ is the dual of the discard of the destructive assignment. Thus, that this requires a zeroing of the corresponding reverse-pass variable (while in-place operations do not) is a consequence of the RDC axioms.

3.4 Optimizing RAD string diagrams

Reverse differential categories are defined without closed structure to maximize generality. However, in a closed setting, we can abstract the differential component of input to the reverse derivative. With this, we can more precisely state the goal of this work: we seek an "adjoint transformation" taking $f : X \to Y$ to $Af : X \to Y \times (Y \Rightarrow X)$, which augments the result of f with a function representing its reverse-mode derivative at the given input. We depict this transformation in Figure 5a.



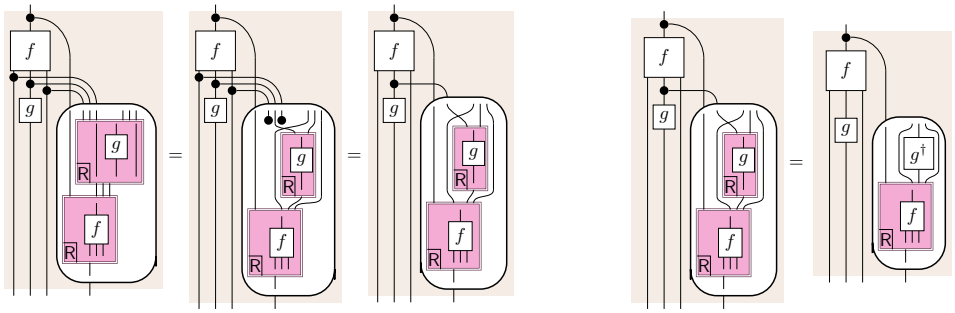
(a) Adjoint transformation. **(b)** Adjoint transformation applied to sequential composition.

Figure 5 Basics of the adjoint transformation.

The RDC axioms, particularly RD5, tell us how to compute this adjoint transformation when f can be decomposed as a sequential composite, as shown in Figure 5b. This immediately suggests important, common optimizations in practical implementations.

In proceeding from the third diagram to the fourth, some computation is saved (f is used only once on the right) at the cost of having to transmit an extra intermediate result to the closure giving the derivatve. This is a common time-space tradeoff in backpropagation where usually space is sacrificed.

Next, if the second layer of computation is a leaf in a foliation, we can make this diagram even more precise. Substituting the reverse derivative of a leaf from Figure 4c into the diagram in Figure 5b, we obtain Figure 6a. This expresses space savings common in all practical implementations: the whole state of the computation does not need to be saved at each step, only the values which are used in the next operation.



(a) Adjoint transformation on a leaf.

(b) Adjoint transformation on a *linear* leaf.

Figure 6 Adjoint transformation on (linear) leaves.

Finally, if the operation in a leaf is linear, we can simplify to Figure 6b. This saves even more space – no inputs to this step need be saved since the derivatives of linear operations do not depend on the base point.

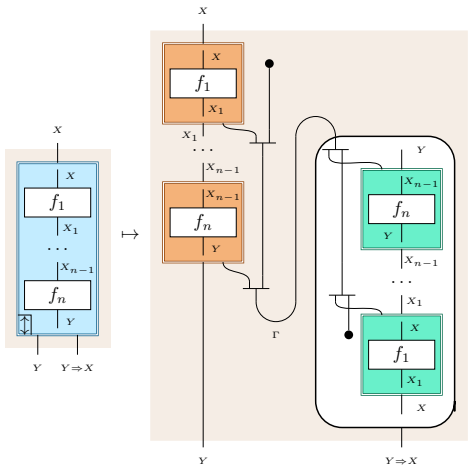
4 A diagrammatic AD algorithm

This section presents the main technical result of our paper: to define and prove the soundness of an algorithm for performing reverse-mode automatic differentiation on hierarchical string diagrams. Our algorithm can be considered a simplified version[2] of the one introduced in [27], which was remarkable for being one of the first such algorithms that can correctly differentiate code containing closures and higher-order functions.

4.1 Rewrite rules on string diagrams

Our algorithm consists of three transformations, the application of which we denote by differently coloured boxes around a diagram. We emphasise that these boxes represent *meta-level transformations*, and are not to be confused with object-level entities such as the rounded rectangles that we use to denote abstraction.

We have previously described the adjoint transformation we seek. Its only rewrite rule can be found in Fig. 7a, is denoted by a **blue** box, and is the entry point of the algorithm. Given a diagram written as a maximal foliation $f_n \circ \ldots \circ f_1 : X \to Y$, this transformation extends it with an extra closure $Y \Rightarrow X$ called a *backpropagator*; the final result being a diagram of type $X \to Y \times (Y \Rightarrow X)$. This backpropagator computes the reverse derivative of the original diagram along a given input vector. In particular, if the original diagram produces a single real number, evaluating its backpropagator at 1 computes the gradient of the original diagram.



(a) Our adjoint rule.

$$
\begin{aligned}
&\overleftarrow{\lambda x_0 \,\mathbf{let}\; x_1 \triangleq e_1; \equiv \lambda \overleftarrow{x_0}\,\mathbf{let}\; \phi\{x_1 \triangleq e_1\};}\\
&\qquad\vdots \qquad\qquad\qquad\qquad \vdots\\
&\quad x_n \triangleq e_n \qquad\qquad \phi\{x_n \triangleq e_n\}\\
&\mathbf{in}\; x_n\;\mathbf{end} \qquad \mathbf{in}\; (\overleftarrow{x_n}, (\lambda \overleftarrow{x_n}\;\mathbf{let}\; \overleftarrow{x'_1} \triangleq \mathbf{0}\; (\overleftarrow{\mathcal{J}}^{-1}\; \overleftarrow{x'_1});\\
&\qquad\qquad\qquad\qquad\qquad\qquad\vdots\\
&\qquad\qquad\qquad\qquad\qquad\qquad \overleftarrow{x'_l} \triangleq \mathbf{0}\; (\overleftarrow{\mathcal{J}}^{-1}\; \overleftarrow{x'_l});\\
&\qquad\qquad\qquad\qquad\qquad\qquad \overleftarrow{x_0} \triangleq \mathbf{0}\; (\overleftarrow{\mathcal{J}}^{-1}\; \overleftarrow{x_0});\\
&\qquad\qquad\qquad\qquad\qquad\qquad\vdots\\
&\qquad\qquad\qquad\qquad\qquad\qquad \overleftarrow{x_{n-1}} \triangleq \mathbf{0}\; (\overleftarrow{\mathcal{J}}^{-1}\; \overleftarrow{x_{n-1}});\\
&\qquad\qquad\qquad\qquad\qquad\qquad \rho\{x_n \triangleq e_n\};\\
&\qquad\qquad\qquad\qquad\qquad\qquad\vdots\\
&\qquad\qquad\qquad\qquad\qquad\qquad \rho\{x_1 \triangleq e_1\}\\
&\qquad\qquad\qquad\qquad\qquad \mathbf{in}\; (\lceil \overleftarrow{x'_1}, \ldots, \overleftarrow{x'_l} \rceil_{x_0}, \overleftarrow{x_0})\;\mathbf{end}))\;\mathbf{end}
\end{aligned}
$$

(b) Reverse transform in [27].

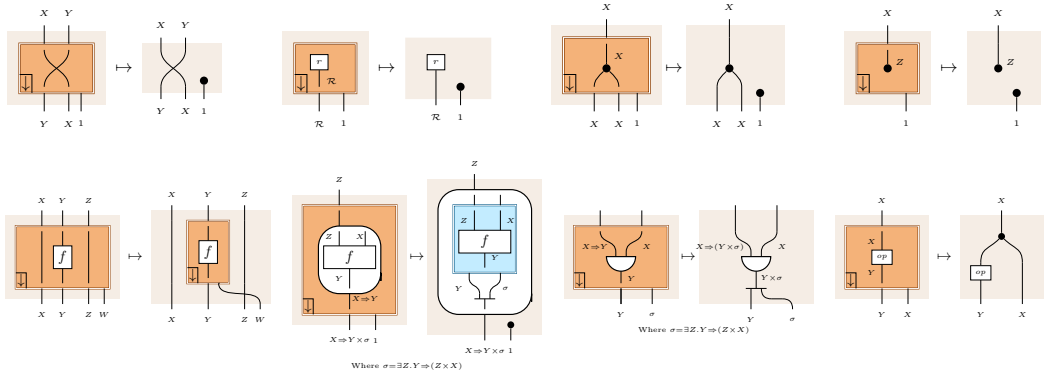
Figure 7 Comparison of RAD transformations.

This adjoint transformation is in turn defined by two further transformations: a *forward pass* (in **orange** labelled by ↓, rewrite rules in Fig. 8a) and a *reverse pass* (in **green** labelled by ↑, rewrite rules in Fig. 8b) which are applied to the leaves of the foliation. These rules correspond to the forward and reverse passes commonly employed in reverse-mode AD. The forward pass executes the original function "as is"; the reverse pass computes the gradient of every sub-expression in reverse order of execution. In our algorithm, as explained in

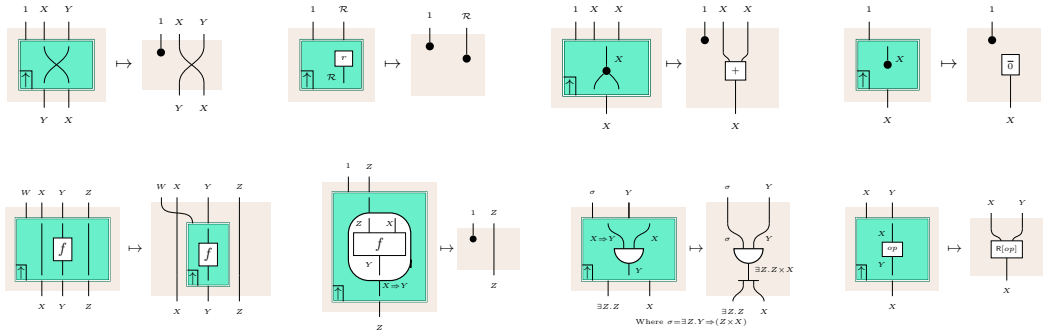
[2] Specifically, our algorithm does not handle lists, Boolean values or conditionals, although it can be extended to account for these features in a straightforward way.

Section 3.4, intermediate values computed during the forward pass are stored and passed along to the diagram corresponding to the reverse pass, shown in Fig. 7a as a bundle of type Γ flowing from the forward pass into the backpropagator, which we refer to as the *context* wire.

The adjoint transformation corresponds to the "reverse transform" of [27, p. 26]. The definition of this reverse transform is shown in Figure 7b. This definition references two other per-line transformations ϕ and ρ, which correspond to our forward (ϕ) and reverse (ρ) passes. The ϕ transformation generally keeps the same program shape, occasionally binding new variables; analogously our forward pass occasionally saves intermediate values for the backpropagator. Inside the binding, Pearlmutter and Siskind's reverse transformation zeroes many variables (since they assume the program has been preprocessed to single-assignment), then ρ-transforms the original code, analogous to our reverse pass.



(a) Rewrites defining the forward pass.



(b) Rewrites defining the reverse pass.

Figure 8 Forward and reverse pass rewrites.

The first four rules (top row) of the forward and reverse pass follow the logic introduced in Section 3.4: linear morphisms are simply replaced by their dual in the reverse pass – observe that, since the reverse derivative of a linear diagram does not depend on its base point, these rules only generate a "dummy" unit value to be added to the context wire. We have already seen the logic for the last rule on the bottom row as well: for a generic nonlinear basic operation, the forward pass needs to save the basepoint and the reverse pass uses that for its reverse-mode derivative.

▶ Remark 12. We note that each copy node in the primal code is replaced by an addition in the reverse pass, adding up all the sensitivities corresponding to each use of a variable before propagating them further up the computation graph. This prevents the *fanout problem* and ensures efficiency without requiring additional optimization passes like *linear factoring* used by [10] or sophisticated, quasi-symbolic representations of operations as in [22].

The rules for abstraction (left bottom row) and evaluation (middle bottom row) are harder to back up with compelling intuitions. For abstraction, the diagram enclosed by the bubble is recursively transformed using the blue rule in the forward pass – that is, any abstraction in the primal diagram is replaced by a new abstraction that computes the adjoint of the original one. Then, when this function is evaluated in the primal diagram, the forward pass gets the result of the adjoint application – both the result of the original abstraction and a backpropagator which is not used in the forward pass but is set aside for the reverse pass.

When rewriting an application node, the reverse pass applies the backpropagator produced by the forward pass. This backpropagator in turn produces a wire for every operand of the body of the original abstraction. The abstraction rule in the reverse pass then expects the sensitivity of an abstraction to consist of a bundle of wires corresponding to the sensitivities of each captured wire.

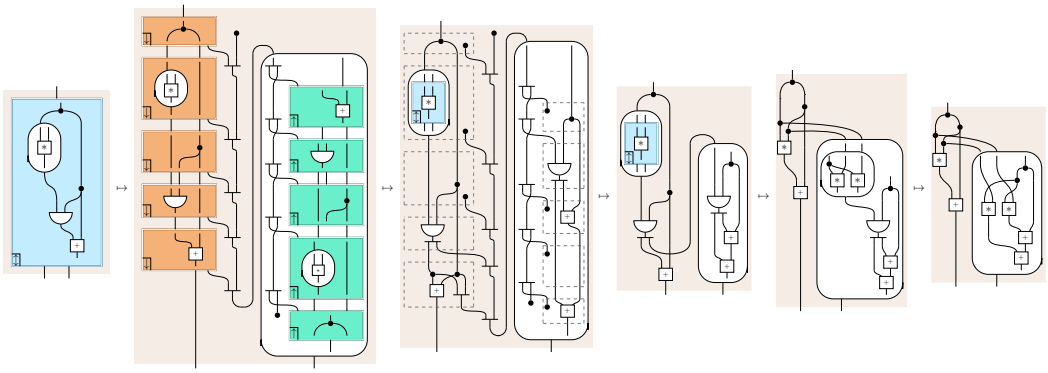


Figure 9 Differentiating a simple polynomial.

As an example, consider the program `let mul z = y * z in mul y + y`, which computes the polynomial $y \mapsto y^2 + y$. In Fig. 9, we show the result of applying the adjoint transformation to this diagram. We assume that the signature that generates our category includes a multiplication[3] morphism  whose corresponding reverse derivative is given by  (which is simply the gradient of the product familiar from basic calculus). The resulting backpropagator, when applied to input 1, gives the correct derivative of the original polynomial.

▶ **Lemma 13.** *The adjoint transformation defined by Fig. 8a and Fig. 8b is independent of the choice of foliation, up to a permutation of the context wire.*

Proof (Sketch). By induction on the length of the foliation. It can be easily checked that every pair of such rules commutes, modulo a permutation of the wires that are propagated from the forward to the reverse pass. For a concrete example, consider the two rewrites stemming from two different foliations of the same graph in Fig. 10, and note that they differ only in the obvious permutation morphism $\gamma : X \times (Y \times \Gamma) \to Y \times (X \times \Gamma)$. ◀

[3] Multiplication here should be understood as multiplication of real numbers, and not be confused with the product structure of our category.

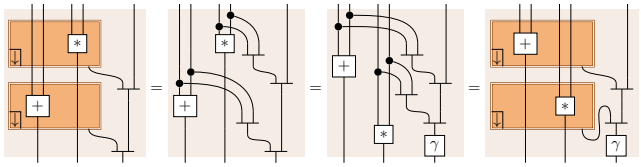


Figure 10 Two rewrite sequences stemming from two distinct foliations of the same graph.

▶ Remark 14. The proof above, although simple, illustrates a proof method that is made possible by using string diagrams: induction on the length of the *foliation* of the diagram (Def. 5). This proof method also benefits from absence of names and all related bureaucratic concerns (free vs. bound variables, alpha equivalence, capture-avoiding substitution).

4.2 Correctness

Our proof of correctness proceeds in two steps. First, we prove that our AD transformation is compatible with Beta reduction: whenever two diagrams are equivalent modulo Beta reduction, then so are their adjoints. Then, we show that the AD transformation is correct for diagrams featuring only first-order nodes (featuring no abstractions or applications).

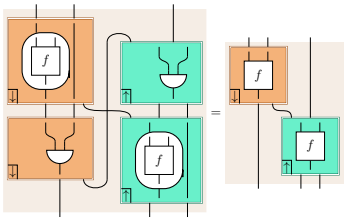


Figure 11 AD is compatible with Beta-reduction.

▶ **Lemma 15.** *The rewriting rules in Fig. 8 are compatible with beta reduction (The equation in Fig. 11 holds).*

Proof. The proof proceeds by straightforward application of the rewrite rules. We provide the calculation in full in Fig. 12. ◀

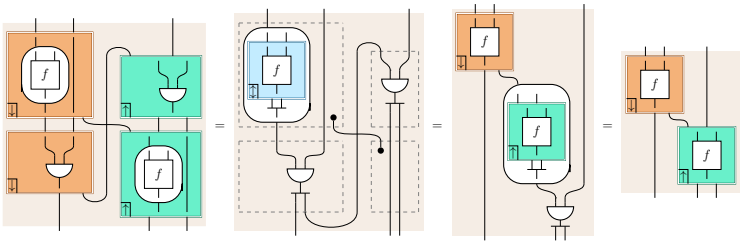
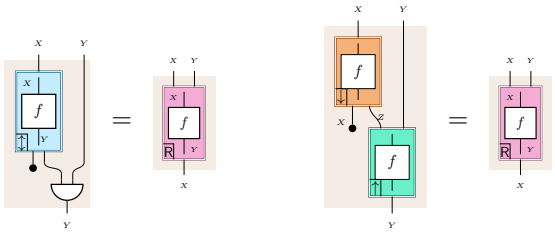


Figure 12 Beta-soundness of AD.

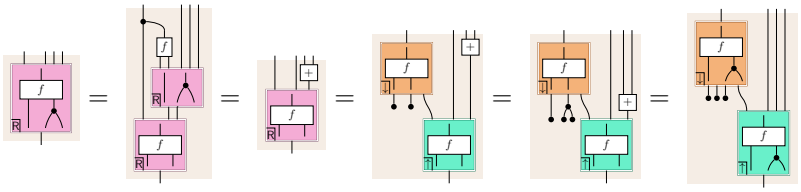
▶ **Lemma 16.** *For every*  *whose operands and results are all of a first-order type, there is a Beta-equivalent*  *that contains no instances of abstraction or evaluation and whose every node has all first-order inputs and outputs.*

Proof. Since our graphical language is simply-typed, evaluation of  is guaranteed to terminate, following an argument similar to the proofs of strong normalisation for the simply-typed λ-calculus (such as the one in [16, Chapter 6]), or for interaction nets (e.g. [24]). Such a normal form cannot contain any redexes, so any abstraction or eval node must be connected to an operand or a result wire, which cannot happen as these have all first-order types. ◀

▶ **Theorem 17.** *For every*  *whose operands and results are all first-order, the following (equivalent) equations hold:*



Proof. Applying Lemma 16 and Lemma 15, it suffices to consider the case where  contains no instances of application or evaluation. Applying the rewrite rule in Fig. 7a shows the equivalence of the two equations above. The result then follows by induction on the foliation of the diagram  . We show the case for contraction in full, with the other cases being similar.



◀

5 Related work

5.1 String diagrams

Cartesian closed categories have been thoroughly studied in the context of logic and type theory, because of the well-known correspondence of their internal language with λ-calculus and intuitionistic logic [32]. The *linear* version of this triad involves monoidal rather than cartesian categories, but also proof nets, and linear logic, as indicated already in the original paper [15]. [25] provides the foundation on which we build our language of string diagrams, noting that all the basic ingredients are already there.

The route of using an enhancement of the monoidal closed structure with additional properties to control sharing is fruitful and has been employed many times. For example it is found in [9], where the manipulation of variables endowed with algebraic theories is modelled as a cartesian structure on the top of a linear structure, or in [14] to specify multiplexers and demultiplexers in high-level synthesis.

Our approach to string diagrams shares similarities with the formalisms of sharing graphs for describing λ-calculus computations [23]. The main difference is that string diagrams can be manipulated as a syntax, whereas sharing graphs are usually studied as combinatorial objects. Unlike syntax, reasoning about graphs algebraically requires a higher degree of technical sophistication [17]. Finally, sharing graphs are typically used to study low-level computational models for functional languages, in particular quantitative resource models [26], whereas our approach is more focussed on equational reasoning and rewriting.

Monoidal closed categories extend not only to cartesian closed categories, but also to $\star$-autonomous categories. This second variation is very much relevant to the study of multiplicative linear logic and it has been extensively studied in terms of proof nets. Our graphical calculus is essentially different from proof nets. The grammar of generating morphism does not stem from a sequent calculus, and we capture the intended semantics via equations rather than a correctness criterion. But the connection might be made precise relying on the existing translations between proof nets and string diagrams [20, 30].

Baez and Stay [3] propose an intriguing graphical innovation for monoidal closed categories, a so-called *clasp* operator on stems. The exponential type is represented using the clasp, and much like in our own language, a *bubble* is used to represent currying. It is unclear to us how to represent higher-order objects like $(A \multimap B) \multimap C$ using the clasp. However, we think it preserves an appealing visual parallel between monoidal closed and compact closed structures.

Although not presented explicitly as a string diagram language, the treatment of closures in [28] is related in methodology to our work, although the use of *partially-traced partially-closed premonoidal categories* as the categorical setting, in order to accommodate for effects, is significantly different than our cartesian closed categorical language.

5.2 Automatic differentiation

Our AD algorithm is an adaptation of [27]. Beyond the presentation based on string diagrams, the main differences are that our algorithm applies to simply-typed, recursion-free code and it acts as a source-to-source transformation, lacking the reflection features that enabled higher-order differentiation in the original work. We chose to focus on this algorithm for a few reasons: first, reverse-mode AD is both more immediately useful (see [4] for a comparison of both approaches) and harder to implement and prove correct than forward-mode AD. Second, it is to our knowledge the first published algorithm for performing reverse-mode AD on higher-order code, it forms the basis of a number of efficient implementations [5, 31] and does not require more complex features, unlike e.g. [36] which makes use of mutable state and continuations or [10] which relies on a limited form of continuations to encode dual spaces.

A wave of recent research has also tackled the issues of correctness in automatic differentiation. Notably, [10] and [33] provide correct reverse-mode AD algorithms capable of handling closures. Unlike the first work, however, our algorithm is purely functional and, while the second one can correctly differentiate terms with higher-order inputs and outputs, it does so by using a more expensive representation of tangents of function spaces. The main contribution of our approach, however, is the simplicity of the involved proofs thanks to our diagrammatic notation which we believe improves on the readability of the original paper [27] and the denser proofs in newer literature [10, 21, 33].

6 Conclusion and further work

In this paper we have presented a recipe for provably correct reverse automatic differentiation built around a hierarchical calculus of string diagrams. As we have seen, the string diagram presentation simplifies much of the bookkeeping of variables a term calculus would require, which makes a complicated algorithm more readable. More importantly, the new perspective offered by string diagrams and in particular the presentation of terms as *foliations* opens the door for new and useful proof techniques.

We have not discussed implementation matters, yet these are of the essence. This is a practical algorithm and it can be incorporated into real-life compilers for real-life programming languages. We surmise that the improved perspective on AD that string diagrams offers will help handle other challenging features of real-life languages, such as effects and recursion. This work is ongoing.

References

1 Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: a system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

2 Mario Alvarez-Picallo, Dan Ghica, David Sprunger, and Fabio Zanasi. Rewriting for monoidal closed categories. In *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

3 John Baez and Mike Stay. Physics, topology, logic and computation: a rosetta stone. In *New structures for physics*, pages 95–172. Springer, 2010. doi:10.1007/978-3-642-12821-9_2.

4 Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, 18:153:1–153:43, 2017. URL: http://jmlr.org/papers/v18/17-468.html.

5 Atilim Günes Baydin, Barak A. Pearlmutter, and Jeffrey Mark Siskind. Diffsharp: An AD library for .net languages. *CoRR*, abs/1611.03423, 2016. arXiv:1611.03423.

6 Nick Benton, Gavin Bierman, Valeria De Paiva, and Martin Hyland. Linear λ-calculus and categorical models revisited. In *International Workshop on Computer Science Logic*, pages 61–84. Springer, 1992.

7 Richard F Blute, J Robin B Cockett, and Robert AG Seely. Cartesian differential categories. *Theory and Applications of Categories*, 22(23):622–672, 2009.

8 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 710–719. ACM, 2016. doi:10.1145/2933575.2935316.

9 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Deconstructing lawvere with distributive laws. *J. Log. Algebraic Methods Program.*, 95:128–146, 2018. doi:10.1016/j.jlamp.2017.12.002.

10 Aloïs Brunel, Damiano Mazza, and Michele Pagani. Backpropagation in the simply typed lambda-calculus with linear negation. *Proc. ACM Program. Lang.*, 4(POPL):64:1–64:27, 2020. doi:10.1145/3371132.

11 Albert Burroni. Higher-dimensional word problems with applications to equational logic. *Theoretical computer science*, 115(1):43–62, 1993.

12 Robin Cockett, Geoffrey Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, Benjamin MacAdam, Gordon Plotkin, and Dorette Pronk. Reverse derivative categories. *arXiv preprint*, 2019. arXiv:1910.07065.

13 Conal Elliott. The simple essence of automatic differentiation. *Proceedings of the ACM on Programming Languages*, 2(ICFP):1–29, 2018.

14 Dan R. Ghica. Geometry of synthesis: a structured approach to VLSI design. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 363–375. ACM, 2007. doi:10.1145/1190216.1190269.

15 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.

16 Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*, volume 7. Cambridge university press Cambridge, 1989.

17 Stefano Guerrini. A general theory of sharing graphs. *Theor. Comput. Sci.*, 227(1-2):99–151, 1999. doi:10.1016/S0304-3975(99)00050-X.

18 Yves Guiraud. *Rewriting methods in higher algebra*. PhD thesis, Université Paris 7, 2019.

19 Chris Heunen and Jamie Vicary. Lectures on categorical quantum mechanics. *Computer Science Department. Oxford University*, 2012. URL: http://www.cs.ox.ac.uk/files/4551/cqm-notes.pdf.

20 Dominic Hughes. Simple free star-autonomous categories and full coherence. *CoRR*, 2005. arXiv:0506521.

21 Mathieu Huot, Sam Staton, and Matthijs Vákár. Correctness of automatic differentiation via diffeologies and categorical gluing. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 319–338. Springer, 2020. doi:10.1007/978-3-030-45231-5_17.

22 Faustyna Krawiec, Simon Peyton Jones, Neel Krishnaswami, Tom Ellis, Richard A Eisenberg, and Andrew W Fitzgibbon. Provably correct, asymptotically efficient, higher-order reverse-mode automatic differentiation. *Proc. ACM Program. Lang.*, 6(POPL):1–30, 2022.

23 John Lamping. An algorithm for optimal lambda calculus reduction. In Frances E. Allen, editor, *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*, pages 16–30. ACM Press, 1990. doi:10.1145/96709.96711.

24 Ian Mackie. Interaction nets for linear logic. *Theor. Comput. Sci.*, 247(1-2):83–140, 2000. doi:10.1016/S0304-3975(00)00198-5.

25 Paul-André Melliès. Functorial boxes in string diagrams. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2006. doi:10.1007/11874683_1.

26 Koko Muroya and Dan R. Ghica. The dynamic geometry of interaction machine: A token-guided graph rewriter. *Log. Methods Comput. Sci.*, 15(4), 2019. doi:10.23638/LMCS-15(4:7)2019.

27 Barak A. Pearlmutter and Jeffrey Mark Siskind. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Trans. Program. Lang. Syst.*, 30(2):7:1–7:36, 2008. doi:10.1145/1330017.1330018.

28 Ralf Schweimeier and Alan Jeffrey. A categorical and graphical treatment of closure conversion. In Stephen D. Brookes, Achim Jung, Michael W. Mislove, and Andre Scedrov, editors, *Fifteenth Conference on Mathematical Foundations of Progamming Semantics, MFPS 1999, Tulane University, New Orleans, LA, USA, April 28 - May 1, 1999*, volume 20 of *Electronic Notes in Theoretical Computer Science*, pages 481–511. Elsevier, 1999. doi:10.1016/S1571-0661(04)80090-2.

29 Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010. arXiv:0908.3347.

30 Michael Shulman. ∗-autonomous envelopes and 2-conservativity of duals. *CoRR*, 2020. arXiv:2004.08487.

31 Jeffrey Mark Siskind and Barak A. Pearlmutter. Efficient implementation of a higher-order language with built-in AD. *CoRR*, abs/1611.03416, 2016. arXiv:1611.03416.

32 Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*. Elsevier, 2006.

33 Matthijs Vákár. Reverse AD at higher types: Pure, principled and denotationally correct. In Nobuko Yoshida, editor, *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12648 of *Lecture Notes in Computer Science*, pages 607–634. Springer, 2021. doi:10.1007/978-3-030-72019-3_22.

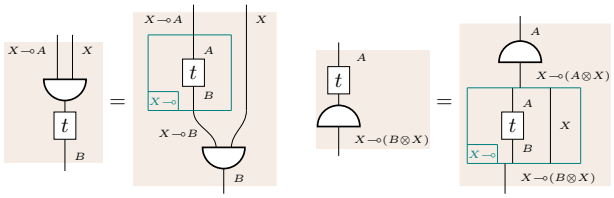
34 Jamie Vicary and Antonin Delpeuch. Normalization for planar string diagrams and a quadratic equivalence algorithm. *Logical Methods in Computer Science*, 18, 2022.

35 Jamie Vicary, Aleks Kissinger, and Krzysztof Bar. Globular: an online proof assistant for higher-dimensional rewriting. *Logical Methods in Computer Science*, 14, 2018.

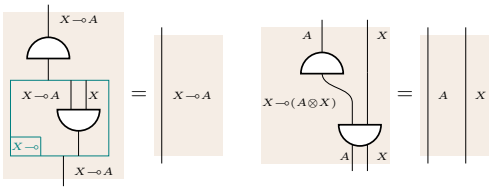
36 Fei Wang, Daniel Zheng, James M. Decker, Xilun Wu, Grégory M. Essertel, and Tiark Rompf. Demystifying differentiable programming: shift/reset the penultimate backpropagator. *Proc. ACM Program. Lang.*, 3(ICFP):96:1–96:31, 2019. doi:10.1145/3341700.

A Monoidal closed adjunctions, diagrammatically

Instantiated to the families of functors F_X, G_X of Section 2.3, the naturality and adjunction equations are expressed in string diagrams as in Fig. 13. The counit of the adjunction is normally called *eval*, and we call the unit *coeval* for the sake of symmetry in terminology and by analogy with compact-closed categories.



(a) Eval and coeval are natural transformations.



(b) Eval and coeval form an adjunction.

Figure 13 String diagram representation of MCC axioms.

B Reverse differential categories

We present here the full definition of reverse differential categories. For a more exhaustive treatment of these, containing a discussion of the intuition behind the axioms and many useful properties of reverse differential categories, we refer to [12].

▶ **Definition 18** ([12, Def. 13]). *A* reverse differential category *is a cartesian left additive category endowed with a combinator* R *sending each morphism* $f : X \to Y$ *to a morphism* $\mathsf{R}[f] : X \times Y \to X$ *which satisfies the following conditions:*

(RD1) $\mathsf{R}[0_X] = \omega_X$ *and* $\mathsf{R}[+_X] = \omega_{X\times X} \times \delta_X$,

(RD2) $(id_X \times 0_Y); \mathsf{R}[f] = \omega_X; 0_X$ *and* $(id_X \times +_Y); \mathsf{R}[f] = (\delta_X \times id_{Y\times Y}); (id_X \times \gamma_{X,Y} \times id_Y); (\mathsf{R}[f] \times \mathsf{R}[f]); +_X$,

(RD3) $\mathsf{R}[id_X] = \omega_X \times id_X$, $\mathsf{R}[\omega_X] = \omega_X \times 0_X$, *and* $\mathsf{R}[\delta_X] = \omega_X \times +_X$

(RD4) $\mathsf{R}[f \times h] = (id_X \times \gamma_{Z,Y} \times id_W); (\mathsf{R}[f] \times \mathsf{R}[h])$,

(RD5) $\mathsf{R}[f; g] = (\delta_X \times id_Z); (id_X \times f \times id_Z); (id_X \times \mathsf{R}[g]); \mathsf{R}[f]$

(RD6) $(id_{X\times Y} \times 0_{X\times X} \times id_Y); \mathsf{R}^3[f]; (\omega_{X\times Y} \times id_X) = (id_X \times \omega_Y \times id_Y); \mathsf{R}[f]$

(RD7) $(id_X \times 0_Y \times id_X \times 0_{X\times Y} \times id_X \times 0_Y \times id_X); \mathsf{R}^4[f]; (\omega_{X\times Y\times X\times X} \times id_Y) = (id_X \times \gamma_{X,X} \times id_X); (id_X \times 0_Y \times id_X \times 0_{X\times Y} \times id_X \times 0_Y \times id_X); \mathsf{R}^4[f]; (\omega_{X\times Y\times X\times X} \times id_Y)$

for all $g : Y \to Z$ *and* $h : Z \to W$.

The careful reader will note that our formulation of these axioms differ from the formulation given by Cockett et al., but the two systems are equivalent and essentially result from our different formulation of left additive structure.

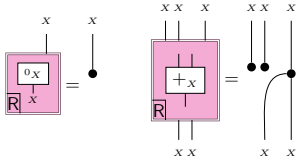
In words, RD1 and RD3 tell us the derivatives of the basic structural morphisms in a cartesian left additive category: RD1 gives the derivatives of the left additive structure, and RD3 gives the derivatives of the cartesian category. These are in some sense the base cases. Axioms RD4 and RD5 give induction steps, spelling out how the derivatives of morphisms composed in parallel (RD4) or in sequence (RD5) are constructed from the derivatives of its components. In particular, RD5 can be seen as a "reverse-mode chain rule." The remaining axioms are not as important for our purposes, but RD2 states that the reverse derivative is additive in its small-change argument, RD6 broadly states that the reverse derivative of any map is linear in its second argument (in the sense that it coincides with its own derivative) and RD7 states that partial (reverse) derivatives commute.

All of these axioms admit corresponding graphical presentations in terms of string diagrams, as can be found in Fig. 14. We only present here diagrams for the axioms that do not involve second derivatives, as they are the only ones relevant to our AD algorithm, but axioms RD6 and RD7 admit equally straightforward diagrammatic formulations.

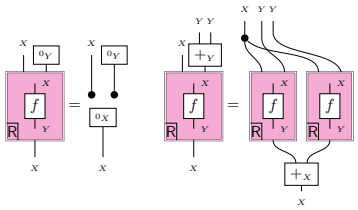
▶ **Lemma 19.** *Suppose* Σ *is an operational signature and suppose that for every operation* $g : X \to Y$ *in* Σ_1*, there is a well-defined reverse derivative* $\mathsf{R}[g]$ *morphism in* $\mathcal{A}_\Sigma$ *satisfying RD2, RD6, and RD7. Then* $\mathcal{A}_\Sigma$ *is a reverse differential category.*

In such a case, we will call Σ *a* differentiable signature.

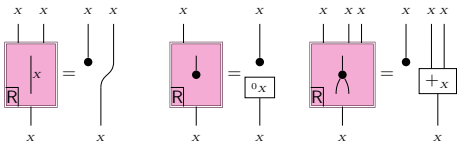
Proof. As noted previously, the RD axioms specify the derivatives of the structural morphisms from $\mathcal{A}_\Sigma$ and the derivatives of composites. It is easy to check that RD2, RD6, and RD7 hold for the derivatives of these structural morphisms. Then this reduces to the (straightforward) check that when two composable morphisms satisfy RD2, RD6, and RD7 then their composite does as well. ◀



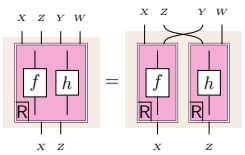
(a) RD1.



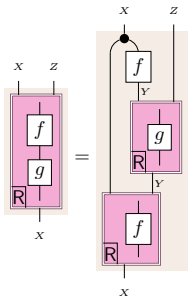
(b) RD2.



(c) RD3.



(d) RD4.



(e) RD5.

Figure 14 RDC axioms as string diagrams.

A Lattice-Theoretical View of Strategy Iteration

Paolo Baldan
University of Padova, Italy

Richard Eggert
Universität Duisburg-Essen, Germany

Barbara König
Universität Duisburg-Essen, Germany

Tommaso Padoan
University of Padova, Italy

Abstract

Strategy iteration is a technique frequently used for two-player games in order to determine the winner or compute payoffs, but to the best of our knowledge no general framework for strategy iteration has been considered. Inspired by previous work on simple stochastic games, we propose a general formalisation of strategy iteration for solving least fixpoint equations over a suitable class of complete lattices, based on MV-chains. We devise algorithms that can be used for non-expansive fixpoint functions represented as so-called min- respectively max-decompositions. Correspondingly, we develop two different techniques: strategy iteration from above, which has to solve the problem that iteration might reach a fixpoint that is not the least, and from below, which is algorithmically simpler, but requires a more involved correctness argument. We apply our method to solve energy games and compute behavioural metrics for probabilistic automata.

2012 ACM Subject Classification Theory of computation → Program verification; Theory of computation → Solution concepts in game theory

Keywords and phrases games, strategy iteration, fixpoints, energy games, behavioural metrics

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.7

Related Version *Full Version*: https://arxiv.org/abs/2207.09872 [7]

Funding Partially supported by the DFG project SpeQt and by the Ministero dell'Universtà e della Ricerca Scientifica of Italy, under Grant No. 201784YSZ5, PRIN2017 – ASPRA.

1 Introduction

Strategy iteration (or policy iteration) is a well known technique in computer science. It has been widely adopted for the solution of two-player games where the players, say Max and Min, aim at maximising and minimising, respectively, some payoff. In many cases there exists an optimal strategy for each player where no deviation is advisable as long as the other player plays optimally. We here assume a scenario where memoryless (or positional) strategies are sufficient. The general idea of strategy iteration is to iteratively fix a strategy for one player, compute the optimal answering strategy for the other player and then improve the strategy of the first player. As long as there are only finitely many strategies, an optimal strategy is bound to be found at some point. Such strategy iteration methods exist for Markov decision processes [21] and for a variety of games, such as simple stochastic games [13, 23, 1], (discounted) mean-payoff games [32, 10] and parity games [31, 28].

Similar ideas apply also to a wide range of different problems. For instance, the computation of behavioural distances for systems embodying quantitative information, e.g., time, probability or cost, is often based on some form of lifting of distances on states [2, 4, 8]. In turn the lifting relies on couplings which play the role of strategies and algorithms based on a progressive improvement of couplings have been devised [2, 3].

Motivating example. To help the intuition, we review simple stochastic games (SSGs) [13] and strategy iteration in that setting as discussed in [5, 6]. An SSG consists of a set of states V, partitioned in four subsets MIN, MAX, AV and $SINK$. States in $SINK$ (sink states) have no successor and yield a payoff in $[0,1]$. For states in AV (average states) the successor is determined by a probability distribution over V, i.e., intuitively, the environment makes a probabilistic choice. In a state in MIN, player Min chooses a successor trying to minimise the expected payoff, while in a state in MAX, it is the player Max that chooses, with the aim of maximising the expected payoff. An example of an SSG is in Fig. 1 on page 7.

When Min and Max play optimally, the expected payoff at each state is given by the least fixpoint of the function $\mathcal{V}\colon [0,1]^V \to [0,1]^V$, defined for $a\colon V \to [0,1]$ and $v \in V$ by

$$\mathcal{V}(a)(v) = \begin{cases} \max_{v \to v'} a(v') & v \in MAX \\ \min_{v \to v'} a(v') & v \in MIN \\ \sum_{v' \in V} p(v)(v') \cdot a(v') & v \in AV \\ c(v) & v \in SINK \end{cases}$$

with $p(v)(v')$ the probability of state v reaching v' and $c(v) \in [0,1]$ the payoff of sink state v.

The idea of strategy iteration from below, instantiated to this context, is to compute the least fixpoint $\mu\mathcal{V}$ via an iteration of the following kind:

1. Guess a strategy $\sigma\colon MAX \to V$ for player Max, i.e., fix a successor for states in MAX.
2. Compute the least fixpoint of $\mathcal{V}_\sigma : [0,1]^V \to [0,1]^V$, which is defined as $\mathcal{V}$ in all cases apart from $v \in MAX$, where we set $\mathcal{V}(a)(v) = a(\sigma(v))$. This fixpoint computation is simpler than the original one and it can be done efficiently via linear programming.
3. Based on $\mu\mathcal{V}_\sigma$, try to improve the strategy for Max. If the strategy does not change, we have computed a fixpoint of $\mathcal{V}$ and, since iteration is from below, this is necessarily the least fixpoint. If the strategy changes, continue with step 2.

A similar approach can be used for converging to the least fixpoint from above. In this case, it is now player Min who fixes a strategy which is progressively improved. This procedure is well-known to work for stopping games [13], i.e., SSGs where each combination of strategies ensures termination, since for these games $\mathcal{V}$ has a unique fixpoint. However, in general, when iterating from above the procedure may get stuck at some fixpoint which is not the least fixpoint of $\mathcal{V}$, a problem which is solved by the theory developed in [5] which can be used to "skip" this fixpoint and continue the iteration from there.

While, as explained above, the general idea of strategy iteration is used in many different settings, to the best of our knowledge a general definition of strategy iteration is still missing. The goal of the present paper is to provide a general and abstract formulation of an algorithm for strategy iteration, proved correct once and for all, which instantiates to a variety of problems. The key observation is that optimal strategies very often arise from some form of extremal (least or greatest) fixpoint of a suitable non-expansive function f over a complete MV-chain [27], the paradigmatic example being the real interval $[0,1]$ with the usual order. We propose a framework where the operation of fixing a strategy for one of the players is captured abstractly, in terms of so-called min- or max-decompositions of the function of interest. Then, we devise strategy iteration approaches which converge to the fixpoint of interest by successively improving the strategy for the chosen player. We will assume that the interest is in least fixpoints, but the theory can be dualised. We propose two strategy iteration algorithms that converge to the least fixpoint "from below" and "from above", respectively. As it happens for SSGs, in the latter case the iteration can reach a fixpoint

which is not the least. Clearly, whenever the function f of interest has a unique fixpoint this problem disappears. Moreover, in some cases, even though f has multiple fixpoints, it can be "patched" in a way that the modified function has the fixpoint of interest as its only fixpoint. Otherwise, we can rely on the results in [5] to check whether the reached fixpoint is the least one and whenever it is not, to get closer to the desired fixpoint and continue the iteration.

Strategy iteration approaches can be slow if compared to other algorithms, such as value iteration. However, the benefit of strategy iteration algorithms is that they allow an exact computation of the desired fixpoint, while other algorithms may never reach the sought-after extreme fixpoint but only converge towards it. This is the case, e.g., for simple stochastic games, where strategy iteration algorithms are the standard methods to obtain exact results. Additionally, strategy iteration, besides determining the fixpoint also singles out a strategy which allows one to obtain it, an information which is often of interest.

In summary, we propose the first, to the best of our knowledge, general definition of strategy iteration providing a lattice-theoretic formalisation of this technique. This requires to single out and solve in this general setting the fundamental challenges of these approaches, which already show up in earlier work on SSGs (see, e.g., [5, 10]). In the iteration from above, we may converge to a fixpoint that is not the least, while from below it is not straightforward to show that improving the strategy of Max leads to a larger fixpoint.

Known algorithms are rediscovered for SSGs and probabilistic automata [3]. Moreover new ones are obtained for energy games [9, 12] where movements in the game graph have an energy cost and the goal of one of the players is to avoid that the energy drops below zero. Given the number of different application domains where strategy iteration is or can be used, we feel that a general framework can unveil unexplored potentials. The two case studies (energy games and behavioural metrics) that we treat can be encoded into SSGs [3], but the obtained strategies have to be translated back to the original setting, which is not always trivial in general, and encodings usually come with a loss of efficiency. For instance, in order to solve SSGs a solver for linear programming is usually required, which is in general not needed for other applications.

The rest of the paper is structured as follows. In §2 we review some order-theoretic notions and recap some results from [5] for identifying least and greatest fixpoints. In §3 we devise two generalized strategy iteration algorithms, from above and from below, using SSGs (already treated in [6]) as a running example. In §4, we show how our technique applies to energy games, while in §5 we discuss an application to the computation of the behavioural distance for probabilistic automata.

Proofs and further material can be found in the full version of the paper [7].

2 Preliminaries on ordered structures and fixpoints

This section reviews some background used throughout the paper. This includes the basics of lattices and MV-algebras, where the functions of interest take values. We also recap some results from [5] useful for detecting if a fixpoint of a given function is the least (or greatest).

For X, Y sets, we denote by $\mathcal{P}(X)$ the powerset of X and $\mathcal{P}_{fin}(X)$ the set of finite subsets of X. Moreover, the set of functions from X to Y is denoted by either Y^X or $X \to Y$.

A partially ordered set $(P, \sqsubseteq)$ is often denoted simply as P, omitting the order relation. For a function $f : X \to P$, we will write $\arg\min_{x \in X} f(x)$ to denote the set of elements where f reaches the minimum, i.e., $\{x \in X \mid \forall y \in X.\, f(x) \sqsubseteq f(y)\}$ and, abusing the notation, we will write $z = \arg\min_{x \in X} f(x)$ instead of $z \in \arg\min_{x \in X} f(x)$.

The *join* and the *meet* of a subset $X \subseteq P$ (if they exist) are denoted $\bigsqcup X$ and $\bigsqcap X$.

A *complete lattice* is a partially ordered set $(\mathbb{L}, \sqsubseteq)$ such that each subset $X \subseteq \mathbb{L}$ admits a join $\bigsqcup X$ and a meet $\bigsqcap X$. A complete lattice $(\mathbb{L}, \sqsubseteq)$ always has a least element $\bot = \bigsqcap \mathbb{L}$ and a greatest element $\top = \bigsqcup \mathbb{L}$.

A function $f : \mathbb{L} \to \mathbb{L}$ is *monotone* if for all $l, l' \in \mathbb{L}$, if $l \sqsubseteq l'$ then $f(l) \sqsubseteq f(l')$. By Knaster-Tarski's theorem [29, Theorem 1], any monotone function on a complete lattice has a least fixpoint μf, characterised as the meet of all pre-fixpoints $\mu f = \bigsqcap\{l \mid f(l) \sqsubseteq l\}$ and, dually, a greatest fixpoint $\nu f = \bigsqcup\{l \mid l \sqsubseteq f(l)\}$, characterised as the join of all post-fixpoints. We denote by $Fix(f)$ the set of all fixpoints of f.

Given a set Y and a complete lattice $\mathbb{L}$, the set of functions $\mathbb{L}^Y = \{f \mid f : Y \to \mathbb{L}\}$, endowed with pointwise order, i.e., for $a, b \in \mathbb{L}^Y$, $a \sqsubseteq b$ if $a(y) \sqsubseteq b(y)$ for all $y \in Y$, is a complete lattice. We write $a \sqsubset b$ when $a \sqsubseteq b$ and $a \neq b$, i.e., for all $y \in Y$ we have $a(y) \sqsubseteq b(y)$ and $a(y) \sqsubset b(y)$ for some $y \in Y$.

We are also interested in the set of probability distributions $\mathcal{D}(Y) \subseteq [0,1]^Y$, i.e., functions $\beta : Y \to [0,1]$ such that $\sum_{y \in Y} \beta(y) = 1$.

An *MV-algebra* [27] is a tuple $\mathbb{M} = (M, \oplus, 0, \overline{(\cdot)})$ where $(M, \oplus, 0)$ is a commutative monoid and $\overline{(\cdot)} : M \to M$ maps each element to its *complement*, such that for all $x, y \in M$

1. $\overline{\overline{x}} = x$
2. $x \oplus \overline{0} = \overline{0}$
3. $\overline{(\overline{x} \oplus y)} \oplus y = \overline{(\overline{y} \oplus x)} \oplus x$.

We denote $1 = \overline{0}$ and subtraction $x \ominus y = \overline{\overline{x} \oplus y}$.

MV-algebras are endowed with a partial order, the so-called *natural order*, defined for $x, y \in M$, by $x \sqsubseteq y$ if $x \oplus z = y$ for some $z \in M$. When $\sqsubseteq$ is total, $\mathbb{M}$ is called an *MV-chain*. We will write $\mathbb{M}$ instead of M.

The natural order gives an MV-algebra a lattice structure where $\bot = 0$, $\top = 1$, $x \sqcup y = (x \ominus y) \oplus y$ and $x \sqcap y = \overline{\overline{x} \sqcup \overline{y}} = x \ominus (x \ominus y)$. We call the MV-algebra *complete* if it is a complete lattice, which is not true in general, e.g., $([0,1] \cap \mathbb{Q}, \leqslant)$.

▶ **Example 2.1.** A prototypical example of an MV-algebra is $([0,1], \oplus, 0, \overline{(\cdot)})$ where $x \oplus y = \min\{x + y, 1\}$, $\overline{x} = 1 - x$ and $x \ominus y = \max\{0, x - y\}$ for $x, y \in [0,1]$. The natural order is $\leqslant$ (less or equal) on the reals. Another example is $K = (\{0, \ldots, k\}, \oplus, 0, \overline{(\cdot)})$ where $n \oplus m = \min\{n + m, k\}$, $\overline{n} = k - n$ and $n \ominus m = \max\{n - m, 0\}$ for $n, m \in \{0, \ldots, k\}$. Both MV-algebras are complete and MV-chains.

We next briefly recap the theory from [5] which will be helpful in the paper for checking whether a fixpoint is the least or the greatest fixpoint of some underlying endo-function.

▶ Remark 2.2. Hereafter, unless stated otherwise, Y, Z will be assumed to be finite sets and $\mathbb{M}$ will be a complete MV-chain.

Given $a \in \mathbb{M}^Y$ we define its *norm* as $\|a\| = \max\{a(y) \mid y \in Y\}$. A function $f : \mathbb{M}^Y \to \mathbb{M}^Z$ is *non-expansive* if for all $a, b \in \mathbb{M}^Y$ it holds $\|f(b) \ominus f(a)\| \sqsubseteq \|b \ominus a\|$. It can be seen that non-expansive functions are monotone. A number of standard operators are non-expansive (e.g., constants, reindexing, max and min over a relation, average), and non-expansiveness is preserved by composition and disjoint union (see [5]). Given $Y' \subseteq Y$ and $\delta \in \mathbb{M}$, we write $\delta_{Y'}$ for the function defined by $\delta_{Y'}(y) = \delta$ if $y \in Y'$ and $\delta_{Y'}(y) = 0$, otherwise.

For a non-expansive endo-function $f : \mathbb{M}^Y \to \mathbb{M}^Y$ and $a \in \mathbb{M}^Y$, the theory in [5] provides a so-called a-approximation $f^a_\#$ of f, which is an endo-function over a suitable subset of Y. More precisely, define $[Y]^a = \{y \in Y \mid a(y) \neq 0\}$ and $\delta^a = \min\{a(y) \mid y \in [Y]^a\}$. For $0 \sqsubset \delta \in \mathbb{M}$ consider the functions $\alpha^{a,\delta} : \mathcal{P}([Y]^a) \to [a \ominus \delta, a]$ and $\gamma^{a,\delta} : [a \ominus \delta, a] \to \mathcal{P}([Y]^a)$, defined, for $Y' \in \mathcal{P}([Y]^a)$ and $b \in [a \ominus \delta, a]$, by

$$\alpha^{a,\delta}(Y') = a \ominus \delta_{Y'} \qquad \gamma^{a,\delta}(b) = \{y \in [Y]^a \mid a(y) \ominus b(y) \sqsupseteq \delta\}.$$

For a non-expansive function $f : \mathbb{M}^Y \to \mathbb{M}^Z$ and $\delta \in \mathbb{M}$, define $f^{a,\delta}_{\#} \colon \mathcal{P}([Y]^a) \to \mathcal{P}([Z]^{f(a)})$ as $f^{a,\delta}_{\#} = \gamma^{f(a),\delta} \circ f \circ \alpha^{a,\delta}$. The function $f^{a,\delta}_{\#}$ is antitone in the parameter δ and there exists a suitable value $\iota^a_f \sqsupset 0$, such that all functions $f^{a,\delta}_{\#}$ for $0 \sqsubset \delta \sqsubseteq \iota^a_f$ are equal. The function $f^a_{\#} := f^{a,\iota^a_f}_{\#}$ is called the *a-approximation* of f. When $\delta \sqsubseteq \delta_a$, the pair $\langle \alpha^{a,\delta}, \gamma^{a,\delta} \rangle$ is a Galois connection, a notion at the heart of abstract interpretation [14, 15], and $f^{a,\delta}_{\#} = \gamma^{f(a),\delta} \circ f \circ \alpha^{a,\delta}$ is the best correct approximation of f.

Intuitively, given some Y', the set $f^a_{\#}(Y')$ contains the points where a decrease of the values of a on the points in Y' "propagates" through the function f. The greatest fixpoint of $f^a_{\#}$ gives us the subset of Y where such a decrease is propagated in a cycle (so-called "vicious cycle"). Whenever $\nu f^a_{\#}$ is non-empty, one can argue that a cannot be the least fixpoint of f since we can decrease the value in all elements of $\nu f^a_{\#}$, obtaining a smaller prefixpoint. Interestingly, for non-expansive functions, it is shown in [5] that also the converse holds, i.e., emptiness of the greatest fixpoint of $f^a_{\#}$ implies that a is the least fixpoint.

▶ **Theorem 2.3** (soundness and completeness for fixpoints). *Let $\mathbb{M}$ be a complete MV-chain, Y a finite set and $f : \mathbb{M}^Y \to \mathbb{M}^Y$ be a non-expansive function. Let $a \in \mathbb{M}^Y$ be a fixpoint of f. Then $\nu f^a_{\#} = \varnothing$ if and only if $a = \mu f$.*

Using the above theorem we can check whether some fixpoint a of f is the least fixpoint. Whenever a is a fixpoint, but not yet the least fixpoint of f, it can be decreased by a fixed value in $\mathbb{M}$ (see [5] for the details) on the points in $\nu f^a_{\#}$ to obtain a smaller pre-fixpoint.

▶ **Lemma 2.4.** *Let $\mathbb{M}$ be a complete MV-chain, $f : \mathbb{M}^Y \to \mathbb{M}^Y$ a non-expansive function, $a \in \mathbb{M}^Y$ a fixpoint of f, and let $f^a_{\#}$ be the corresponding a-approximation. If a is not the least fixpoint and thus $\nu f^a_{\#} \neq \varnothing$ then there is $0 \sqsubset \delta \in \mathbb{M}$ such that $a \ominus \delta_{\nu f^a_{\#}}$ is a pre-fixpoint of f.*

In the following we will use this result as a "black box": we assume that given f and a fixpoint a of f we can determine whether $a = \mu f$ and, if not, obtain $a' \sqsubset a$ such $f(a') \sqsubseteq a'$.

The above theory can easily be dualised (see [5] for the details of the dual view).

3 Generalized strategy iteration

In this section we develop two strategy iteration techniques for determining least fixpoints. The first technique requires a so-called min-decomposition and approaches the least fixpoint from above, while the second uses a max-decomposition to ascend to the least fixpoint from below.

Hence fixpoint iteration from above is seen strictly from the point of view of the Min player, while fixpoint iteration from below is from the view of the Max player, who want to minimize respectively maximize the payoff. The player starts by guessing a strategy, which in the case of the Min (Max) player over-approximates (under-approximates) the true payoff. This strategy is then locally improved at each iteration based on the payoff produced by the player following such a strategy. That is, we compute fixpoints for a fixed strategy, which in a two-player game means that the opponent plays optimally. When the set of strategies is finite (or, at least, the search can be restricted to a finite set), an optimal strategy will be found at some point.

3.1 Function decomposition

We next introduce the setting where the generalisations of strategy iteration will be developed. We assume that the game we are interested in is played on a finite set of positions Y and the payoff at each position is an element of a suitable complete MV-chain $\mathbb{M}$. This payoff

is given by a function in $\mathbb{M}^Y$ that can be characterised as the least fixpoint of a monotone function $f : \mathbb{M}^Y \to \mathbb{M}^Y$. If we concentrate on the Min player, each position $y \in Y$ is assigned a set of functions $H_{\min}(y) \subseteq (\mathbb{M}^Y \to \mathbb{M})$ where each function $h \in H_{\min}(y)$ is one possible option that can be chosen by Min. Given $a \colon Y \to \mathbb{M}$ as the current estimate of the payoff, $h(a)$ is the resulting payoff at y. If the player does not have a choice, this set is a singleton. Since it is the aim of Min to minimise she will choose an h such that $h(a)$ is minimal.

▶ **Definition 3.1** (min-decomposition). *Let Y be a finite set and $\mathbb{M}$ be a complete MV-chain. Given a function $f : \mathbb{M}^Y \to \mathbb{M}^Y$, a* min-decomposition *of f is a function $H_{\min} : Y \to \mathcal{P}_{fin}(\mathbb{M}^Y \to \mathbb{M})$ such that for all $y \in Y$ the set $H_{\min}(y)$ consists only of monotone functions and for all $a \in \mathbb{M}^Y$ it holds $f(a)(y) = \min_{h \in H_{\min}(y)} h(a)$.*

Observe that any monotone function $f : \mathbb{M}^Y \to \mathbb{M}^Y$ admits a trivial min-decomposition I defined by $I(y) = \{h_y\}$ where $h_y(a) = f(a)(y)$ for all $a \in \mathbb{M}^Y$.

Whenever all $h \in H_{\min}(y)$ are not only monotone, but also non-expansive, it can be shown easily that f is also non-expansive and we can obtain an approximation as discussed in §2. Max-decompositions, with analogous properties, are defined dually, i.e. $H_{\max} : Y \to \mathcal{P}_{fin}(\mathbb{M}^Y \to \mathbb{M})$ and $f(a)(y) = \max_{h \in H_{\max}(y)} h(a)$.

Fixing a strategy can be seen as fixing, for all $y \in Y$, some element in $H_{\min}(y)$.

▶ **Definition 3.2** (strategy). *Let Y be a finite set, $\mathbb{M}$ be a complete MV-chain, $f : \mathbb{M}^Y \to \mathbb{M}^Y$ and let $H_{\min} : Y \to \mathcal{P}_{fin}(\mathbb{M}^Y \to \mathbb{M})$ be a min-decomposition of f. A* strategy *in $H_{\min}$ is a function $C : Y \to (\mathbb{M}^Y \to \mathbb{M})$ such that for all $y \in Y$ it holds that $C(y) \in H_{\min}(y)$. For a fixed C we define $f_C : \mathbb{M}^Y \to \mathbb{M}^Y$ as $f_C(a)(y) = C(y)(a)$ for all $a \in \mathbb{M}^Y$ and $y \in Y$.*

Strategies in a max-decomposition are defined dually.

The letter C stands for "choice" and typically μf_C is easier to compute than μf.

▶ **Example 3.3.** As a running example for illustrating our theory and the resulting algorithms we will use *simple stochastic games* (SSGs). We first show that they fall into the framework. Fix an SSG with a finite set V of states, partitioned into MIN, MAX, AV (average) and $SINK$. Successors of MIN and MAX states are given by a relation $\to \subseteq (MIN \cup MAX) \times V$, while $p : AV \to [0,1]^V$ maps each $v \in AV$ to a distribution $p(v) \in \mathcal{D}(V)$. Finally, $c : SINK \to [0,1]$ provides the payoff of sink states.

The fixpoint function $\mathcal{V} : [0,1]^V \to [0,1]^V$, as defined in the introduction, admits a min-decomposition $H_{\min} \colon V \to \mathcal{P}_{fin}(\mathbb{M}^V \to \mathbb{M})$ defined for all $a \in \mathbb{M}^Y$ as follows:

- for $v \in MIN$, $H_{\min}(v) = \{h_{v'} \mid v \to v'\}$ with $h_{v'}(a) = a(v')$;
- for $v \in MAX$, $H_{\min}(v) = \{h\}$ with $h(a) = \max_{v \to v'} a(v')$;
- for $v \in AV$, $H_{\min}(v) = \{h\}$ with $h(a) = \sum_{v' \in V} p(v)(v') \cdot a(v')$;
- for $v \in SINK$, $H_{\min}(v) = \{h\}$ with $h(a) = c(v)$.

A max-decomposition can be defined dually.

For instance, consider the SSG in Fig. 1 where $V = \{\mathbf{1}, \boldsymbol{\varepsilon}, \boldsymbol{av}, \boldsymbol{max}, \boldsymbol{min}\}$ with the obvious partitioning. The fixpoint function is $\mathcal{V} \colon [0,1]^V \to [0,1]^V$ defined, for $a \in [0,1]^V$, by

$$\mathcal{V}(a)(\mathbf{1}) = 1 \qquad \mathcal{V}(a)(\boldsymbol{\varepsilon}) = \varepsilon \qquad \mathcal{V}(a)(\boldsymbol{av}) = \frac{1}{2}a(\boldsymbol{min}) + \frac{1}{2}a(\boldsymbol{max})$$
$$\mathcal{V}(a)(\boldsymbol{max}) = \max\{a(\boldsymbol{\varepsilon}), a(\boldsymbol{av})\} \qquad \mathcal{V}(a)(\boldsymbol{min}) = \min\{a(\mathbf{1}), a(\boldsymbol{av})\}.$$

The min-decomposition defined in general above, in this case is $H_{\min} \colon V \to \mathcal{P}_{fin}(\mathbb{M}^V \to \mathbb{M})$ defined for all $a \in \mathbb{M}^Y$ as follows: for $v \in V \setminus \{\boldsymbol{min}\}$, we let $H_{\min}(v) = \{h\}$ with $h(a) = \mathcal{V}(a)(v)$, while $H_{\min}(\boldsymbol{min}) = \{h_{\mathbf{1}}, h_{\boldsymbol{av}}\}$ with $h_{\mathbf{1}}(a) = a(\mathbf{1})$ and $h_{\boldsymbol{av}}(a) = a(\boldsymbol{av})$. All strategies in

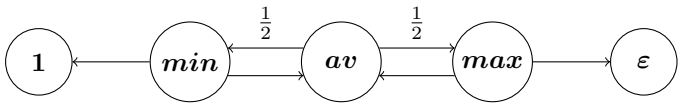


Figure 1 An example of a simple stochastic game. States $\mathbf{1}$, $\boldsymbol{\varepsilon}$ have payoff 1, $\varepsilon > 0$ respectively.

$H_{\min}$ assign to every state $v \in V\backslash\{\boldsymbol{min}\}$ the only element in $H_{\min}(v)$. Hence they are determined by the value on state $\boldsymbol{min}$: thus there are two strategies $C_1^{\min}, C_2^{\min}$ in $H_{\min}$ with $C_1^{\min}(\boldsymbol{min}) = h_{\mathbf{1}}$ and $C_2^{\min}(\boldsymbol{min}) = h_{\boldsymbol{av}}$.

Dually, a max-decomposition $H_{\max} \colon V \to \mathcal{P}_{fin}(\mathbb{M}^V \to \mathbb{M})$ is defined for all $a \in \mathbb{M}^Y$ as follows: $H_{\max}(v) = \{h\}$ with $h(a) = \mathcal{V}(a)(v)$ for all $v \in V\backslash\{\boldsymbol{max}\}$ and $H_{\max}(\boldsymbol{max}) = \{h_{\boldsymbol{\varepsilon}}, h_{\boldsymbol{av}}\}$ with $h_{\boldsymbol{\varepsilon}}(a) = a(\boldsymbol{\varepsilon})$ and $h_{\boldsymbol{av}}(a) = a(\boldsymbol{av})$. Again, there are two strategies $C_1^{\max}$ and $C_2^{\max}$ in $H_{\max}$ that differ for the value assigned to $\boldsymbol{max}$: $C_1^{\max}(\boldsymbol{max}) = h_{\boldsymbol{\varepsilon}}$ and $C_2^{\max}(\boldsymbol{max}) = h_{\boldsymbol{av}}$.

3.2 Strategy iteration from above

In this section we propose a generalized strategy iteration algorithm from above. It is based on a min-decomposition of the function and, intuitively, at each iteration the player Min improves her strategy. An issue here is that this iteration may get stuck at a fixpoint strictly larger than the least one. Recognising and overcoming this problem, thus continuing the iteration until the least fixpoint is reached, requires the theory described in §2.

The basic result that motivates strategy iteration from above is a characterisation of the least fixpoint of a function in terms of a min-decomposition.

▶ **Proposition 3.4** (least fixpoint from min-decompositions). *Let Y be a finite set, $\mathbb{M}$ a complete MV-chain, $f : \mathbb{M}^Y \to \mathbb{M}^Y$ a monotone function and let $H_{\min} : Y \to \mathcal{P}_{fin}(\mathbb{M}^Y \to \mathbb{M})$ be a min-decomposition of f. Then $\mu f = \min\{\mu f_C \mid C \text{ is a strategy in } H_{\min}\}$.*

Although we do not focus on complexity issues, we observe that – under suitable assumptions – we can show that given a function f as a min-decomposition, the problem of checking whether $\mu f \sqsubseteq b$ for some bound $b \in \mathbb{M}^Y$ is in NP. For each $y \in Y$ we can nondeterministically guess $C(y) \in H_{\min}(y)$ thus defining a strategy. Assuming that the computation of μf_C is polynomial, we can thus determine in non-deterministic polynomial time (in the size of the representation of f) whether $\mu f \sqsubseteq \mu f_C \sqsubseteq b$.

Now in order to compute the least fixpoint, the idea is to start from some (arbitrary) strategy, say C_0, in $H_{\min}$. At each iteration, if the current strategy is C_i one tries to construct, on the basis of μf_{C_i}, a new strategy C_{i+1} which improves C_i, in the sense that $\mu f_{C_{i+1}}$ becomes smaller. This motivates the notion of improvement.

▶ **Definition 3.5** (min-improvement). *Let $f : \mathbb{M}^Y \to \mathbb{M}^Y$ be a monotone function, where Y is a finite set and $\mathbb{M}$ a complete MV-chain, and let $H_{\min}$ be a min-decomposition. Given strategies C, C' in $H_{\min}$, we say that C' is a* min-improvement *of C if $f_{C'}(\mu f_C) \sqsubset \mu f_C$. It is called a* stable *min-improvement if in addition $C'(y) = C(y)$ for all $y \in Y$ such that $f_{C'}(\mu f_C)(y) = \mu f_C(y)$. We denote by $imp_{\min}(C)$ (respectively $imp^s_{\min}(C)$) the set of (stable) min-improvements of C.*

The notion of stability will turn out to be useful later, for performing strategy iteration from below (as explained in the next section). In a stable min-improvement, the player is only allowed to switch the strategy in a state if this yields a strictly better payoff. Interestingly,

instances of this notion are adopted, more or less implicitly, in other strategy improvement algorithms in the literature (cf. [1, Definition 13] and the way in which improvements are computed in [10]). Clearly $imp^s_{\min}(C) \subseteq imp_{\min}(C)$. In addition, it can be easily seen that there exists a stable min-improvement as long as there is any improvement.

▶ Remark 3.6 (obtaining min-improvements). For a strategy C, if $imp_{\min}(C) \neq \varnothing$, one can obtain a min-improvement of C by taking $C' \neq C$ defined as $C'(y) = \arg\min_{h \in H_{\min}(y)} h(\mu f_C)$ and a stable min-improvement as:

$$C'(y) = \begin{cases} C(y) & \text{if } f(\mu f_C)(y) = \mu f_C(y) \\ \arg\min_{h \in H_{\min}(y)} h(\mu f_C) & \text{otherwise} \end{cases}$$

There could be several $h \in H_{\min}(y)$ where $h(\mu f_C)$ is minimal. Any such choice is valid.

We next show that, as suggested by the terminology, a min-improvement leads to a smaller least fixpoint.

▶ **Lemma 3.7** (min-improvements reduce fixpoints). *Let Y be a finite set, $\mathbb{M}$ a complete MV-chain, $f : \mathbb{M}^Y \to \mathbb{M}^Y$ a monotone function and $H_{\min}$ a min-decomposition of f. Given a strategy C in $H_{\min}$ and a min-improvement $C' \in imp_{\min}(C)$ it holds $\mu f_{C'} \sqsubset \mu f_C$.*

Thus, once the strategy can be improved, we will get closer to the least fixpoint of f. We next show that an improvement of the current strategy exists as long as we have not encountered a fixpoint of f.

▶ **Lemma 3.8** (min-improvements exist for non-fixpoints). *Let Y be a finite set, $\mathbb{M}$ a complete MV-chain, $f : \mathbb{M}^Y \to \mathbb{M}^Y$ a monotone function and $H_{\min}$ a min-decomposition. Given a strategy C in $H_{\min}$, the following are equivalent:*

1. $\mu f_C \notin Fix(f)$
2. $imp_{\min}(C) \neq \varnothing$
3. $f(\mu f_C) \sqsubset \mu f_C$

The above result suggests an algorithm for computing a fixpoint of a function $f : \mathbb{M}^Y \to \mathbb{M}^Y$ on the basis of some min-decomposition. The idea is to guess some strategy C, determine μf_C and check $imp_{\min}(C)$. If this set is empty we have reached some fixpoint, otherwise choose $C' \in imp_{\min}(C)$ for the next iteration. Note that for this algorithm it is irrelevant whether we use min-improvements or restrict to stable min-improvements. We also note that this procedure and the developed theory to this point work for monotone functions $f \colon \mathbb{L}^Y \to \mathbb{L}^Y$ where $\mathbb{L}$ is a complete lattice.

When we are interested in the least fixpoint and the function admits many fixpoints, the sketched algorithm determines a fixpoint which might not be the desired one. Exploiting the theory from [5], summarised in §2, we can refine the algorithm to ensure that it computes μf. For this, we have to work with non-expansive functions $f : \mathbb{M}^Y \to \mathbb{M}^Y$ with $\mathbb{M}$ being a complete MV-chain and Y a finite set. In fact, in this setting, given a fixpoint of f, say $a \in \mathbb{M}^Y$, relying on Theorem 2.3, we can check whether it is the least fixpoint of f. In case it is not, we can "improve" it obtaining a smaller pre-fixpoint of f in a way that we can continue the iteration from there. The resulting algorithm is reported in Fig. 2. Observe that in step 2b we clearly do not need to compute all improvements. Rather, a min-improvement, whenever it exists, can be determined, on the basis of Definition 3.5, using μf_{C_i} computed in step 2a. Moreover step 2c relies on Theorem 2.3 and Lemma 2.4.

▶ **Theorem 3.9** (least fixpoint, from above). *Let Y be a finite set, $\mathbb{M}$ a complete MV-chain, $f : \mathbb{M}^Y \to \mathbb{M}^Y$ be a non-expansive function and let $H_{\min}$ be a min-decomposition of f. The algorithm in Fig. 2 terminates and computes μf.*

1. Initialize: guess a strategy C_0, $i := 0$
2. **iterate**
   a. determine μf_{C_i}
   b. **if** $imp_{\min}(C_i) \neq \varnothing$, let $C_{i+1} \in imp_{\min}(C_i)$; $i := i + 1$; **goto** (a)
   c. **else if** $\mu f_{C_i} \neq \mu f$ let $a \sqsubset \mu f_{C_i}$ be a pre-fixpoint of f and determine C_{i+1} via

$$C_{i+1}(y) = \arg \min_{h \in H_{\min}(y)} h(a)$$

   $i := i + 1$; **goto** 2.(a)
   d. **else stop**

Figure 2 Computing the least fixpoint, from above.

Termination easily follows from the fact that the number of strategies is finite (since Y is finite and $H_{\min}(y)$ is finite for all $y \in Y$). Given that at any iteration the fixpoint decreases, no strategy can be considered twice, and thus the number of iterations is bounded by the number of strategies.

▶ **Example 3.10.** Let us revisit Example 3.3 and the fixpoint function $\mathcal{V}$ defined there. Its least fixpoint satisfies $\mu\mathcal{V}(\mathbf{1}) = 1$ and $\mu\mathcal{V}(v) = \varepsilon$ for any $v \in V \backslash \{\mathbf{1}\}$.

The optimal strategy for Min is to choose $\boldsymbol{av}$ as its successor since this forces Max to exit the cycle formed by $\boldsymbol{min}, \boldsymbol{av}, \boldsymbol{max}$ to $\boldsymbol{\varepsilon}$, yielding a payoff of ε for these states. If Max would behave in a way that the play keeps cycling he would obtain a payoff of 0, which is suboptimal.

We now apply our algorithm. We start by guessing a strategy for Min, so we assume $C_0(\boldsymbol{min}) = h_{\mathbf{1}}$, i.e. $C_0 = C_1^{\min}$ (for the naming of the strategies we refer to Example 3.3). The least fixpoint $\mu\mathcal{V}_{C_0}$ can be found by solving the following linear program:

$$\min \sum_{v \in V} a(v) \qquad a(\mathbf{1}) = 1 \qquad a(\boldsymbol{\varepsilon}) = \varepsilon \qquad a(\boldsymbol{av}) = \frac{1}{2}a(\boldsymbol{min}) + \frac{1}{2}a(\boldsymbol{max})$$
$$a(\boldsymbol{max}) \geqslant a(\boldsymbol{\varepsilon}) \quad a(\boldsymbol{max}) \geqslant a(\boldsymbol{av}) \quad a(\boldsymbol{min}) = a(\mathbf{1})$$

with $0 \leqslant a(v) \leqslant 1$ for $v \in V$, which yields $\mu\mathcal{V}_{C_0}(\boldsymbol{\varepsilon}) = \varepsilon$ and $\mu\mathcal{V}_{C_0}(v) = 1$ for all $v \in V \backslash \{\boldsymbol{\varepsilon}\}$. Now $\mu\mathcal{V}_{C_0}$ is a fixpoint of $\mathcal{V}$ – but not the least – and thus we find the vicious cycle formed by $\boldsymbol{min}, \boldsymbol{av}, \boldsymbol{max}$, i.e. $\nu\mathcal{V}_{\#}^{\mu\mathcal{V}_{C_0}} = \{\boldsymbol{min}, \boldsymbol{av}, \boldsymbol{max}\}$ and decrease the values of those states in a by δ, i.e. we obtain $a = \mu\mathcal{V}_{C_0} \ominus \delta_{\{\boldsymbol{min}, \boldsymbol{av}, \boldsymbol{max}\}}$. This results in $a(\mathbf{1}) = 1$, $a(\boldsymbol{\varepsilon}) = \varepsilon$ and $a(v) = 1 - \delta$ for all $v \in V \backslash \{\mathbf{1}, \boldsymbol{\varepsilon}\}$. Any $\delta \in (0, 1 - \varepsilon]$ is a valid choice.

Computing $C_1(y) = \arg\min_{h \in H_{\min}(y)} h(a)$ yields the strategy $C_1 = C_2^{\min}$, i.e. $C_1(\boldsymbol{min}) = h_{\boldsymbol{av}}$. By linear programming (replace $a(\boldsymbol{min}) = a(\mathbf{1})$ by $a(\boldsymbol{min}) = a(\boldsymbol{av})$) we obtain $\nu\mathcal{V}_{\#}^{\mu f_{C_1}} = \varnothing$, thus $\mu\mathcal{V}_{C_1} = \mu\mathcal{V}$ and the algorithm terminates.

3.3 Strategy iteration from below

Here we present a different generalized strategy iteration algorithm approaching the least fixpoint from below. Intuitively, now it is player Max who improves his strategy step by step, creating an ascending chain of least fixpoints which reaches the least fixpoint of the underlying function f. Despite the fact that in this case we cannot get stuck at a fixpoint which is not the least, the correctness argument is more involved.

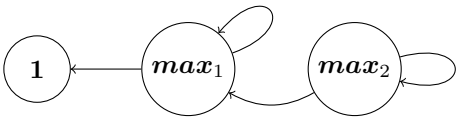


Figure 3 An example of a simple stochastic game where state **1** has payoff 1.

We will deal with max-decompositions of a function and we will need a notion of (stable) max-improvement which is naturally defined as a dualisation of the notion of (stable) min-improvement (Definition 3.5).

▶ **Definition 3.11** (max-improvement). *Let Y be a finite set, $\mathbb{M}$ a complete MV-chain, $f : \mathbb{M}^Y \to \mathbb{M}^Y$ a monotone function and $H_{\max}$ a max-decomposition. Given C, C' strategies in $H_{\max}$, we say that C' is a* max-improvement *of C if $\mu f_C \sqsubset f_{C'}(\mu f_C)$. It is called a* stable max-improvement *if in addition $C'(y) = C(y)$ for all $y \in Y$ such that $f_{C'}(\mu f_C)(y) = \mu f_C(y)$. We denote by $imp_{\max}(C)$ (respectively $imp^s_{\max}(C)$) the set of (stable) max-improvements of C.*

When iterating from above it was rather easy to show that given a strategy C and a min-improvement C', the latter yields a smaller least fixpoint $\mu f_{C'} \sqsubset \mu f_C$ (Lemma 3.7). Observing that μf_C is a pre-fixpoint of $f_{C'}$ was enough to prove this.

Here, however, we cannot simply dualise the argument. If C' is a max-improvement of C, we obtain that μf_C is a post-fixpoint of $f_{C'}$ which, in general, does not guarantee $\mu f_{C'} \sqsupset \mu f_C$. We have to resort to stable max-improvements and, in order to show that such improvements in fact yield greater least fixpoints, we need, again, to use the theory reviewed in §2. Hence, we have to work with non-expansive functions $f \colon \mathbb{M}^Y \to \mathbb{M}^Y$ where $\mathbb{M}$ is a complete MV-chain.

▶ **Lemma 3.12** (max-improvements increase fixpoints). *Let Y be a finite set, $\mathbb{M}$ a complete MV-chain, $f : \mathbb{M}^Y \to \mathbb{M}^Y$ a non-expansive function and $H_{\max}$ a max-decomposition. Given a strategy C in $H_{\max}$ and a stable max-improvement $C' \in imp^s_{\max}(C)$, then $\mu f_C \sqsubset \mu f_{C'}$.*

▶ **Example 3.13.** We note that working with max-improvements which are stable is essential for the validity of Lemma 3.12 above. In fact, consider the SSG in Figure 3 where $\boldsymbol{max}_1, \boldsymbol{max}_2 \in MAX$ and $\mathbf{1} \in SINK$, with reward 1. Let C be the strategy for Max where $\boldsymbol{max}_1$ and $\boldsymbol{max}_2$ have as successors $\mathbf{1}$ and $\boldsymbol{max}_2$, respectively. It is easy to see that $\mu\mathcal{V}_C(\mathbf{1}) = \mu\mathcal{V}_C(\boldsymbol{max}_1) = 1$ and $\mu\mathcal{V}_C(\boldsymbol{max}_2) = 0$. Now, an improvement in $imp_{\max}(C)$ can be the strategy C' which chooses $\boldsymbol{max}_1$ as a successor for both $\boldsymbol{max}_1$ and $\boldsymbol{max}_2$. Then we have $\mu\mathcal{V}_{C'}(\boldsymbol{max}_1) = \mu\mathcal{V}_{C'}(\boldsymbol{max}_2) = 0$, hence $\mu\mathcal{V}_C \sqsupset \mu\mathcal{V}_{C'}$. The reason why this happens is that C' is not a stable improvement of C since it uselessly changes the successor of $\boldsymbol{max}_1$ from $\mathbf{1}$ to $\boldsymbol{max}_1$, both mapped to 1 by $\mu\mathcal{V}_C$. A stable improvement of C is C'' where $\boldsymbol{max}_1$ and $\boldsymbol{max}_2$ have as successors $\mathbf{1}$ and $\boldsymbol{max}_1$, respectively. Then it can be seen that $\mu\mathcal{V}_{C''}(v) = 1$ for all states.

Relying on Lemma 3.12, we can easily prove the dual of Lemma 3.8, showing that a strategy admits a stable max-improvement as long as we have not reached a fixpoint of f.

▶ **Lemma 3.14** (max-improvements exist for non-fixpoints). *Let Y be a finite set, $\mathbb{M}$ a complete MV-chain, $f : \mathbb{M}^Y \to \mathbb{M}^Y$ a monotone function and $H_{\max}$ a max-decomposition. Given a strategy C in $H_{\max}$, the following are equivalent:*

1. $\mu f_C \notin Fix(f)$
2. $imp^s_{\max}(C) \neq \varnothing$
3. $\mu f_C \sqsubset f(\mu f_C)$

1. Initialize: guess a strategy C_0, $i := 0$
2. **iterate**
   a. determine μf_{C_i}
   b. **if** $imp^s_{\max}(C_i) \neq \varnothing$, let $C_{i+1} \in imp^s_{\max}(C_i)$; $i := i + 1$; **goto** (a)
   c. else **stop**

Figure 4 Computing the least fixpoint, from below.

To summarise, given a strategy C with $\mu f_C \notin Fix(f)$ we can construct a strategy C' with $\mu f_C \sqsubset \mu f_{C'}$. This creates an ascending chain of least fixpoints and since there are only finitely many strategies we will at some point find an optimal strategy C^* with $\mu f_{C^*} = \mu f$.

▶ **Proposition 3.15** (least fixpoint from max-decomposition). *Let Y be a finite set, $\mathbb{M}$ a complete MV-chain, $f : \mathbb{M}^Y \to \mathbb{M}^Y$ a non-expansive function and let $H_{\max} : Y \to \mathcal{P}_{fin}(\mathbb{M}^Y \to \mathbb{M})$ be a max-decomposition of f. Then $\mu f = \max\{\mu f_C \mid C \text{ is a strategy in } H_{\max}\}$.*

The above results lead us to a generalised strategy iteration algorithm which approaches the least fixpoint from below.

▶ **Theorem 3.16** (least fixpoint, from below). *Let Y be a finite set, $\mathbb{M}$ a complete MV-chain, $f : \mathbb{M}^Y \to \mathbb{M}^Y$ be a non-expansive function and let $H_{\max}$ be a max-decomposition of f. The algorithm in Fig. 4 terminates and computes μf.*

The iteration from below may seem more appealing since it cannot get stuck at any fixpoint of f. However, it has to be noted that the computation of μf_C – for a chosen strategy C – may be more difficult than before, which is illustrated by the following example.

▶ **Example 3.17.** Let us apply the above algorithm to the SSG in Example 3.3. Recall that the least fixpoint is given by $\mu\mathcal{V}(\mathbf{1}) = 1$ and $\mu\mathcal{V}(v) = \varepsilon$ for all $v \in V\backslash\{\mathbf{1}\}$.

We start by guessing a strategy for Max, so we assume $C_0(\boldsymbol{max}) = h_{\boldsymbol{av}}$, i.e. $C_0 = C_2^{\boldsymbol{max}}$. With this choice of strategy, Min is able to keep the game going infinitely in the cycle formed by $\boldsymbol{min}, \boldsymbol{av}, \boldsymbol{max}$ and thus payoff 0 is obtained. Now $\mu\mathcal{V}_{C_0}$ is given by $\mu\mathcal{V}_{C_0}(\boldsymbol{\varepsilon}) = \varepsilon$, $\mu\mathcal{V}_{C_0}(\mathbf{1}) = 1$ and $\mu\mathcal{V}_{C_0}(v) = 0$ for all $v \in V\backslash\{\boldsymbol{\varepsilon}, \mathbf{1}\}$. We note that $\mu\mathcal{V}_{C_0}$ cannot immediately be computed via linear programming, but there is a way to modify the fixpoint equation to have a unique fixpoint and hence linear programming can be used again [5]. This is done by precomputing states from which Min can force a non-terminating play and assigning payoff value 0 to them. Next, Max updates his strategy and we obtain $C_1 = C_1^{\max}$. As above we can compute $\mu\mathcal{V}_{C_1}$ – which, this time, equals $\mu\mathcal{V}$ – via linear programming.

▶ Remark 3.18. Given μf (without the corresponding strategy) an interesting question is how one can derive optimal strategies for Min or Max. Note that each presented strategy iteration algorithm only produces an optimal strategy for one player, but not for the other.

It is rather easy to find an optimal strategy with respect to $H_{\min}$. We can simply compute $C^*(y) = \arg\min_{h \in H_{\min}(y)} h(\mu f)$ which yields some optimal strategy C^*, i.e. $\mu f_{C^*} = \mu f$. It is enough to choose some minimum, even if this is ambiguous and there are several choices, each of which produces an optimal strategy. The strategy C^* is optimal since μf is a pre-fixpoint of f_{C^*} and $\mu f = \mu f_{C^*}$ follows from Proposition 3.4.

On the other hand, given μf, we cannot easily obtain an optimal strategy in $H_{\max}$. We will discuss in §4.1 (Example 4.1) that defining $C^*(y) = \arg\max_{h \in H_{\max}(y)} h(\mu f)$ for an arbitrary h where the value is maximal does not work in general.

4 Application: energy games

In this section we examine energy games [16] and show how both strategy iterations in §3 can be applied to solve energy games and have the advantage of providing us not only with the value vector, but also with a strategy, which is interesting, in particular, for Player 1 (Max).

Energy games are two-player games on finite graphs where Player 0 (Min) wants to keep the game going forever, while Player 1 (Max) wants it to stop eventually. Each state belongs to one player where he chooses the successor and each traversed edge reduces or increases the energy level by some integer. The game stops when an edge is taken which reduces the energy level below 0. The main question is how much initial energy is needed, such that Player 0 can keep the game going forever. It is possible to require an initial energy level of infinity.

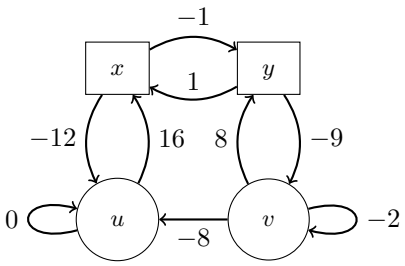
4.1 Introduction to energy games

A game graph is a tuple $\Gamma = (V_0, V_1, E, w)$ where $V = V_0 \cup V_1$, with $V_0 \cap V_1 = \varnothing$, is the set of states, $E \subseteq V \times V$ is the set of edges and $w\colon E \to \mathbb{Z}$ is a weight function. We assume that each state has at least one outgoing edge. We define $post(v) = \{v' \in V \mid (v, v') \in E\} \neq \varnothing$ for $v \in V$. States in V_0 and V_1 are owned by Player 0 and Player 1 respectively. Moving from state v to state v' will change the energy level by adding the value $w(v, v')$. If this value is positive, some energy is gained, otherwise energy decreases. It is the aim of Player 0 to keep the energy level from getting negative. An energy game is an infinite play on a game graph Γ and we note that optimal positional strategies exist for both players [16].

The solution of Γ is a function $g_\Gamma\colon V \to \mathbb{N}^\infty$ (where $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$) that assigns to each state the least energy level which is sufficient for Player 0 to keep the game going, independently of the chosen strategy of Player 1. It is known that the solution is the least fixpoint of the following function $\bar{\mathcal{E}}\colon (\mathbb{N}^\infty)^V \to (\mathbb{N}^\infty)^V$, defined as

$$\bar{\mathcal{E}}(a)(v) = \begin{cases} \min\limits_{v' \in post(v)} \max\{a(v') - w(v, v'), 0\} & \text{if } v \in V_0 \\ \max\limits_{v' \in post(v)} \max\{a(v') - w(v, v'), 0\} & \text{if } v \in V_1 \end{cases}$$

▶ **Example 4.1.** Consider the following energy game, where it is intended that circular and rectangular states belong to Player 0 and Player 1, respectively.



The optimal strategy for Player 0 is to choose u as the successor to u and v. Thus v requires an initial energy of 8 to keep going forever. For u an initial energy of 0 is sufficient. On the other hand, the optimal strategy for Player 1 is to choose y as successor to x and v as successor to y. This results in a required initial energy of 17 for y and 18 for x.

Thus, we obtain as least fixpoint $g_\Gamma(x) = 18$, $g_\Gamma(y) = 17$, $g_\Gamma(u) = 0$, $g_\Gamma(v) = 8$. Note that, if from u Player 0 would choose x, Player 1 could keep the game in a negative cycle.

Given only the least fixpoint g_Γ, the strategy of Player 1 is not deducible with a local reasoning, since from y the choices x, v are indistinguishable (in fact $g_\Gamma(y) = 17 = g_\Gamma(x) - 1 = g_\Gamma(v) - (-9)$). However, if x is chosen as successor to y (and still y as successor to x), we end up in a value vector where Min needs 0 initial energy in y to keep going.

4.2 Strategy iteration for energy games

In order to solve energy games in our framework, we have to consider non-expansive functions over MV-algebras, however $\mathbb{N}^\infty$ is unfortunately not an MV-algebra. For this, we use the results of [16], where it is shown that any energy game $\Gamma = (V_0, V_1, E, w)$ can be transformed into an energy game $\Gamma' = (V_0', V_1', E', w')$ with finite values only. Concretely, this is done by adding an "emergency exit" for each state in V_0 guaranteeing a finite amount of required energy to keep the game going forever. The solution $g_{\Gamma'}$ of Γ' satisfies $g_{\Gamma'}(v) < \infty$ for all $v \in V$ and the solution g_Γ of Γ can be easily reconstructed from $g_{\Gamma'}$. This allows us to restrict to energy games with finite values, where the solution is bounded by a suitable k. In this setting, letting $K = \{0, \dots, k\}$ and $\ominus_\mathbb{Z} \colon K \times \mathbb{Z} \to K$ given by $x \ominus_\mathbb{Z} y = \min\{\max\{x - y, 0\}, k\}$, we can define $\mathcal{E} \colon K^V \to K^V$ for $a \colon V \to K$ and $v \in V$ as

$$\mathcal{E}(a)(v) = \begin{cases} \min\limits_{(v,v') \in E} a(v') \ominus_\mathbb{Z} w(v, v') & \text{if } v \in V_0 \\ \max\limits_{(v,v') \in E} a(v') \ominus_\mathbb{Z} w(v, v') & \text{if } v \in V_1 \end{cases}$$

▶ **Lemma 4.2** (solution is least fixpoint of $\mathcal{E}$). *Let Γ be an energy game with finite values, bounded by k. Then $\mu\mathcal{E} = g_\Gamma$, i.e. the least fixpoint of $\mathcal{E}$ coincides with the solution of Γ.*

Recall from Example 2.1 that K is an MV-chain. Moreover $\mathcal{E} : K^V \to K^V$ can be proved to be non-expansive by showing that it can be expressed in terms of basic functions which are known or easily shown to be non-expansive and exploiting the fact that non-expansiveness is preserved by composition (see the full version [7]) and thus both generalised strategy iteration approaches in §3, from below and from above, can be applied for determining $\mu\mathcal{E}$, i.e., the solution of Γ.

Observe that the algorithms do not only compute $\mu\mathcal{E}$, but also provide an optimal strategy, for Player 0 when approaching from above and for Player 1 when approaching from below. The second case is of particular interest as it derives an optimal strategy for Player 1, which is often not treated in the literature (we are only aware of [10]).

We also remark that, when performing iteration from above or below, at each iteration, once a strategy C for Player 0 is fixed, we need to compute $\mu\mathcal{E}_C$. This can be done via linear programming, however it turns out that it is more efficient to use some form of value iteration, due to finiteness of the MV-algebra $\{0, \dots, k\}$.

The full version [7] spells out the approximation $\mathcal{E}^a_\#$ of the function $\mathcal{E}$ which – according to the theory in §2 – can be used for checking whether its least fixpoint has already been reached in strategy iteration from above. It also analyses known algorithms for solving energy games and compares their runtime to both kinds of strategy iteration. While other algorithms might in some cases have better runtimes, strategy iteration has the advantage of providing the optimal strategy.

5 Application: behavioural metrics for probabilistic automata

In this section we show how our technique can be used to compute behavioural distances over probabilistic automata. After introducing the necessary notions, we provide a min-decomposition of the corresponding function. The algorithm that we obtain by instantiating our generalised strategy iteration from above using such min-decomposition can be seen to be essentially the same as the one presented in [3].

▶ **Definition 5.1** (probabilistic automaton). *A* probabilistic automaton *(PA) is a tuple* $\mathcal{A} = (S, L, \delta, \ell)$ *consisting of a nonempty finite set* S *of states, a finite set of labels* L*, a successor function* $\delta : S \to \mathcal{P}_{fin}(\mathcal{D}(S))$ *and a labeling function* $\ell \colon S \to L$*.*

The idea is that from a state s, one can non-deterministically move to one of the probability distributions in $\delta(s)$.

The behavioural distance function is defined by combining Hausdorff and Kantorovich liftings for the nondeterministic and probabilistic parts, respectively. Recall that the *Kantorovich lifting* [30] $K : [0,1]^{Y\times Y} \to [0,1]^{\mathcal{D}(Y)\times\mathcal{D}(Y)}$ transforming a pseudometric d on Y to a pseudometric on $\mathcal{D}(Y)$ is defined, for $\beta, \beta' \in \mathcal{D}(Y)$, by

$$K(d)(\beta,\beta') = \min_{\omega\in\Omega(\beta,\beta')} \sum_{y,y'\in Y} d(y,y')\cdot\omega(y,y'),$$

where $\Omega(\beta,\beta')$ is the set of probabilistic couplings of β, β':

$$\Omega(\beta,\beta') = \{\omega \in \mathcal{D}(Y\times Y) \mid \forall y, y' \in Y \colon \sum_{x'\in Y} \omega(y,x') = \beta(y) \wedge \sum_{x\in Y} \omega(x,y') = \beta'(y')\}$$

Actually, the minimum is reached in one of the finitely many vertices of the polytope $\Omega(\beta,\beta')$, a set which we denote by $\Omega_V(\beta,\beta')$. The *Hausdorff lifting* $H : [0,1]^{Y\times Y} \to [0,1]^{\mathcal{P}(Y)\times\mathcal{P}(Y)}$ (in the variant of [26]) is defined, for $X, X' \in \mathcal{P}(Y)$, by

$$H(d)(X,X') = \min_{R\in\mathcal{R}(X,X')} \max_{(x,x')\in R} d(x,x'),$$

with $\mathcal{R}(X,X') = \{R \in \mathcal{P}(Y\times Y) \mid \pi_1(R) = X \wedge \pi_2(R) = X'\}$ the set-couplings of X, X' [26].

The rough idea is the following. If two states s and t have different labels they are at distance 1. Otherwise, in order to compute their distance one has find a "best match" between the outgoing transitions of such states, i.e., a set coupling as those considered in the Hausdorff lifting H. In turn, since, transitions are probabilistic, matching transitions means finding an optimal probabilistic coupling, as done by the Kantorovich lifting K, which is intuitively the best transport plan balancing the "supply" β and the "demand" β'. In this way the distance of s and t is expressed in terms of the distance of the states they can reach, hence, formally, behavioural distance is characterised as a least fixpoint.

▶ **Definition 5.2** (behavioural distance). *Let* $\mathcal{A} = (S, L, \delta, \ell)$ *be a PA. The behavioural distance on* $\mathcal{A}$ *is the least fixpoint of* $\mathcal{M} : [0,1]^{S\times S} \to [0,1]^{S\times S}$ *defined, for* $d \in [0,1]^{S\times S}$ *and* $s, t \in S$*, by* $\mathcal{M}(d)(s,t) = H(K(d))(\delta(s),\delta(t))$ *if* $\ell(s) = \ell(t)$ *and* $\mathcal{M}(d)(s,t) = 1$*, otherwise.*

▶ **Example 5.3.** Consider the probabilistic automaton in Fig. 5 with state space $Y = \{s,t,u\}$, labels $\ell(s) = \ell(t) = a$ and $\ell(u) = b$ and probability distributions $\beta_1, \beta_2, \beta_1', \beta_2', \beta''$ as indicated. For instance, from state s, there are two possible transitions β_1 which with probability $1/2$ goes to u and with probability $1/2$ stays in s, and β_2 which goes to t with probability 1.

In order to explain how function $\mathcal{M}$, resulting from the combination of Hausdorff and Kantorovich lifting, works, let us consider the pseudometric $d(s,t) = 1/2$, $d(s,u) = d(t,u) = 1$. This is not the least fixpoint, since the distance of states s, t is clearly 0 as the two states exihibit the same behaviour.

We now illustrate how to compute $\mathcal{M}(d)(s,t)$. We obtain $\mathcal{M}(d)(s,u) = \mathcal{M}(d)(t,u) = 1$ and, since $\ell(s) = \ell(t) = a$, we have

$$\mathcal{M}(d)(s,t) = H(K(d))(\delta(s),\delta(t)).$$

where $\delta(s) = \{\beta_1, \beta_2\}$ and $\delta(t) = \{\beta_1', \beta_2'\}$.

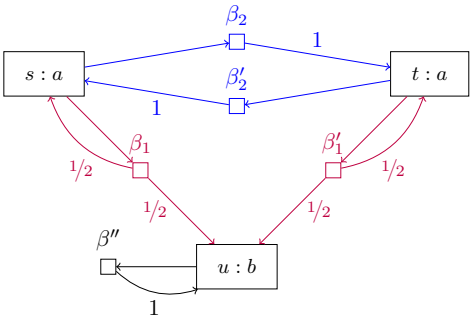


Figure 5 A probabilistic automaton.

It is relatively straightforward to see that the vertices of the coupling polytope $\Omega(\beta_1, \beta_1')$ are $\Omega_V(\beta_1, \beta_1') = \{\omega_1, \omega_2\}$ with

$$\omega_1(s,t) = 1/2,\ \omega_1(u,u) = 1/2 \quad \text{and} \quad \omega_2(s,u) = 1/2,\ \omega_2(u,t) = 1/2$$

and $\omega_i(x,y) = 0$, $i \in \{1,2\}$, for every other pair $(x,y) \in Y \times Y$. Then the Kantorovich lifting is determined as follows:

$$K(d)(\beta_1, \beta_1') = \min\{\sum_{x,y\in S} d(x,y)\cdot\omega_1(x,y), \sum_{x,y\in S} d(x,y)\cdot\omega_2(x,y)\} = \min\{1/4, 1\} = 1/4.$$

Similarly we can obtain $K(d)(\beta_1, \beta_2') = 1/2$, $K(d)(\beta_2, \beta_1') = 1/2$, $K(d)(\beta_2, \beta_2') = 1/4$.

In order to conclude the computation via the Hausdorff lifting, note that the minimal set-couplings of $\delta(s) = \{\beta_1, \beta_2\}$ and $\delta(t) = \{\beta_1', \beta_2'\}$ are

$$R_1 = \{(\beta_1, \beta_1'), (\beta_2, \beta_2')\} \qquad R_2 = \{(\beta_1, \beta_2'), (\beta_2, \beta_1')\}$$

and any other set-coupling includes R_1 or R_2. Then we obtain

$$\begin{aligned}
\mathcal{M}(d)(s,t) &= H(K(d))(\delta(s), \delta(t)) \\
&= \min\{\max_{(x,x')\in R_1} K(d)(x,x'), \max_{(x,x')\in R_2} K(d)(x,x')\} \\
&= \min\{\max\{K(d)(\beta_1,\beta_1'), K(d)(\beta_2,\beta_2')\}, \max\{K(d)(\beta_1,\beta_2'), K(d)(\beta_2,\beta_1')\}\} \\
&= \min\{\max\{1/4, 1/4\}, \max\{1/2, 1/2\}\} = \min\{1/4, 1/2\} = 1/4.
\end{aligned}$$

In order to cast this problem in our framework, we identify a suitable min-decomposition of $\mathcal{M}$. Observe that, for $d \in [0,1]^{S\times S}$ and $s,t \in S$ such that $\ell(s) = \ell(t)$, expanding the definitions of the liftings and taking advantage of complete distributivity, we have

$$\begin{aligned}
\mathcal{M}(d)(s,t) &= \min_{R\in\mathcal{R}(\delta(s),\delta(t))} \max_{(\beta,\beta')\in R} \min_{\omega\in\Omega_V(\beta,\beta')} \sum_{u,v\in S} d(u,v)\cdot\omega(u,v) \\
&= \min_{R\in\mathcal{R}(\delta(s),\delta(t))} \min_{f\in F_R} \max_{(\beta,\beta')\in R} \sum_{u,v\in S} d(u,v)\cdot f(\beta,\beta')(u,v)
\end{aligned}$$

where $F_R = \{f \colon R \to \mathcal{D}(S\times S) \mid f(\beta,\beta') \in \Omega_V(\beta,\beta') \text{ for } (\beta,\beta') \in R\}$, which is a finite set.

We can thus define a min-decomposition $H_{\min}$ for $\mathcal{M}$ (see Definition 3.1) such that $\mathcal{M}(d)(s,t) = \min_{h\in H_{\min}(s,t)} h(d)$ for all $s,t \in S$.

▶ **Definition 5.4** (min-decomposition of $\mathcal{M}$). *Let $\mathcal{A} = (S, L, \delta, \ell)$ be a PA. We denote by $H_{\min}$ the min-decomposition of $\mathcal{M}$ defined as follows. For $s, t \in S$ such that $\ell(s) = \ell(t)$, we let $H_{\min}(s,t) = \{h_{R,f} \mid R \in \mathcal{R}(\delta(s), \delta(t)), f \in F_R\}$, with $h_{R,f} : [0,1]^{S\times S} \to [0,1]$ defined as*

$$h_{R,f}(d) = \max_{(\beta,\beta')\in R} \sum_{u,v\in S} d(u,v) \cdot f(\beta,\beta')(u,v).$$

If instead $\ell(s) \neq \ell(t)$, we let $H_{\min}(s,t) = \{h_1\}$ where $h_1(d) = 1$ for all d.

A strategy C in $H_{\min}$ maps each pair of states $s, t \in S$ to a function in $H_{\min}(s,t)$, that is

- if $\ell(s) \neq \ell(t)$, to the unique element $h_1 \in H_{\min}(s,t)$;
- if $\ell(s) = \ell(t)$ to some $h_{R,f} \in H_{\min}(s,t)$, with $R \in \mathcal{R}(\delta(s), \delta(t))$ set-coupling and $f \in F_R$.

The decomposition above can be used to deduce that $\mathcal{M}$ is non-expansive and thus we can safely instantiate the algorithm in Fig. 2 to compute the least fixpoint from above. The resulting algorithm is quite similar to the one specifically developed for PAs in [3]. In particular, it can be seen that, apart from the different presentation, a strategy C corresponds to what [3] refers to as a *coupling structure*. In addition, the step in item (2c) of the algorithm (see Fig. 2) is analogous to that in [3]. In fact, in order to check whether the fixpoint obtained with the current strategy C_i, i.e. $\mu\mathcal{M}_{C_i}$, is the least fixpoint of $\mathcal{M}$, one considers the approximation $\mathcal{M}_{\#}^{\mu\mathcal{M}_{C_i}}$ and checks whether its greatest fixpoint is empty. Recalling that the post-fixpoints of $\mathcal{M}_{\#}^{\mu\mathcal{M}_{C_i}}$ have been shown in [5] to be the self-closed relations of [3], one derives that verifying the emptiness of the greatest fixpoint of $\mathcal{M}_{\#}^{\mu\mathcal{M}_{C_i}}$ corresponds exactly to checking whether the largest self-closed relation is empty (see [7] for more details).

6 Conclusion

We developed abstract algorithms for strategy iterations which allow to compute least fixpoints (or, dually, greatest fixpoints) of non-expansive functions over MV-algebras. The idea consists in expressing the function of interest as a minimum (or a maximum), and view the process of computing the function as a game between players Min and Max trying to minimise and maximise, respectively, the outcome. Then the algorithms proceed via a sequence of steps which converge to the least fixpoint from above, progressively improving the strategy of player Min, or from below, progressively improving the strategy of the player Max. The two procedures have similar worst-case complexity. The number of iterations is bounded by the number of strategies of the corresponding player $p \in \{\min, \max\}$, which is exponential in the input size (the number of strategies is $\prod_{y\in Y} |H_p(y)|$). This suggests that, depending on the setting, the fastest algorithm is the one using the smaller decomposition $H_{\min}$ respectively $H_{\max}$. However, a deeper analysis is still needed, as a smaller decomposition usually leads to a higher cost for computing μf_C.

The algorithms generalise an approach which has been recently proposed for simple stochastic games in [5, 6]. We showed how our technique instantiates to energy games, thus giving a method for determining the optimal strategies of both players, and to the computation of the behavioural distance for probabilistic automata, resulting in an algorithm similar to the non-trivial procedure in [3], which was also a source of inspiration.

Strategy iteration is used in many different application domains with fairly similar underlying ideas and we believe that it is fruitful to provide a general definition of the technique, clarifying and solving several issues on this level, such as the need for stable improvements or ways to deal with non-unique fixpoints.

There is an extremely wide literature on strategy iteration, often also referred to as policy iteration or strategy improvement (for an overview see [19]). As mentioned in the introduction, after its use on nonterminating stochastic games [23], it has been applied to solve many kinds of games, including discounted mean-payoff games [32], parity games [31, 28] and simple stochastic games [13]. Several quasi-polynomial algorithms have been recently devised for parity games [11, 22, 24], while the existence of a polynomial algorithm is still an important open problem. This has been generalized to finite lattices by [20].

Various papers on strategy iteration focus on lower bounds [18, 17]. Our paper, rather than concentrating on complexity issues, provides a general framework capturing strategy iteration in a general lattice theoretical setting. A work similar in spirit is [1] which proposes a meta-algorithm GSIA such that a number of strategy improvement algorithms for SSGs arise as instances, along with a general complexity bound. Differently from ours, this paper focuses on SSGs and iteration from below. However, it allows for the parametrisation of the algorithm on a subset of edges of interest in the game graph, which is not possible in our approach, and so it can provide interesting suggestions for further generalisations.

Another interesting setting of application is the lower-weak-upper-bound problem in mean-payoff games [9], reminiscent of energy games. For this problem, differently from the usual definition, the aim for one player is to maximise, never going negative, some resource which cannot exceed a given bound, while the other player has to minimise it. Also in this case, the solution can be computed as a least fixpoint. Due to the upper bound imposed to the resource, the function is not non-expansive, thus it is not captured by our theory. Still, the algorithm KASI proposed in [10], which computes the solution via strategy iteration, shares many similarities with our approach from below: at each iteration the algorithm computes a stable max-improvement of the current strategy. Indeed, when applying KASI to the special case where there is no upper bound to the accumulated resource, called lower-bound problem in [9] (also studied under different names in [12, 25]), the algorithm comes out as an exact instantiation of our general strategy iteration from below.

Given their generality, we believe that the algorithms proposed in the present paper have the potential to be applicable to a variety of other settings. In particular, some preliminary investigations show their applicability to computing behavioural metrics in an abstract coalgebraic setting [4]. Here the behavioural distance is naturally characterised as a least fixpoint of an operator based on the Wasserstein lifting of the behavioural functor. Then the idea is to view couplings used in the computation of the Wasserstein lifting as strategies and use strategy iteration for converging to the coalgebraic metric.

Our abstract strategy iteration algorithms rely on the assumption that, once a strategy for one of the players is fixed, the optimal "answering" strategy for the opponent can be computed efficiently. Identifying abstract settings where a min- or max-decompositions of a function ensures that the answering strategy can be indeed computed efficiently (e.g., via linear programming as it happens for simple stochastic games), is an interesting direction of future research.

References

1 David Auger, Xavier Badin de Montjoye, and Yann Strozecki. A generic strategy improvement method for simple stochastic games. In *Proc. of MFCS 2021*, volume 202 of *LIPIcs*, pages 12:1–12:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

2 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. On-the-fly exact computation of bisimilarity distances. *Logical Methods in Computer Science*, 13(2:13):1–25, 2017.

3 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, Radu Mardare, Qiyi Tang, and Franck van Breugel. Computing probabilistic bisimilarity distances for probabilistic automata. *Logical Methods in Computer Science*, 17(1), 2021.

4 Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König. Coalgebraic behavioral metrics. *Logical Methods in Computer Science*, 14(3), 2018. Selected Papers of the 6th Conference on Algebra and Coalgebra in Computer Science (CALCO 2015). URL: `https://lmcs.episciences.org/4827`.

5 Paolo Baldan, Richard Eggert, Barbara König, and Tommaso Padoan. Fixpoint theory – upside down. In *Proc. of FOSSACS 2021*, volume 12650 of *Lecture Notes in Computer Science*, pages 62–81. Springer, 2021.

6 Paolo Baldan, Richard Eggert, Barbara König, and Tommaso Padoan. Fixpoint theory – upside down. *CoRR*, abs/2101.08184, 2021. `arXiv:2101.08184`.

7 Paolo Baldan, Richard Eggert, Barbara König, and Tommaso Padoan. A lattice-theoretical view of strategy iteration, 2022. `arXiv:2207.09872`.

8 Filippo Bonchi, Barbara König, and Daniela Petrişan. Up-to techniques for behavioural metrics via fibrations. In *Proc. of CONCUR 2018*, volume 118 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl – Leibniz Center for Informatics, 2018. `doi:10.4230/LIPIcs.CONCUR.2018.17`.

9 Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *Formal Modeling and Analysis of Timed Systems*, pages 33–47. Springer, 2008.

10 Luboš Brim and Jakub Chaloupka. Using strategy improvement to stay alive. *Electronic Proceedings in Theoretical Computer Science*, 25:40–54, 2010. `doi:10.4204/eptcs.25.8`.

11 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proc. of STOC 2017*, pages 252–263, 2017.

12 Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *Prof. of EMSOFT '03 (International Workshop on Embedded Software)*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003.

13 Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992. `doi:10.1016/0890-5401(92)90048-K`.

14 Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL 1977*, pages 238–252. ACM, 1977.

15 Patrick Cousot and Radhia Cousot. Temporal abstract interpretation. In Mark N. Wegman and Thomas W. Reps, editors, *Proc. of POPL 2000*, pages 12–25. ACM, 2000.

16 Dani Dorfman, Haim Kaplan, and Uri Zwick. A faster deterministic exponential time algorithm for energy games and mean payoff games. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *Proc. of ICALP 2019*, volume 132 of *LIPIcs*, pages 114:1–114:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ICALP.2019.114`.

17 John Fearnley. Exponential lower bounds for policy iteration. In *Proc. of ICALP 2010*, volume 6199 of *Lecture Notes in Computer Science*, pages 551–562. Springer, 2010.

18 Oliver Friedmann. An exponential lower bound for the latest deterministic strategy iteration algorithms. *Log. Methods Comput. Sci.*, 7(3), 2011.

19 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

20 Daniel Hausmann and Lutz Schröder. Quasipolynomial computation of nested fixpoints. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Proc. of TACAS 2021*, volume 12651 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2021.

21 Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

22 Marcin Jurdzinski and Ranko Lazic. Succinct progress measures for solving parity games. In *Proc. of LICS 2017*, pages 1–9. ACM/IEEE, 2017.

23 Richard M. Karp and Alan J. Hoffman. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

24 Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *Proc. of LICS 2018*, pages 639–648. ACM/IEEE, 2018.

25 Yu. M. Lifshits and D. S. Pavlov. Potential theory for mean payoff games. *Zapiski Nauchnykh Seminarov POMI*, 340:61–75, 2006.

26 Facundo Mémoli. Gromov-Wasserstein distances and the metric approach to object matching. *Foundations of Computational Mathematics*, 11(4):417–487, 2011.

27 Daniele Mundici. MV-algebras. A short tutorial. Available at `http://www.matematica.uns.edu.ar/IXCongresoMonteiro/Comunicaciones/Mundici_tutorial.pdf`.

28 Sven Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Proc. of CSL 2008*, volume 5213 of *Lecture Notes in Computer Science*, pages 369–384. Springer, 2008.

29 Alfred Tarski. A lattice-theoretical theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

30 Cédric Villani. *Optimal Transport – Old and New*, volume 338 of *A Series of Comprehensive Studies in Mathematics*. Springer, 2009.

31 Jens Vöge and Marcin Jurdzinski. A discrete strategy improvement algorithm for solving parity games. In *Proc. of CAV 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215. Springer, 2000.

32 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1&2):343–359, 1996.

Reductions in Higher-Order Rewriting and Their Equivalence

Pablo Barenbaum
National University of Quilmes (CONICET), Bernal, Argentina
University of Buenos Aires, Argentina

Eduardo Bonelli
Stevens Institute of Technology, Hoboken, NJ, USA

Abstract

Proof terms are syntactic expressions that represent computations in term rewriting. They were introduced by Meseguer and exploited by van Oostrom and de Vrijer to study *equivalence of reductions* in (left-linear) first-order term rewriting systems. We study the problem of extending the notion of proof term to *higher-order rewriting*, which generalizes the first-order setting by allowing terms with binders and higher-order substitution. In previous works that devise proof terms for higher-order rewriting, such as Bruggink's, it has been noted that the challenge lies in reconciling composition of proof terms and higher-order substitution (β-equivalence). This led Bruggink to reject "nested" composition, other than at the outermost level. In this paper, we propose a notion of higher-order proof term we dub *rewrites* that supports nested composition. We then define *two* notions of equivalence on rewrites, namely *permutation equivalence* and *projection equivalence*, and show that they coincide.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Type theory

Keywords and phrases Term Rewriting, Higher-Order Rewriting, Proof terms, Equivalence of Computations

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.8

Related Version *Extended Version*: `https://arxiv.org/abs/2210.15654` [2]

Funding *Pablo Barenbaum*: Partially supported by project ECOS-Sud A17C01.

1 Introduction

Term rewriting systems model computation as sequences of steps between terms, *reduction sequences*, where steps are instances of term rewriting rules [15]. It is natural to consider reduction sequences up to swapping of orthogonal steps since such reductions perform the "same work". The ensuing notion of equivalence is called *permutation equivalence* and was first studied by Lévy [11] in the setting of the λ-calculus but has appeared in other guises connected with concurrency [15, Rem.8.1.1]. As an example, consider the rewrite rule $\mathbf{c}(x, \mathbf{f}(y)) \to \mathbf{d}(x, x)$ and the following reduction sequence where, in each step, the contracted redex is underlined:

$$\underline{\mathbf{c}(\mathbf{c}(z, \mathbf{f}(z)), \mathbf{f}(z))} \to \mathbf{d}(\underline{\mathbf{c}(z, \mathbf{f}(z))}, \mathbf{c}(z, \mathbf{f}(z))) \to \mathbf{d}(\mathbf{d}(z, z), \underline{\mathbf{c}(z, \mathbf{f}(z))}) \to \mathbf{d}(\mathbf{d}(z, z), \mathbf{d}(z, z)) \tag{1}$$

Performing the innermost redex first, rather than the outermost one, leads to:

$$\mathbf{c}(\underline{\mathbf{c}(z, \mathbf{f}(z))}, \mathbf{f}(z)) \to \underline{\mathbf{c}(\mathbf{d}(z, z), \mathbf{f}(z))} \to \mathbf{d}(\mathbf{d}(z, z), \mathbf{d}(z, z)) \tag{2}$$

The first step in (1) makes two copies of the innermost redex. It is the two steps contracting these copies that are swapped with the first one in (1) to produce (2). Such duplication (and erasure) contribute most of the complications behind permutation equivalence, both in its formulation and the study of its properties.

Proof Terms. *Proof terms* are a natural representation for computations. They were introduced by Meseguer as a means of representing proofs in Rewriting Logic [13] and exploited by van Oostrom and de Vrijer in the setting of first-order left-linear rewriting systems, to study equivalence of reductions in [17] and [15, Chapter 9]. Rewrite rules are assigned *rule symbols* denoting the application of a rewriting rule. Proof terms are expressions built using function symbols, a binary operator "**;**" denoting sequential composition of proof terms, and rule symbols. Assuming the following rule symbol for our rewrite rule $\varrho(x,y) : \mathbf{c}(x,\mathbf{f}(y)) \twoheadrightarrow \mathbf{d}(x,x)$, reduction (1) may be represented as the proof term: $\varrho(\mathbf{c}(z,\mathbf{f}(z)),z)$ **;** $\mathbf{d}(\varrho(z,z),\mathbf{c}(z,\mathbf{f}(z)))$ **;** $\mathbf{d}(\mathbf{d}(z,z),\varrho(z,z))$ and reduction (2) as the proof term: $\mathbf{c}(\varrho(z,z),\mathbf{f}(z))$ **;** $\varrho(\mathbf{d}(z,z),z)$. One notable feature of proof terms is that they support parallel steps. For instance, both proof terms above are permutation equivalent to $\varrho(\mathbf{c}(z,\mathbf{f}(z)),z)$ **;** $\mathbf{d}(\varrho(z,z),\varrho(z,z))$, which performs the two last steps in parallel, as well as to $\varrho(\varrho(z,z),z)$, which performs all steps simultaneously. Permutation equivalence now can be studied in terms of equational theories on proof terms.

Equivalence of Reductions via Proof Terms for First-Order Rewriting. In [17], van Oostrom and de Vrijer characterize permutation equivalence of proof terms in four alternative ways. First, they formulate an equational theory of permutation equivalence $\rho \approx \sigma$ between proof terms, such that for example $\varrho(\mathbf{c}(z,\mathbf{f}(z)),z)$ **;** $\mathbf{d}(\varrho(z,z),\varrho(z,z)) \approx \varrho(\varrho(z,z),z)$ holds. These equations account for the behavior of proof term composition, which has a monoidal structure, in the sense that composition is associative and *empty* steps act as identities. Second, they define an operation of *projection* ρ/σ, denoting the computational work that is left of ρ after σ. For example, $\mathbf{c}(\varrho(z,z),\mathbf{f}(z))/\varrho(\mathbf{c}(z,\mathbf{f}(z)),z) = \mathbf{d}(\varrho(z,z),\varrho(z,z))$. This induces a notion of *projection equivalence* between proof terms ρ and σ, declared to hold when both ρ/σ and σ/ρ are empty, *i.e.* they contain no rule symbols. Third, they define a *standardization procedure* to reorder the steps of a reduction in outside-in order, mapping each proof term ρ to a proof term ρ^* in *standard form*. For example, the (parallel) standard form of $\mathbf{c}(\varrho(z,z),\mathbf{f}(z))$ **;** $\varrho(\mathbf{d}(z,z),z)$ is $\varrho(\mathbf{c}(z,\mathbf{f}(z)),z)$ **;** $\mathbf{d}(\varrho(z,z),\varrho(z,z))$. This induces a notion of *standardization equivalence* between proof terms ρ and σ, declared to hold when $\rho^* = \sigma^*$. Fourth, they define a notion of *labelling equivalence*, based on lifting computational steps to labelled terms. Although these notions of equivalence were known prior to [17], the main result of that paper is that they are systematically studied using proof terms and, moreover, shown to coincide.

Higher-Order Rewriting. Higher-order term rewriting (HOR) generalizes first-order term rewriting by allowing binders. Function symbols are generalized to constants of any given simple type, and first-order terms are generalized to simply-typed λ-terms, including constants and up to $\beta\eta$-equivalence. The paradigmatic example of a higher-order rewriting system is the λ-calculus. It includes a base type ι and two constants $\mathbf{app} : \iota \to \iota \to \iota$ and $\mathbf{lam} : (\iota \to \iota) \to \iota$; β-reduction may be expressed as the higher-order rewrite rule $\mathbf{app}\,(\mathbf{lam}\,(\lambda z.x\,z))\,y \twoheadrightarrow x\,y$. A sample reduction sequence is:

$$\mathbf{lam}(\lambda v.\mathbf{app}(\mathbf{lam}(\lambda x.x),\underline{\mathbf{app}(\mathbf{lam}(\lambda w.w),v)})) \twoheadrightarrow \mathbf{lam}(\lambda v.\underline{\mathbf{app}(\mathbf{lam}(\lambda x.x),v)}) \twoheadrightarrow \mathbf{lam}(\lambda v.v) \quad (3)$$

Generalizing proof terms to the setting of higher-order rewriting is a natural goal. Just like in the first-order case, we assign rule symbols to rewrite rules. One would then expect to obtain proof terms by adding these rule symbols and the ";" composition operator to the simply typed λ-calculus. If we assume the following rule symbol for our rewrite rule $\varrho\, x\, y : \mathbf{app}\,(\mathbf{lam}\,(\lambda z.x\, z))\, y \twoheadrightarrow x\, y$, then an example of a higher-order proof term for (3) is:

$$\mathbf{lam}\Big(\lambda v.\big(\mathbf{app}(\mathbf{lam}(\lambda x.x), \varrho\,(\lambda w.w)\, v)\ ;\ \varrho\,(\lambda u.u)\, v\big)\Big)$$

However, higher-order substitution and proof term composition seem not to be in consonance, an issue already observed by Bruggink [4]. Consider a variable x. This variable itself denotes an empty computation $x \twoheadrightarrow x$, so the composition $(x\ ;\ x)$ also denotes an empty computation $x \twoheadrightarrow x$. If σ is an arbitrary proof term $s \twoheadrightarrow t$, the proof term $(\lambda x.(x\ ;\ x))\,\sigma$ should, in principle, represent a computation $(\lambda x.x)\, s \twoheadrightarrow (\lambda x.x)\, t$. This is the same as $s \twoheadrightarrow t$, because terms are regarded up to $\beta\eta$-equivalence. The challenge lies in lifting $\beta\eta$-equivalence to the level of proof terms: if β-reduction is naively extended to operate on proof terms, the well-formed proof term $(\lambda x.(x\ ;\ x))\,\sigma$ becomes equal to $(\sigma\ ;\ \sigma)$, which is ill-formed because σ is not composable with itself if $s \neq_{\beta\eta} t$. Rather than simply disallowing the use of ";" under applications and abstractions (the route taken in [4]), our aim is to integrate it with $\beta\eta$-reduction.

Contribution. We propose a **syntax for higher-order proof terms**, called **rewrites**, that includes $\beta\eta$-equivalence and allows rewrites to be freely composed. We then define a relation $\rho \approx \sigma$ of **permutation equivalence** between rewrites, the central notion of our work. The issue mentioned above is avoided by *disallowing* the ill-behaved substitution of a rewrite in a rewrite "$\rho\{x\backslash\sigma\}$", and by only allowing notions of substitution of a term in a rewrite $\rho\{x\backslash s\}$, and of a rewrite in a term $s\{x\backslash\!\backslash\rho\}$. From these, a well-behaved notion of substitution of a rewrite in a rewrite $\rho\{x\backslash\!\backslash\!\backslash\sigma\}$ can be shown to be *derivable*. We also define a notion of **projection** $\rho /\!\!/ \sigma$. The induced notion of **projection equivalence coincides with permutation equivalence**, in the sense that $\rho \approx \sigma$ iff $\rho /\!\!/ \sigma \approx \sigma^{\mathsf{tgt}}$ and $\sigma /\!\!/ \rho \approx \rho^{\mathsf{tgt}}$, where ρ^{tgt} stands for the *target* term of ρ. The equivalence is established by means of **flattening**, a method to convert an arbitrary rewrite ρ into a (*flat*) representative $\rho^\flat$ that only uses the composition operator "**;**" at the top level and a notion of **flat permutation equivalence** $\rho \sim \sigma$. Flattening is achieved by means of a rewriting system whose objects are themselves rewrites. This system is shown to be confluent and strongly normalizing. We also show that **permutation equivalence is sound and complete with respect to flat permutation equivalence** in the sense that $\rho \approx \sigma$ if and only if $\rho^\flat \sim \sigma^\flat$.

Structure of the Paper. In Section 2 we review Nipkow's Higher-Order Rewriting Systems. Section 3 proposes our notion of rewrite and Section 4 introduces permutation equivalence for them. Flattening is presented in Section 5. In this section, we also formulate an equational theory defining the relation $\rho \sim \sigma$ of flat permutation equivalence between flat rewrites. It relies crucially on a ternary relation between *multisteps*, called *splitting* and written $\mu \Leftrightarrow \mu_1\ ;\ \mu_2$, meaning that μ and $\mu_1\ ;\ \mu_2$ perform the same computational work. In Section 6 we first define a projection operator for flat rewrites ρ/σ, and we lift it to a projection operator for arbitrary rewrites $\rho /\!\!/ \sigma \stackrel{\text{def}}{=} \rho^\flat/\sigma^\flat$. Then we show that the induced notion of projection equivalence coincides with permutation equivalence. Finally, we conclude and discuss related and future work. Detailed proofs can be found in the accompanying technical report [2].

2 Higher-Order Rewriting

There are various approaches to HOR in the literature, including Klop's Combinatory Reduction Systems (CRSs) [8] and Nipkow's Higher-Order Rewriting Systems (HRSs) [14, 12]. We consider HRSs in this paper. Their use of the simply-typed lambda calculus for representing terms and substitution provides a suitable starting point for modeling our rewrites. Moreover, HRS are arguably more general than CRS in that their instantiation mechanism is more powerful [15, Sec.11.4.2]. We next introduce HRS. Assume given a denumerably infinite set of *variables* $(x, y, \ldots)$, *base types* $(\alpha, \beta, \ldots)$, and *constant symbols* $(\mathbf{c}, \mathbf{d}, \ldots)$. The sets of *terms* $(s, t, \ldots)$ and *types* $(A, B, \ldots)$ are given by:

$$s \quad ::= \quad x \mid \mathbf{c} \mid \lambda x.s \mid s\,s \qquad\qquad A \quad ::= \quad \alpha \mid A \to A$$

A term can either be a variable, a constant, an abstraction or an application. A type can either be a base type or an arrow type. We write $\mathsf{fv}(s)$ for the free variables of s. We use $\overline{X_n}$, or sometimes just $\overline{X}$ if n is clear from the context, to denote a sequence $X_1, \ldots, X_n$. Following standard conventions, $s\,\overline{t_n}$ stands for the iterated application $s\,t_1 \ldots t_n$, and $\overline{A_n} \to B$ for the type $A_1 \to \ldots A_n \to B$. We write $s\{x \backslash t\}$ for the capture-avoiding substitution of all free occurrences of x in s with t and call it a *term/term substitution*. We identify terms that differ only in the names of their bound variables. A *typing context* $(\Gamma, \Gamma', \ldots)$ is a partial function from variables to types. We write $\mathsf{dom}(\Gamma)$ for the *domain* of Γ. Given a typing context Γ and $x \notin \mathsf{dom}(\Gamma)$, we write $\Gamma, x : A$ for the typing context such that $(\Gamma, x : A)(x) = A$, and $(\Gamma, x : A)(y) = \Gamma(y)$ whenever $y \neq x$. We write $\cdot$ for the empty typing context and $x \in \Gamma$ if $x \in \mathsf{dom}(\Gamma)$. A *signature* of a HRS is a set $\mathcal{C}$ of typed constants $\mathbf{c} : A$. A sample signature is $\mathcal{C} = \{\mathbf{app} : \iota \to \iota \to \iota, \mathbf{lam} : (\iota \to \iota) \to \iota\}$ for ι a base type.

▶ **Definition 1** (Type system for terms). *Terms are typed using the usual typing rules of the simply-typed λ-calculus:*

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}\,\mathsf{Var} \qquad \frac{(\mathbf{c} : A) \in \mathcal{C}}{\Gamma \vdash \mathbf{c} : A}\,\mathsf{Con} \qquad \frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x.s : A \to B}\,\mathsf{Abs} \qquad \frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash s\,t : B}\,\mathsf{App}$$

Given any Γ and A such that $\Gamma \vdash s : A$ can be proved using these rules, we say s is a typed term *over $\mathcal{C}$. We typically drop $\mathcal{C}$ assuming it is implicit.*

We assume the usual definition of β and η-reduction between terms. Recall that β-reduction (resp. η-reduction) is confluent and terminating on typed terms. We write $s \downarrow^{\beta}$ (resp. $s \downarrow^{\eta}$) for the unique β-normal form (resp. η-normal form) of s. The β-normal form of a term s has the form $\lambda\overline{x_k}.a\,t_1 \ldots t_m$, for a either a constant or a variable. The η-expanded form of s is defined as:

$$s \uparrow^{\eta} \quad \stackrel{\text{def}}{=} \quad \lambda\overline{x_{n+k}}.a\,(\overline{t_m} \uparrow^{\eta})\,(x_{n+1} \uparrow^{\eta}) \ldots (x_{n+k} \uparrow^{\eta})$$

where s is assumed to have type $\overline{A_{n+k}} \to B$ and the $x_{n+1}, \ldots, x_{n+k}$ are fresh. We use $s \updownarrow^{\eta}_{\beta}$ to denote the term $s \downarrow^{\beta} \uparrow^{\eta}$ and call it the $\beta\overline{\eta}$-normal form of s.

A *substitution* θ is a function from variables to typed terms such that $\theta(x) \neq x$ only for finitely many x. The *domain* of a substitution is defined as $\mathsf{dom}(\theta) = \{x \mid \theta(x) \neq x\}$. The application of a substitution $\theta = \{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$ to a term t is defined as $\theta\,t \stackrel{\text{def}}{=} ((\lambda\overline{x_n}.t)\,\overline{s_n}) \updownarrow^{\eta}_{\beta}$.

▶ **Definition 2.** *A* pattern *is a typed term in β-normal form such that all free occurrences of a variable x_i are in a subterm of the form $x_i\, t_1 \dots t_k$ with $t_1, \dots, t_k$ η-equivalent to distinct bound variables. A* rewriting rule *is a pair $\langle \ell, r \rangle$ of typed terms in $\beta\overline{\eta}$-normal form of the same base type with ℓ a pattern not η-equivalent to a variable and $\mathsf{fv}(r) \subseteq \mathsf{fv}(\ell)$. An HRS is a pair consisting of a signature and a set of rewriting rules over that signature. We typically omit the signature.*

▶ **Definition 3.** *The rewrite relation $\to_{\mathcal{R}}$ for an HRS $\mathcal{R}$ is the relation over typed terms in $\beta\overline{\eta}$-normal form defined as follows:*

$$\frac{\langle \ell, r \rangle \in \mathcal{R}}{\theta\,\ell \;\to_{\mathcal{R}}\; \theta\, r}\,\mathsf{Root} \qquad \frac{s \;\to_{\mathcal{R}}\; t}{a\,\overline{r_m}\,s\,\overline{p_n} \;\to_{\mathcal{R}}\; a\,\overline{r_m}\,t\,\overline{p_n}}\,\mathsf{App} \qquad \frac{s \;\to_{\mathcal{R}}\; t}{\lambda x.s \;\to_{\mathcal{R}}\; \lambda x.t}\,\mathsf{Abs}$$

where a is either a constant or a variable of type $\overline{A_{m+1+n}} \to B$. We write $\to_{\mathcal{R}}^*$ (resp. $\leftrightarrow_{\mathcal{R}}^*$) for the reflexive, transitive (resp. reflexive, symmetric and transtive) closure of $\to_{\mathcal{R}}$.

▶ **Example 4.** Consider a base type ι and typed constants $\mathbf{mu} : (\iota \to \iota) \to \iota$ and $\mathbf{f} : \iota \to \iota$. Two sample rewriting rules are: $\langle \mathbf{mu}(\lambda y.x\, y), x\,(\mathbf{mu}(\lambda y.x\, y)) \rangle$ and $\langle \mathbf{f}\, x, \mathbf{g}\, x \rangle$. All four terms have base type ι. An example of a sequence of rewrite steps is $\mathbf{mu}\,(\lambda x.\mathbf{f}\, x) \;\to_{\mathcal{R}}\; \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\mathbf{f}\, x)) \;\to_{\mathcal{R}}\; \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\, x)) \;\to_{\mathcal{R}}\; \mathbf{g}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\, x))$.

An HRS is *orthogonal* if: **1.** The rules are *left-linear*, *i.e.* if the left-hand side ℓ has $\mathsf{fv}(\ell) = \{x_1, \dots, x_n\}$, then there is *exactly* one free occurrence of x_i in ℓ, for each $1 \le i \le n$. **2.** There are *no critical pairs*, as defined for example in [14, Def. 4.1]. Orthogonal HRSs are deterministic in the sense that their rewrite relation is confluent. All of the examples of HRSs presented above are orthogonal. In the sequel of this paper, we assume given a fixed, orthogonal HRS $\mathcal{R}$.

3 Rewrites

In this section we propose a syntax for higher-order proof terms, called **rewrites**[1]. Rewrites for an HRS $\mathcal{R}$ are a means for denoting proofs in Higher-Order Rewriting Logic (HORL, *cf.* Def. 7) which, in turn, correspond to reduction sequences in $\mathcal{R}$ (*cf.* Thm. 9). As in the first-order case [13], HORL is simply the equational theory that results from an HRS but disregarding symmetry. Given an HRS $\mathcal{R}$, let $\mathcal{R}^c$ denote the set of pairs $\langle \lambda\overline{x_n}.\ell, \lambda\overline{x_n}.r \rangle$ such that $\langle \ell, r \rangle \in \mathcal{R}$ and $\{x_1, \dots, x_n\} = \mathsf{fv}(\ell)$. We begin by recalling the definition of equational logic (*cf.* Def. 5), the equational theory induced by an HRS. It is essentially that of [12, Def. 3.11], except that in the inference rule ERule we use $\mathcal{R}^c$ rather than $\mathcal{R}$. This equivalent formulation will be convenient when introducing rewrites since free variables in the LHS of a rewrite rule will be reflected in the rewrite too.

[1] Our notion of rewrite is unrelated to that of Def. 2.4 in [13]; it corresponds to "proof terms" as introduced in Sec. 3.1 in [13].

▶ **Definition 5** (Equational Logic). *An HRS $\mathcal{R}$ induces a relation $\doteq_{\mathcal{R}}$ on terms defined by the following rules:*

$$\frac{\Gamma, x : A \vdash s : B \qquad \Gamma \vdash t : A}{\Gamma \vdash (\lambda x.s)\, t \doteq_{\mathcal{R}} s\{x \backslash t\} : B}\,\mathsf{EBeta} \qquad \frac{\Gamma, x : A \vdash s : B \qquad x \notin \mathsf{fv}(s)}{\Gamma \vdash \lambda x.s\, x \doteq_{\mathcal{R}} s : B}\,\mathsf{EEta}$$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x \doteq_{\mathcal{R}} x : A}\,\mathsf{EVar} \qquad \frac{(\mathbf{c} : A) \in \mathcal{C}}{\Gamma \vdash \mathbf{c} \doteq_{\mathcal{R}} \mathbf{c} : A}\,\mathsf{ECon} \qquad \frac{\Gamma, x : A \vdash s_0 \doteq_{\mathcal{R}} s_1 : B}{\Gamma \vdash \lambda x.s_0 \doteq_{\mathcal{R}} \lambda x.s_1 : A \to B}\,\mathsf{EAbs}$$

$$\frac{\Gamma \vdash s_0 \doteq_{\mathcal{R}} s_1 : A \to B \qquad \Gamma \vdash t_0 \doteq_{\mathcal{R}} t_1 : A}{\Gamma \vdash s_0\, t_0 \doteq_{\mathcal{R}} s_1\, t_1 : B}\,\mathsf{EApp} \qquad \frac{\langle s, t\rangle \in \mathcal{R}^c \qquad \cdot \vdash s : A \qquad \cdot \vdash t : A}{\Gamma \vdash s \doteq_{\mathcal{R}} t : A}\,\mathsf{ERule}$$

$$\frac{\Gamma \vdash s_0 \doteq_{\mathcal{R}} s_1 : A}{\Gamma \vdash s_1 \doteq_{\mathcal{R}} s_0 : A}\,\mathsf{ESymm} \qquad \frac{\Gamma \vdash s_0 \doteq_{\mathcal{R}} s_1 : A \qquad \Gamma \vdash s_1 \doteq_{\mathcal{R}} s_2 : A}{\Gamma \vdash s_0 \doteq_{\mathcal{R}} s_2 : A}\,\mathsf{ETrans}$$

▶ **Theorem 6** (Thm. 3.12 in [12]). $\Gamma \vdash s \doteq_{\mathcal{R}} t : A$ *iff* $s \updownarrow_{\beta}^{\eta} \leftrightarrow_{\mathcal{R}}^{*} t \updownarrow_{\beta}^{\eta}$.

The ($\Leftarrow$) direction follows from observing that $\to_{\beta,\overline{\eta}}$ and $\leftrightarrow_{\mathcal{R}}^{*}$ are all included in $\doteq_{\mathcal{R}}$. The ($\Rightarrow$) direction is by induction on the derivation of $\Gamma \vdash s \doteq_{\mathcal{R}} t : A$.

Higher-Order Rewriting Logic results from dropping ESymm in Def. 5 and adding a proof witness. Its judgments take the form $\Gamma \vdash \rho : s \twoheadrightarrow t : A$ where the proof witness ρ is called a *rewrite*. Given a set of *rule symbols* $(\varrho, \vartheta, \ldots)$, the set of *rewrites* $(\rho, \sigma, \ldots)$ is given by:

$$\rho ::= x \mid \mathbf{c} \mid \varrho \mid \lambda x.\rho \mid \rho\,\rho \mid \rho \,;\, \rho$$

A rewrite can either be a variable, a constant, a rule symbol, an abstraction congruence, an application congruence, or a composition. Note that composition may occur anywhere inside a rewrite. For the sake of clarity we present the full system for Higher-Order Rewriting Logic next. We assume given an HRS $\mathcal{R}$ such that each rewrite rule $\langle \ell, r\rangle \in \mathcal{R}$ has been assigned a unique rule symbol ϱ and shall write $\langle \varrho, \ell, r\rangle \in \mathcal{R}$ and also use the same notation for $\mathcal{R}^c$. HORL consists of two forms of typing judgments:

1. $\Gamma \vdash s =_{\beta\eta} t : A$, meaning that s and t are $\beta\eta$-equivalent terms of type A under Γ; and
2. $\Gamma \vdash \rho : s \twoheadrightarrow_{\mathcal{R}} t : A$, meaning that ρ is a rewrite with source s and target t, which are terms of type A under Γ.

▶ **Definition 7** (Higher-Order Rewriting Logic). *Term equivalence is defined as the reflexive, symmetric, transitive, and contextual closure of:*

$$\frac{\Gamma, x : A \vdash s : B \qquad \Gamma \vdash t : A}{\Gamma \vdash (\lambda x.s)\, t =_{\beta\eta} s\{x \backslash t\} : B}\,\mathsf{EqBeta} \qquad \frac{\Gamma, x : A \vdash s : B \qquad x \notin \mathsf{fv}(s)}{\Gamma \vdash \lambda x.s\, x =_{\beta\eta} s : B}\,\mathsf{EqEta}$$

Typing rules for rewrites are as follows:

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : x \twoheadrightarrow_{\mathcal{R}} x : A}\,\mathsf{RVar} \qquad \frac{(\mathbf{c} : A) \in \mathcal{C}}{\Gamma \vdash \mathbf{c} : \mathbf{c} \twoheadrightarrow_{\mathcal{R}} \mathbf{c} : A}\,\mathsf{RCon} \qquad \frac{\Gamma, x : A \vdash \rho : s_0 \twoheadrightarrow_{\mathcal{R}} s_1 : B}{\Gamma \vdash \lambda x.\rho : \lambda x.s_0 \twoheadrightarrow_{\mathcal{R}} \lambda x.s_1 : A \to B}\,\mathsf{RAbs}$$

$$\frac{\Gamma \vdash \rho : s_0 \twoheadrightarrow_{\mathcal{R}} s_1 : A \to B \qquad \Gamma \vdash \sigma : t_0 \twoheadrightarrow_{\mathcal{R}} t_1 : A}{\Gamma \vdash \rho\,\sigma : s_0\,t_0 \twoheadrightarrow_{\mathcal{R}} s_1\,t_1 : B}\,\mathsf{RApp}$$

$$\frac{\langle \varrho, s, t\rangle \in \mathcal{R}^c \qquad \cdot \vdash s : A \qquad \cdot \vdash t : A}{\Gamma \vdash \varrho : s \twoheadrightarrow_{\mathcal{R}} t : A}\,\mathsf{RRule} \qquad \frac{\Gamma \vdash \rho : s_0 \twoheadrightarrow_{\mathcal{R}} s_1 : A \qquad \Gamma \vdash \sigma : s_1 \twoheadrightarrow_{\mathcal{R}} s_2 : A}{\Gamma \vdash \rho\,;\,\sigma : s_0 \twoheadrightarrow_{\mathcal{R}} s_2 : A}\,\mathsf{RTrans}$$

$$\frac{\Gamma \vdash \rho : s' \twoheadrightarrow_{\mathcal{R}} t' : A \qquad \Gamma \vdash s =_{\beta\eta} s' : A \qquad \Gamma \vdash t' =_{\beta\eta} t : A}{\Gamma \vdash \rho : s \twoheadrightarrow_{\mathcal{R}} t : A}\,\mathsf{RConv}$$

The RVar and RCon rules express that variables and constants represent identity rewrites. The RAbs and RApp rules express congruence below abstraction and application. The RRule rule allows us to use a rule symbol to stand for a rewrite between its source and its target, which must be closed terms of the same type. The RConv rule states that the source and the target of a rewrite are regarded up to $\beta\eta$-equivalence. Note that there are no rules equating rewrites; such rules are the purpose of Section 4 which introduces permutation equivalence.

▶ **Example 8.** Suppose we assign the following rule symbols to the rewriting rules of Ex. 4: $\langle \varrho, \mathbf{mu}(\lambda y.x\,y), x\,(\mathbf{mu}(\lambda y.x\,y))\rangle$ and $\langle \vartheta, \mathbf{f}\,x, \mathbf{g}\,x\rangle$. Recall that $\mathcal{C} \stackrel{\text{def}}{=} \{\mathbf{mu} : (\iota \to \iota) \to \iota, \mathbf{f} : \iota \to \iota\}$. The reduction of Ex. 4 can be represented as a rewrite:

$$\cdot \vdash \varrho\,(\lambda x.\mathbf{f}\,x)\;;\;\mathbf{f}\,(\mathbf{mu}\,(\lambda x.\vartheta\,x))\;;\;\vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) : \mathbf{mu}\,(\lambda x.\mathbf{f}\,x) \twoheadrightarrow_{\mathcal{R}} \mathbf{g}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x) : \iota$$

Inspection of the proof of Thm. 6 in [12] reveals that β and η are only needed for substitutions in rewrite rules. As a consequence:

▶ **Theorem 9.** *There is a rewrite ρ such that $\Gamma \vdash \rho : s \twoheadrightarrow_{\mathcal{R}} t : A$ if and only if $s \updownarrow_{\beta}^{\eta} \to_{\mathcal{R}}^{*} t \updownarrow_{\beta}^{\eta}$.*

Now that we know that rewrites over an HRS $\mathcal{R}$ are sound and complete with respect to reduction sequences in $\mathcal{R}$, we review some basic properties of rewrites and then focus, in the remaining sections, on equivalences between rewrites. In the sequel we will omit $\mathcal{R}$ in $\Gamma \vdash \rho : s \twoheadrightarrow_{\mathcal{R}} t : A$ and write $\Gamma \vdash \rho : s \twoheadrightarrow t : A$.

▶ **Definition 10** (Source and target of a rewrite)**.** *For each rewrite ρ we define the* source ρ^{src} *and the* target ρ^{tgt} *as the following terms:*

$$\begin{array}{rcl@{\qquad}rcl}
x^{\mathsf{src}} & \stackrel{\text{def}}{=} & x & x^{\mathsf{tgt}} & \stackrel{\text{def}}{=} & x \\
\mathbf{c}^{\mathsf{src}} & \stackrel{\text{def}}{=} & \mathbf{c} & \mathbf{c}^{\mathsf{tgt}} & \stackrel{\text{def}}{=} & \mathbf{c} \\
\varrho^{\mathsf{src}} & \stackrel{\text{def}}{=} & s \quad \textit{if } (\varrho : s \twoheadrightarrow t : A) \in \mathcal{R} & \varrho^{\mathsf{tgt}} & \stackrel{\text{def}}{=} & t \quad \textit{if } (\varrho : s \twoheadrightarrow t : A) \in \mathcal{R} \\
(\lambda x.\rho)^{\mathsf{src}} & \stackrel{\text{def}}{=} & \lambda x.\rho^{\mathsf{src}} & (\lambda x.\rho)^{\mathsf{tgt}} & \stackrel{\text{def}}{=} & \lambda x.\rho^{\mathsf{tgt}} \\
(\rho\,\sigma)^{\mathsf{src}} & \stackrel{\text{def}}{=} & \rho^{\mathsf{src}}\,\sigma^{\mathsf{src}} & (\rho\,\sigma)^{\mathsf{tgt}} & \stackrel{\text{def}}{=} & \rho^{\mathsf{tgt}}\,\sigma^{\mathsf{tgt}} \\
(\rho\,;\,\sigma)^{\mathsf{src}} & \stackrel{\text{def}}{=} & \rho^{\mathsf{src}} & (\rho\,;\,\sigma)^{\mathsf{tgt}} & \stackrel{\text{def}}{=} & \rho^{\mathsf{tgt}}
\end{array}$$

The free variables of an expression X (which may be a term or a rewrite) are written $\mathsf{fv}(X)$, and defined as expected, with lambdas binding variables in their bodies. For any given term or rewrite X, we write $X\{x\backslash t\}$ for the capture-avoiding substitution of the variable x in X by t. The operation $\rho\{x\backslash t\}$ is called *rewrite/term substitution.*

We mention a few important syntactic properties of terms and rewrites (detailed statements and proofs can be found in Section A of [2]). First, some basic properties hold, such as weakening (*e.g.* if $\Gamma \vdash \rho : s \twoheadrightarrow t : A$ then $\Gamma, x : B \vdash \rho : s \twoheadrightarrow t : A$) and commuting substitution with the source and target operators (*e.g.* $\rho\{x\backslash s\}^{\mathsf{src}} = \rho^{\mathsf{src}}\{x\backslash s\}$). Terms appearing in valid equality and rewriting judgments can always be shown to be typable, that is, if either $\Gamma \vdash s =_{\beta\eta} t : A$ or $\Gamma \vdash \rho : s \twoheadrightarrow t : A$, then $\Gamma \vdash s : A$ and $\Gamma \vdash t : A$. Second, given a typable rewrite, $\Gamma \vdash \rho : s \twoheadrightarrow t : A$, the source of ρ and s are not necessarily equal, but they are interconvertible, that is $\Gamma \vdash s =_{\beta\eta} \rho^{\mathsf{src}} : A$, and similarly for the target, *i.e.* $\Gamma \vdash t =_{\beta\eta} \rho^{\mathsf{tgt}} : A$. For example, if $\varrho : \lambda x.\mathbf{c}\, x \twoheadrightarrow \lambda x.\mathbf{d} : A \to A$ then it can be shown that $\vdash \varrho\, \mathbf{d} : \mathbf{c}\, \mathbf{d} \twoheadrightarrow \mathbf{d} : A$, and indeed $\mathbf{c}\, \mathbf{d} =_{\beta\eta} (\lambda x.\mathbf{c}\, x)\, \mathbf{d} = (\varrho\, \mathbf{d})^{\mathsf{src}}$. Third, any typable term s can be understood as an empty or *unit* rewrite $\underline{s}$, without occurrences of rule symbols, between s and itself: if $\Gamma \vdash s : A$ then $\Gamma \vdash \underline{s} : s \twoheadrightarrow s : A$. We usually coerce terms to rewrites implicitly if there is little danger of confusion. Substitution of a variable for a term is functorial, that is, given a rewrite $\Gamma, x : A \vdash \rho : s \twoheadrightarrow t : B$ and a term $\Gamma \vdash r : A$, then $\Gamma \vdash \rho\{x\backslash r\} : s\{x\backslash r\} \twoheadrightarrow t\{x\backslash r\} : B$.

Term/rewrite substitution generalizes term/term substitution $s\{x\backslash t\}$ when t is a rewrite, *i.e.* $s\{x\backslash\!\backslash\rho\}$. Sometimes we also call this notion *lifting substitution*, as $s\{x\backslash\!\backslash\rho\}$ "lifts" the expression s from the level of terms to the level of rewrites.

▶ **Definition 11** (Term/rewrite substitution).

$$y\{x\backslash\!\backslash\rho\} \stackrel{\text{def}}{=} \begin{cases} \rho & \text{if } x = y \\ y & \text{if } x \neq y \end{cases} \qquad \mathbf{c}\{x\backslash\!\backslash\rho\} \stackrel{\text{def}}{=} \mathbf{c}$$

$$(\lambda y.s)\{x\backslash\!\backslash\rho\} \stackrel{\text{def}}{=} \lambda y.s\{x\backslash\!\backslash\rho\} \quad \text{if } x \neq y \qquad (s\, t)\{x\backslash\!\backslash\rho\} \stackrel{\text{def}}{=} s\{x\backslash\!\backslash\rho\}\, t\{x\backslash\!\backslash\rho\}$$

We mention some important properties of term/rewrite substitution. First, term/rewrite substitution is a kind of *horizontal composition*, in the sense that if $\Gamma, x : A \vdash s : B$ and $\Gamma \vdash \rho : t \twoheadrightarrow t' : A$ then $\Gamma \vdash s\{x\backslash\!\backslash\rho\} : s\{x\backslash t\} \twoheadrightarrow s\{x\backslash t'\} : B$. Second, term/rewrite and rewrite/term substitution commute according to the equation $s\{x\backslash\!\backslash\rho\}\{y\backslash t\} = s\{y\backslash t\}\{x\backslash\!\backslash\rho\{y\backslash t\}\}$, assuming that $\Gamma, x : A, y : B \vdash s : C$ and $\Gamma, y : B \vdash \rho : r \twoheadrightarrow r' : A$ and $\Gamma \vdash t : B$ (where, by convention, $x \notin \mathsf{fv}(t)$). Note that, in particular, if y does not occur free in ρ, this means that $s\{x\backslash\!\backslash\rho\}\{y\backslash t\} = s\{y\backslash t\}\{x\backslash\!\backslash\rho\}$. Third, term/rewrite substitution commutes with reflexivity in the sense that $s\{x\backslash\!\backslash\underline{t}\} = \underline{s\{x\backslash t\}}$ holds whenever $\Gamma, x : A \vdash s : B$ and $\Gamma \vdash t : A$. It also commutes with the source and target operators, in the sense that $s\{x\backslash\!\backslash\rho\}^{\mathsf{src}} = s\{x\backslash\rho^{\mathsf{src}}\}$ and $s\{x\backslash\!\backslash\rho\}^{\mathsf{tgt}} = s\{x\backslash\rho^{\mathsf{tgt}}\}$ hold whenever $\Gamma, x : A \vdash s : B$ and $\Gamma \vdash \rho : t \twoheadrightarrow t' : A$.

4 Permutation equivalence

This section presents *permutation equivalence* (Def. 12), a relation over (typed) rewrites $\rho \approx \sigma$ that identifies any two rewrites ρ and σ denoting computations in a given HRS $\mathcal{R}$ that are equivalent up to permutation of steps.

Towards Permutation Equivalence for Rewrites. Equipped with the self-evident operations of *term/rewrite substitution* $s\{x\backslash\!\backslash\rho\}$, *rewrite/term substitution* $\rho\{x\backslash t\}$ and the fact that rewrites may be freely composed, we set out to synthesize a definition of permutation equivalence by attempting to assign a meaning for $(\lambda x.\rho)\, \sigma$, where $\Gamma \vdash \rho : s_0 \twoheadrightarrow s_1 : A$ and $\Gamma \vdash \sigma : t_0 \twoheadrightarrow t_1 : A$. We begin by assuming we have equations that allow rewrites to be post-composed with their targets ($\approx$-IdR) and pre-composed with their source ($\approx$-IdL) and reason as follows:

$$(\lambda x.\rho)\, \sigma \quad \approx^{(\mathsf{IdR})} \quad ((\lambda x.\rho)\, ;\, (\lambda x.s_1))\, \sigma \quad \approx^{(\mathsf{IdL})} \quad ((\lambda x.\rho)\, ;\, (\lambda x.s_1))\, (t_0\, ;\, \sigma)$$

These rewrites are syntactically valid since we allow composition inside an application. Next, we allow application to commute with composition by introducing a rule $\approx$-App: $(\rho_1\rho_2)\ ;\ (\sigma_1\sigma_2) \approx (\rho_1\ ;\ \sigma_1)(\rho_2\ ;\ \sigma_2)$. Applying this equation leads us to:

$$((\lambda x.\rho)\ ;\ (\lambda x.s_1))\,(t_0\ ;\ \sigma) \quad \approx^{(\mathsf{App})} \quad (\lambda x.\rho)\,t_0\ ;\ (\lambda x.s_1)\,\sigma$$

Finally, we introduce β-equality on rewrites. Arbitrary β-reduction of rewrites is not allowed *a priori*. It is only allowed when either the abstraction or the argument are unit rewrites, for which the substitution operators mentioned above can be used. These equations take the form $(\lambda x.\underline{s})\,\rho \approx s\{x \Vdash \rho\}$ and $(\lambda x.\rho)\,\underline{s} \approx \rho\{x \backslash s\}$ and are called, $\approx$-BetaTR and $\approx$-BetaRT.

$$(\lambda x.\rho)\,t_0\ ;\ (\lambda x.s_1)\,\sigma \quad \approx^{(\mathsf{BetaRT})} \quad \rho\{x \backslash t_0\}\ ;\ (\lambda x.s_1)\,\sigma \quad \approx^{(\mathsf{BetaTR})} \quad \rho\{x \backslash t_0\}\ ;\ s_1\{x \Vdash \sigma\}$$

In summary we have $(\lambda x.\rho)\,\sigma \approx \rho\{x \backslash t_0\}\ ;\ s_1\{x \Vdash \sigma\}$. We could equally well have deduced $(\lambda x.\rho)\,\sigma \approx s_0\{x \Vdash \sigma\}\ ;\ \rho\{x \backslash t_1\}$. As it turns out, however, $\rho\{x \backslash t_0\}\ ;\ s_1\{x \Vdash \sigma\}$ and $s_0\{x \Vdash \sigma\}\ ;\ \rho\{x \backslash t_1\}$ are permutation equivalent in our theory.

Permutation Equivalence for Rewrites: Definition and Properties. We collect the observations above in the following definition.

▶ **Definition 12** (Permutation equivalence). *Suppose* $\Gamma \vdash \rho : s \twoheadrightarrow t : A$ *and* $\Gamma \vdash \rho' : s' \twoheadrightarrow t' : A$ *are derivable. Permutation equivalence, written* $\Gamma \vdash (\rho : s \twoheadrightarrow t) \approx (\rho' : s' \twoheadrightarrow t') : A$ *(or simply* $\rho \approx \rho'$ *if* Γ, s, t, s', t', A *are clear from the context), is defined as the reflexive, symmetric, transitive, and contextual closure of the following axioms:*

$$\begin{array}{rcll}
\underline{\rho^{\mathsf{src}}}\ ;\ \rho & \approx & \rho & \approx\text{-IdL} \\
\rho\ ;\ \underline{\rho^{\mathsf{tgt}}} & \approx & \rho & \approx\text{-IdR} \\
(\rho\ ;\ \sigma)\ ;\ \tau & \approx & \rho\ ;\ (\sigma\ ;\ \tau) & \approx\text{-Assoc} \\
(\lambda x.\rho)\ ;\ (\lambda x.\sigma) & \approx & \lambda x.(\rho\ ;\ \sigma) & \approx\text{-Abs} \\
(\rho_1\rho_2)\ ;\ (\sigma_1\sigma_2) & \approx & (\rho_1\ ;\ \sigma_1)(\rho_2\ ;\ \sigma_2) & \approx\text{-App} \\
(\lambda x.\underline{s})\,\rho & \approx & s\{x \Vdash \rho\} & \approx\text{-BetaTR} \\
(\lambda x.\rho)\,\underline{s} & \approx & \rho\{x \backslash s\} & \approx\text{-BetaRT} \\
\lambda x.\rho\,x & \approx & \rho \quad \textit{if } x \notin \mathsf{fv}(\rho) & \approx\text{-Eta}
\end{array}$$

Rules $\approx$-IdL, $\approx$-IdR and $\approx$-Assoc, state that rewrites together with rewrite composition have a monoidal structure. Recall from Section 3 that ρ^{src} is a term and $\underline{\rho^{\mathsf{src}}}$ is its corresponding rewrite. Rules $\approx$-Abs and $\approx$-App state that rewrite composition commutes with abstraction and application. An important thing to be wary of is that rules may be applied only if both the left and the right-hand sides are well-typed. In particular, the right-hand side of the $\approx$-App rule may not be well-typed even if the left-hand side is; for example given rule symbols $\mathbf{c} : A \to B$ and $\mathbf{d} : A$, the expression $((\lambda x.x)(\mathbf{c}\,\mathbf{d}))\ ;\ (\mathbf{c}\,\mathbf{d})$ is well-typed, with source and target $\mathbf{c}\,\mathbf{d}$, while $((\lambda x.x)\ ;\ \mathbf{c})\,((\mathbf{c}\,\mathbf{d})\ ;\ \mathbf{d})$ is not well-typed.

Finally, rules $\approx$-BetaTR, $\approx$-BetaRT and $\approx$-Eta introduce $\beta\eta$-equivalence for rewrites. Note that $\approx$-BetaTR and $\approx$-BetaRT restrict either the body of the abstraction or the argument to a unit rewrite, thus avoiding the issue mentioned in the introduction where a naive combination of composition and $\beta\eta$-equivalence can lead to invalid rewrites.

Note that there are no explicit sequencing equations such as the I/O equations[2] defining permutation equivalence in the first-order case [15] and the corresponding equations flat-l and flat-r of [4] for the higher-order case. Nonetheless, we can derive the following coherence equation (see Lem. 63 and Lem. 64 in [2] for the proof):

$$\rho\{x\backslash s'\} \mathbin{;} t\{x\Vert\sigma\} \quad \approx \quad s\{x\Vert\sigma\} \mathbin{;} \rho\{x\backslash t'\} \quad (\approx\text{-Perm})$$

where $\Gamma, x : A \vdash \rho : s \twoheadrightarrow t : B$ and $\Gamma \vdash \sigma : s' \twoheadrightarrow t' : A$.

▶ **Example 13.** Consider the HRS of Ex. 4 and the reduction of Ex. 8. We recall the latter below (R_2) and present a second one (R_1).

$$\begin{array}{lcl} R_1 & : & \mathbf{mu}\,(\lambda x.\mathbf{f}\,x) \twoheadrightarrow \mathbf{mu}\,(\lambda x.\mathbf{g}\,x) \twoheadrightarrow \mathbf{g}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) \\ R_2 & : & \mathbf{mu}\,(\lambda x.\mathbf{f}\,x) \twoheadrightarrow \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\mathbf{f}\,x)) \twoheadrightarrow \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) \twoheadrightarrow \mathbf{g}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) \end{array}$$

Reduction sequence R_1 can be encoded as the rewrite $\mathbf{mu}\,(\lambda x.\vartheta\,x); \varrho\,(\lambda x.\mathbf{g}\,x)$ and R_2 as $\varrho\,(\lambda x.\mathbf{f}\,x) \mathbin{;} \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\vartheta\,x)) \mathbin{;} \vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x))$. These two rewrites are permutation equivalent:

$$\begin{array}{ll} & \mathbf{mu}\,(\lambda x.\vartheta\,x) \mathbin{;} \varrho\,(\lambda x.\mathbf{g}\,x) \\ \approx^{(\mathsf{Eta})} & \mathbf{mu}\,\vartheta \mathbin{;} \varrho\,\mathbf{g} \\ = & (\mathbf{mu}\,y)\{y\Vert\vartheta\} \mathbin{;} (\varrho\,y)\{y\backslash\mathbf{g}\} \\ \approx^{(\mathsf{Perm})} & (\varrho\,y)\{y\backslash\mathbf{f}\} \mathbin{;} (y\,(\mathbf{mu}\,y))\{y\Vert\vartheta\} \\ = & \varrho\,\mathbf{f} \mathbin{;} \vartheta\,(\mathbf{mu}\,\vartheta) \\ \approx^{(\mathsf{IdL})} & \varrho\,\mathbf{f} \mathbin{;} (\mathbf{f} \mathbin{;} \vartheta)\,(\mathbf{mu}\,\vartheta) \\ \approx^{(\mathsf{IdR})} & \varrho\,\mathbf{f} \mathbin{;} (\mathbf{f} \mathbin{;} \vartheta)\,((\mathbf{mu}\,\vartheta) \mathbin{;} (\mathbf{mu}\,\mathbf{g})) \\ \approx^{(\mathsf{App})} & \varrho\,\mathbf{f} \mathbin{;} \mathbf{f}\,(\mathbf{mu}\,\vartheta) \mathbin{;} \vartheta\,(\mathbf{mu}\,\mathbf{g}) \\ \approx^{(\mathsf{Eta})} & \varrho\,(\lambda x.\mathbf{f}\,x) \mathbin{;} \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\vartheta\,x)) \mathbin{;} \vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) \end{array}$$

The $\approx$-Perm rule motivates the definition of *rewrite/rewrite substitution*, $\rho\{x\Vert\sigma\} \stackrel{\text{def}}{=} \rho\{x\backslash s'\} \mathbin{;} t\{x\Vert\sigma\}$, which defines a rewrite $s\{x\backslash s'\} \twoheadrightarrow t\{x\backslash t'\}$. Note that $\rho\{x\Vert\sigma\}$ depends on t and s', and hence on the particular typing derivations for ρ and σ. Congruence results (Lem. 63 and Lem. 64 in [2]) ensure that the value of $\rho\{x\Vert\sigma\}$ does not depend, up to permutation equivalence, on those typing derivations. Rewrite/rewrite substitution generalizes rewrite/term and term/rewrite substitution, in the sense that $\rho\{x\backslash t\} \approx \rho\{x\Vert\underline{t}\}$ and $s\{x\Vert\rho\} \approx \underline{s}\{x\Vert\rho\}$.

Other important facts involving rewrite/rewrite substitution are the following. First, it commutes with abstraction, application, and composition, that is $(\lambda y.\rho)\{x\Vert\sigma\} \approx \lambda y.\rho\{x\Vert\sigma\}$, $(\rho_1\,\rho_2)\{x\Vert\sigma\} \approx \rho_1\{x\Vert\sigma\}\,\rho_2\{x\Vert\sigma\}$, and $(\rho_1 \mathbin{;} \rho_2)\{x\Vert\sigma_1 \mathbin{;} \sigma_2\} \approx \rho_1\{x\Vert\sigma_1\} \mathbin{;} \rho_2\{x\Vert\sigma_2\}$. Second, permutation equivalence is a congruence with respect to rewrite/rewrite substitution, that is, if $\rho \approx \rho'$ and $\sigma \approx \sigma'$ then $\rho\{x\Vert\sigma\} \approx \rho'\{x\Vert\sigma'\}$. Third, an analog of the substitution lemma holds, namely $\rho\{x\Vert\sigma\}\{y\Vert\tau\} \approx \rho\{y\Vert\tau\}\{x\Vert\sigma\{y\Vert\tau\}\}$. Finally, as discussed above, a β-rule for arbitrary rewrites holds in the form $(\lambda x.\rho)\,\sigma \approx \rho\{x\Vert\sigma\}$. The full theory of rewrite/rewrite substitution is not developed here for lack of space (but see Section B.2 in [2]).

5 Flattening

Allowing composition to be nested within application and abstraction can give rise to rewrites in which it is not obvious what reduction sequences of steps are being denoted. An example from the previous section might be the rewrite $((\lambda x.\mathbf{f}\,x) \mathbin{;} \vartheta)\,((\mathbf{mu}\,(\lambda x.\vartheta\,x)) \mathbin{;} (\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)))$

[2] $I : \varrho(\sigma_1, ..., \sigma_n) \approx l(\sigma_1, ..., \sigma_n) \cdot \varrho(t_1, ..., t_n)$ and $O : \varrho(\sigma_1, ..., \sigma_n) \approx \varrho(s_1, ..., s_n) \cdot r(\sigma_1, ..., \sigma_n)$

which denotes the reduction sequence $\mathbf{f}\,(\mathbf{mu}\,(\lambda x.\mathbf{f}\,x)) \twoheadrightarrow \mathbf{g}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x))$ that replaces both occurrences of $\mathbf{f}$ with $\mathbf{g}$ simultaneously. This section shows how rewrites can be "flattened" so as to expose an underlying reduction sequence, expressed as a canonical (*flat*) rewrite. One additional use of flattening will be to use it to show that permutation equivalence is decidable (*cf.* end of Sec. Section 6). Before introducing flat rewrites we define *multisteps*.

A *multistep* is a rewrite without any occurrences of the composition operator. We use $\mu, \nu, \xi, \ldots$ to range over multisteps. The capture-avoiding substitution of the free occurrences of x in μ by ν is written $\mu\{x\backslash\nu\}$, which is in turn a multistep. A *flat multistep* $(\hat{\mu}, \hat{\nu}, \ldots)$, is a multistep in β-normal form, *i.e.* without subterms of the form $(\lambda x.\mu)\,\nu$. A *flat rewrite* $(\hat{\rho}, \hat{\sigma}, \ldots)$, is a rewrite given by the grammar $\hat{\rho} ::= \hat{\mu} \mid \hat{\rho}\ ;\ \hat{\sigma}$. Flat rewrites use the composition operator "**;**" at the top level, that is they are of the form $\hat{\mu}_1\ ;\ \ldots\ ;\ \hat{\mu}_n$ (up to associativity of "**;**"), where each $\hat{\mu}_i$ is a flat multistep. Note that we do not require the $\hat{\mu}_i$ to be in $\beta\eta$-normal form nor in $\beta\overline{\eta}$-normal form. As mentioned in the introduction, flattening is achieved by means of a *rewriting system whose objects are themselves rewrites* (Def. 15) which is shown to be *confluent and terminating* (Prop. 17).

We also formulate an equational theory defining a relation $\rho \sim \sigma$ of *flat permutation equivalence* between flat rewrites (Def. 19). The main result of this section is that permutation equivalence is *sound and complete* with respect to flat permutation equivalence (Thm. 20).

▶ Remark 14. A substitution $\mu\{x\backslash\nu\}$ in which μ is a term is a term/rewrite substitution, *i.e.* $s\{x\backslash\nu\} = s\{x\backslash\!\backslash\nu\}$. A substitution in which ν is a term is a rewrite/term substitution, *i.e.* $\mu\{x\backslash s\} = \mu\{x\backslash s\}$.

▶ **Definition 15** (Flattening Rewrite System $\mathcal{F}$). *The flattening system $\mathcal{F}$ is given by the following rules, closed under arbitrary contexts, defined between* **typable** *rewrites:*

$$
\begin{array}{rcll}
\lambda x.(\rho\ ;\ \sigma) & \overset{\flat}{\mapsto} & (\lambda x.\rho)\ ;\ (\lambda x.\sigma) & \mathcal{F}\text{-Abs} \\
(\rho\ ;\ \sigma)\,\mu & \overset{\flat}{\mapsto} & (\rho\,\underline{\mu^{\mathsf{src}}})\ ;\ (\sigma\,\mu) & \mathcal{F}\text{-App1} \\
\mu\,(\rho\ ;\ \sigma) & \overset{\flat}{\mapsto} & (\mu\,\rho)\ ;\ (\underline{\mu^{\mathsf{tgt}}}\,\sigma) & \mathcal{F}\text{-App2} \\
(\rho_1\ ;\ \rho_2)\,(\sigma_1\ ;\ \sigma_2) & \overset{\flat}{\mapsto} & ((\rho_1\ ;\ \rho_2)\,\underline{\sigma_1^{\mathsf{src}}})\ ;\ (\underline{\rho_2^{\mathsf{tgt}}}\,(\sigma_1\ ;\ \sigma_2)) & \mathcal{F}\text{-App3} \\
(\lambda x.\mu)\,\nu & \overset{\flat}{\mapsto} & \mu\{x\backslash\nu\} & \mathcal{F}\text{-BetaM} \\
\lambda x.\mu\,x & \overset{\flat}{\mapsto} & \mu \quad \textit{if } x \notin \mathsf{fv}(\mu) & \mathcal{F}\text{-EtaM}
\end{array}
$$

Note that rules $\mathcal{F}$-BetaM and $\mathcal{F}$-EtaM apply to multisteps only. The reduction relation $\overset{\flat}{\mapsto}$ is the union of all these rules, closed by compatibility under arbitrary contexts. We write $\rho^{\flat}$ for the unique $\overset{\flat}{\mapsto}$-normal form of ρ.

▶ **Example 16.** Consider a rewriting rule $\varrho : \mathbf{c} \twoheadrightarrow \mathbf{d} : A$. The rewrite $(\lambda x.(x\ ;\ x))\,\varrho$, whose meaning (as previously mentioned) is not obvious, can be flattened as follows:

$$
\begin{array}{lllll}
(\lambda x.(x\ ;\ x))\,\varrho & \overset{\flat}{\mapsto}_{\mathcal{F}\text{-Abs}} & ((\lambda x.x)\ ;\ (\lambda x.x))\,\varrho & \overset{\flat}{\mapsto}_{\mathcal{F}\text{-App1}} & (\lambda x.x)\,\mathbf{c}\ ;\ (\lambda x.x)\,\varrho \\
 & \overset{\flat}{\mapsto}_{\mathcal{F}\text{-BetaM}} & \mathbf{c}\ ;\ (\lambda x.x)\,\varrho & \overset{\flat}{\mapsto}_{\mathcal{F}\text{-BetaM}} & \mathbf{c}\ ;\ \varrho
\end{array}
$$

The following result is proved by noting that $\mathcal{F}$-BetaM and $\mathcal{F}$-EtaM steps can be postponed after steps of other kinds and then providing a well-founded measure for steps in $\mathcal{F}$ without $\mathcal{F}$-BetaM and $\mathcal{F}$-EtaM to prove it is SN. Confluence of $\mathcal{F}$ follows from Newman's lemma.

▶ **Proposition 17.** *The flattening system $\mathcal{F}$ is strongly normalizing and confluent.*

Flat Permutation Equivalence. We now turn to the definition of the relation $\rho \sim \sigma$ of flat permutation equivalence. The key notion to define is the following ternary relation:

▶ **Definition 18** (Splitting). *Let* $\Gamma \vdash \mu : s \twoheadrightarrow t : A$ *and* $\Gamma \vdash \mu_1 : s' \twoheadrightarrow r_1 : A$ *and* $\Gamma \vdash \mu_2 : r_2 \twoheadrightarrow t' : A$ *be multisteps. We say that* μ splits *into* μ_1 *and* μ_2 *if the following inductively defined ternary relation, written* $\mu \Leftrightarrow \mu_1 \mathbin{;} \mu_2$*, holds:*

$$\frac{}{x \Leftrightarrow x \mathbin{;} x}\,\mathsf{SVar} \qquad \frac{}{\mathbf{c} \Leftrightarrow \mathbf{c} \mathbin{;} \mathbf{c}}\,\mathsf{SCon} \qquad \frac{}{\varrho \Leftrightarrow \varrho \mathbin{;} \underline{\varrho^{\mathsf{tgt}}}}\,\mathsf{SRuleL} \qquad \frac{}{\varrho \Leftrightarrow \underline{\varrho^{\mathsf{src}}} \mathbin{;} \varrho}\,\mathsf{SRuleR}$$

$$\frac{\mu \Leftrightarrow \mu_1 \mathbin{;} \mu_2}{\lambda x.\mu \Leftrightarrow \lambda x.\mu_1 \mathbin{;} \lambda x.\mu_2}\,\mathsf{SAbs} \qquad \frac{\mu \Leftrightarrow \mu_1 \mathbin{;} \mu_2 \quad \nu \Leftrightarrow \nu_1 \mathbin{;} \nu_2}{\mu\,\nu \Leftrightarrow \mu_1\,\nu_1 \mathbin{;} \mu_2\,\nu_2}\,\mathsf{SApp}$$

▶ **Definition 19** (Flat permutation equivalence). *Flat permutation equivalence judgments are of the form:* $\Gamma \vdash (\rho : s \twoheadrightarrow t) \sim (\rho' : s' \twoheadrightarrow t') : A$*, meaning that* ρ *and* ρ' *are equivalent rewrites, with sources* s *and* s' *respectively, and targets* t *and* t' *respectively. The rewrites* ρ *and* ρ' *are* **assumed to be in** $\overset{\flat}{\mapsto}$**-normal form**, *which in particular means that they must be flat rewrites. Sometimes we write* $\rho \sim \rho'$ *if* Γ, s, t, s', t', A *are irrelevant or clear from the context. Derivability is defined by the two following axioms, which are closed by reflexivity, symmetry, transitivity, and closure under* composition contexts *(given by* $\mathtt{S} ::= \Box \mid \mathtt{S} \mathbin{;} \rho \mid \rho \mathbin{;} \mathtt{S}$*):*

$$(\rho \mathbin{;} \sigma) \mathbin{;} \tau \sim \rho \mathbin{;} (\sigma \mathbin{;} \tau) \qquad\qquad \sim\text{-Assoc}$$

$$\mu \sim \mu_1^\flat \mathbin{;} \mu_2^\flat \quad \text{if } \mu \Leftrightarrow \mu_1 \mathbin{;} \mu_2 \qquad \sim\text{-Perm}$$

Note that in $\sim$-Perm, $-^\flat$ operates over multisteps. So the only rules of $\mathcal{F}$ that are applied here are the $\mathcal{F}$-BetaM and $\mathcal{F}$-EtaM rules.

▶ **Theorem 20** (Soundness and completeness of flat permutation equivalence). *Let* $\Gamma \vdash \rho : s \twoheadrightarrow t : A$ *and* $\Gamma \vdash \sigma : s' \twoheadrightarrow t' : A$*. Then* $\rho \approx \sigma$ *if and only if* $\rho^\flat \sim \sigma^\flat$*.*

Proof. The ($\Leftarrow$) direction is immediate, given that reduction $\overset{\flat}{\mapsto}$ in the flattening system $\mathcal{F}$ is included in permutation equivalence ($\rho \overset{\flat}{\mapsto} \sigma$ implies $\rho \approx \sigma$) and, similarly, flat permutation equivalence is included in permutation equivalence ($\rho \sim \sigma$ implies $\rho \approx \sigma$).

The ($\Rightarrow$) direction is by induction on the derivation of $\rho \approx \sigma$. It is subtle and requires numerous auxiliary results (see Section D.8 in [2]). ◀

▶ **Example 21.** With the same notation as in Ex. 13, it can be checked that the rewrites $\mathbf{mu}\,(\lambda x.\vartheta\,x) \mathbin{;} \varrho\,(\lambda x.\mathbf{g}\,x)$ and $\varrho\,(\lambda x.\mathbf{f}\,x) \mathbin{;} \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\vartheta\,x)) \mathbin{;} \vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x))$ are permutation equivalent by means of flattening. Indeed, using the $\sim$-Perm rule three times:

$$\begin{array}{lcll} \mathbf{mu}\,\vartheta \mathbin{;} \varrho\,\mathbf{g} & \sim & \varrho\,\vartheta & \text{as } \varrho\,\vartheta \Leftrightarrow (\lambda x.\mathbf{mu}\,(\lambda y.x\,y))\,\vartheta \mathbin{;} \varrho\,(\lambda x.\mathbf{g}\,x) \\ & \sim & \varrho\,\mathbf{f} \mathbin{;} \vartheta\,(\mathbf{mu}\,\vartheta) & \text{as } \varrho\,\vartheta \Leftrightarrow \varrho\,(\lambda x.\mathbf{f}\,x) \mathbin{;} (\lambda x.x\,(\mathbf{mu}\,(\lambda y.x\,y)))\,\vartheta \\ & \sim & \varrho\,\mathbf{f} \mathbin{;} (\mathbf{f}(\mathbf{mu}\,\vartheta) \mathbin{;} \vartheta(\mu\,\mathbf{g})) & \text{as } \vartheta\,(\mathbf{mu}\,\vartheta) \Leftrightarrow (\lambda x.\mathbf{f}\,x)\,(\mathbf{mu}\,\vartheta) \mathbin{;} \vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) \end{array}$$

Note that $\varrho\,\vartheta \Leftrightarrow (\lambda x.\mathbf{mu}\,(\lambda y.x\,y))\,\vartheta \mathbin{;} \varrho\,(\lambda x.\mathbf{g}\,x)$ follows from SApp, SRuleR for the upper left hypothesis and SRuleL for the upper right one. Hence

$$\begin{array}{lcl} (\mathbf{mu}\,(\lambda x.\vartheta\,x) \mathbin{;} \varrho\,(\lambda x.\mathbf{g}\,x))^\flat & = & \mathbf{mu}\,\vartheta \mathbin{;} \varrho\,\mathbf{g} \\ & \sim & \varrho\,\mathbf{f} \mathbin{;} (\mathbf{f}(\mathbf{mu}\,\vartheta) \mathbin{;} \vartheta(\mu\,\mathbf{g})) \\ & = & (\varrho\,(\lambda x.\mathbf{f}\,x) \mathbin{;} \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\vartheta\,x)) \mathbin{;} \vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)))^\flat \end{array}$$

6 Projection

This section presents projection equivalence. Two rewrites ρ and σ are said to be projection equivalent if the steps performed by ρ are included in those performed by σ and vice-versa. We proceed in stages as follows. First, we define *projection* of multisteps over multisteps (Def. 25) and prove some of its properties (Prop. 26). Second, we extend projection to flat rewrites (Def. 28). Third, we extend projection to arbitrary rewrites (Def. 29) and, again, we prove some of its properties (Prop. 30). Finally, we show that the induced notion of *projection equivalence* turns out to coincide with permutation equivalence (Thm. 31).

Projection for Multisteps. Consider the rewrites $\mathbf{mu}\,\vartheta$ and $\varrho\,\mathbf{f}$, using the notation of Ex. 13, each representing one step. Since rewrites are subject to $\beta\eta$-equivalence, to define projection one must "line up" rule symbols with the left-hand side of the rewrite rules they witness[3]. For example, if the above two multisteps were rewritten as $(\lambda y.\mathbf{mu}\,(\lambda x.y\,x))\,\vartheta$ and $\varrho\,(\lambda x.\mathbf{f}\,x)$, respectively, then one can reason inductively as follows to compute the projection of the former over the latter (the inference rules themselves are introduced in Def. 22):

$$\dfrac{\dfrac{}{\lambda y.\mathbf{mu}\,(\lambda x.y\,x)\mathbin{/\!\!/\!\!/}\varrho \Rightarrow \lambda y.y\,(\mathbf{mu}\,(\lambda x.y\,x))}\,\mathsf{ProjRuleR} \qquad \dfrac{}{\vartheta \mathbin{/\!\!/\!\!/} \lambda x.\mathbf{f}\,x \Rightarrow \vartheta}\,\mathsf{ProjRuleL}}{(\lambda y.\mathbf{mu}\,(\lambda x.y\,x))\,\vartheta \mathbin{/\!\!/\!\!/} \varrho\,(\lambda x.\mathbf{f}\,x) \Rightarrow (\lambda y.y\,(\mathbf{mu}\,(\lambda x.y\,x)))\,\vartheta}\,\mathsf{ProjApp}$$

The flat normal form of $(\lambda y.y\,(\mathbf{mu}\,(\lambda x.y\,x)))\,\vartheta$ is the rewrite $\vartheta\,(\mathbf{mu}\,\vartheta)$. Hence we would deduce $\mathbf{mu}\,\vartheta \mathbin{/\!\!/\!\!/} \varrho\,\mathbf{f} \Rightarrow \vartheta\,(\mathbf{mu}\,\vartheta)$. We begin by introducing an auxiliary notion of projection on coinitial multisteps that may not be flat (*i.e.* may not be in $\mathcal{F}$-BetaM, $\mathcal{F}$-EtaM-normal form) called *weak projection*. We then make use of this notion, to define projection for flat multisteps (Def. 25).

▶ **Definition 22** (Weak projection and compatibility). *Let* $\Gamma \vdash \mu : s \twoheadrightarrow t : A$ *and* $\Gamma \vdash \nu : s' \twoheadrightarrow r : A$ *be multisteps, not necessarily in normal form, such that* $s =_{\beta\eta} s'$*. The judgment* $\mu \mathbin{/\!\!/\!\!/} \nu \Rightarrow \xi$ *is defined as follows:*

$$\dfrac{}{x \mathbin{/\!\!/\!\!/} x \Rightarrow x}\,\mathsf{ProjVar} \qquad \dfrac{}{\mathbf{c} \mathbin{/\!\!/\!\!/} \mathbf{c} \Rightarrow \mathbf{c}}\,\mathsf{ProjCon} \qquad \dfrac{}{\varrho \mathbin{/\!\!/\!\!/} \varrho \Rightarrow \varrho^{\mathsf{tgt}}}\,\mathsf{ProjRule} \qquad \dfrac{}{\varrho \mathbin{/\!\!/\!\!/} \varrho^{\mathsf{src}} \Rightarrow \varrho}\,\mathsf{ProjRuleL}$$

$$\dfrac{}{\varrho^{\mathsf{src}} \mathbin{/\!\!/\!\!/} \varrho \Rightarrow \varrho^{\mathsf{tgt}}}\,\mathsf{ProjRuleR} \qquad \dfrac{\mu \mathbin{/\!\!/\!\!/} \nu \Rightarrow \xi}{\lambda x.\mu \mathbin{/\!\!/\!\!/} \lambda x.\nu \Rightarrow \lambda x.\xi}\,\mathsf{ProjAbs} \qquad \dfrac{\mu_1 \mathbin{/\!\!/\!\!/} \nu_1 \Rightarrow \xi_1 \qquad \mu_2 \mathbin{/\!\!/\!\!/} \nu_2 \Rightarrow \xi_2}{\mu_1\,\mu_2 \mathbin{/\!\!/\!\!/} \nu_1\,\nu_2 \Rightarrow \xi_1\,\xi_2}\,\mathsf{ProjApp}$$

We say that μ *and* ν *are* compatible*, written* $\mu \uparrow \nu$ *if, intuitively speaking,* μ *and* ν *are coinitial, and are "almost"* η*-expanded and* β*-normal forms, with the exception that the head of the term may be the source of a rule,* i.e. *a term of the form* ϱ^{src}*. Compatibility is defined as follows:*

$$\dfrac{(\mu_i \uparrow \nu_i)_{i=1}^m}{\lambda\overline{x}.y\,\overline{\mu} \uparrow \lambda\overline{x}.y\,\overline{\nu}} \qquad \dfrac{(\mu_i \uparrow \nu_i)_{i=1}^m}{\lambda\overline{x}.\mathbf{c}\,\overline{\mu} \uparrow \lambda\overline{x}.\mathbf{c}\,\overline{\nu}} \qquad \dfrac{(\mu_i \uparrow \nu_i)_{i=1}^m}{\lambda\overline{x}.\varrho\,\overline{\mu} \uparrow \lambda\overline{x}.\varrho\,\overline{\nu}} \qquad \dfrac{(\mu_i \uparrow \nu_i)_{i=1}^m}{\lambda\overline{x}.\varrho\,\overline{\mu} \uparrow \lambda\overline{x}.\varrho^{\mathsf{src}}\,\overline{\nu}} \qquad \dfrac{(\mu_i \uparrow \nu_i)_{i=1}^m}{\lambda\overline{x}.\varrho^{\mathsf{src}}\,\overline{\mu} \uparrow \lambda\overline{x}.\varrho\,\overline{\nu}}$$

[3] See also the discussion on pg. 120 of [4].

The interesting cases are the two last rules, which state essentially that a rule symbol is compatible with its source term. Clearly if $\mu \uparrow \nu$, then there exists a unique ξ such that $\mu /\!\!/\!/ \nu \Rightarrow \xi$. Moreover, weak projection is coherent with respect to flattening:

▶ **Lemma 23** (Coherence of projection). *Let $\mu_1, \nu_1, \mu_2, \nu_2$ be multisteps such that the following are satisfied:*
1. *$\mu_1 \uparrow \nu_1$ and $\mu_2 \uparrow \nu_2$;*
2. *$\mu_1^\flat = \mu_2^\flat$ and $\nu_1^\flat = \nu_2^\flat$; and*
3. *$\mu_1 /\!\!/\!/ \nu_1 \Rightarrow \xi_1$ and $\mu_2 /\!\!/\!/ \nu_2 \Rightarrow \xi_2$.*

Then $\xi_1^\flat = \xi_2^\flat$.

Thus for arbitrary, coinitial multisteps μ and ν, it suffices to show that we can always find corresponding *compatible* "almost" η-expanded and β-normal forms, as mentioned above.

▶ **Proposition 24** (Existence and uniqueness of projection). *Let μ, ν be such that $\mu^{\mathsf{src}} =_{\beta\eta} \nu^{\mathsf{src}}$. Then:*
1. Existence. *There exist multisteps $\dot\mu, \dot\nu, \dot\xi$ such that $\dot\mu^\flat = \mu^\flat$ and $\dot\nu^\flat = \nu^\flat$ and $\dot\mu /\!\!/\!/ \dot\nu \Rightarrow \dot\xi$.*
2. Compatibility. *Furthermore, $\dot\mu$ and $\dot\nu$ can be chosen in such a way that $\dot\mu \uparrow \dot\nu$.*
3. Uniqueness. *If $(\dot\mu')^\flat = \mu^\flat$ and $(\dot\nu')^\flat = \nu^\flat$ and $\dot\mu' /\!\!/\!/ \dot\nu' \Rightarrow \dot\xi'$ then $(\dot\xi')^\flat = \xi^\flat$.*

Prop. 24 relies on the left-hand side of the rewrite rules of the HRS being patterns. This ensures, among other things, that flattening is injective when applied to left-hand sides of rewrite rules in the sense that if $(\varrho^{\mathsf{src}}\, \mu_1 \dots \mu_n)^\flat = (\varrho^{\mathsf{src}}\, \nu_1 \dots \nu_n)^\flat$ then $\mu_i^\flat = \nu_i^\flat$ for all $1 \le i \le n$. We can now define projection on arbitrary coinitial rewrites as follows.

▶ **Definition 25** (Projection operator for multisteps). *Let μ, ν be such that $\mu^{\mathsf{src}} =_{\beta\eta} \nu^{\mathsf{src}}$. We write μ/ν for the unique multistep of the form $\dot\xi^\flat$ such that there exist $\dot\mu, \dot\nu$ such that $\dot\mu^\flat = \mu^\flat$ and $\dot\nu^\flat = \nu^\flat$ and $\dot\mu /\!\!/\!/ \dot\nu \Rightarrow \dot\xi$, as guaranteed by Prop. 24. The proof is constructive (this relies on the HRS being orthogonal), thus providing an effective method to compute μ/ν.*

▶ **Proposition 26** (Properties of projection for multisteps).
1. *$\mu/\nu = (\mu/\nu)^\flat = \mu^\flat/\nu^\flat$*
2. *Projection commutes with abstraction and application, that is, $(\lambda x.\mu)/(\lambda x.\nu) = (\lambda x.(\mu/\nu))^\flat$ and $(\mu_1\, \mu_2)/(\nu_1\, \nu_2) = ((\mu_1/\nu_1)\, (\mu_2/\nu_2))^\flat$, provided that μ_1/ν_1 and μ_2/ν_2 are defined.*
3. *The set of multisteps with the projection operator form a residual system [15, Def. 8.7.2]:*
   - 3.1 *$(\mu/\nu)/(\xi/\nu) = (\mu/\xi)/(\nu/\xi)$, known as the* Cube Lemma.
   - 3.2 *$\mu/\mu = (\mu^{\mathsf{tgt}})^\flat$ and, as particular cases: $\underline{s}/\underline{s} = \underline{s}^\flat$, $x/x = x$, $\mathbf{c}/\mathbf{c} = \mathbf{c}$, and $\varrho/\varrho = (\varrho^{\mathsf{tgt}})^\flat$.*
   - 3.3 *$(\mu^{\mathsf{src}})^\flat/\mu = (\mu^{\mathsf{tgt}})^\flat$ and, as a particular case, $(\varrho^{\mathsf{src}})^\flat/\varrho = (\varrho^{\mathsf{tgt}})^\flat$.*
   - 3.4 *$\mu/(\mu^{\mathsf{src}})^\flat = \mu^\flat$ and, as a particular case, $\varrho/(\varrho^{\mathsf{src}})^\flat = \varrho$.*

▶ **Example 27.** Let $\vartheta : \lambda x.\mathbf{f}\, x \to \lambda x.\mathbf{g}\, x$. Then:

$$\begin{array}{rcccccl} & (\lambda x.(\lambda x.\mathbf{f}\, x)\, x)/(\lambda x.\vartheta\, x) & = & (\lambda x.((\lambda x.\mathbf{f}\, x)\, x)/(\vartheta\, x))^\flat & = & (\lambda x.(((\lambda x.\mathbf{f}\, x)/\vartheta)(x/x))^\flat)^\flat \\ = & (\lambda x.((\lambda x.\mathbf{g}\, x)\, x)^\flat)^\flat & = & (\lambda x.\mathbf{g}\, x)^\flat & = & \mathbf{g} \end{array}$$

Projection for Flat Rewrites. The projection operator from Def. 25 is extended to operate on flat rewrites. One may try to define ρ/σ using equations such as $(\rho_1 \mathbin{;} \rho_2)/\sigma = (\rho_1/\sigma) \mathbin{;} (\rho_2/(\sigma/\rho_1))$. However, it is not *a priori* clear that this recursive definition is well-founded[4]. This is why the following definition proceeds in three stages:

[4] Another way to prove well-foundedness is by interpretation, as done in [15, Example 6.5.43].

▶ **Definition 28** (Projection operator for flat rewrites). *We define:*

1. *projection of a flat multistep over a coinitial flat rewrite* $(\mu \mathbin{/^1} \rho)$*, by induction on* ρ*;*
2. *projection of a flat rewrite over a coinitial flat multistep* $(\rho \mathbin{/^2} \mu)$*, by induction on* ρ*; and*
3. *projection of a flat rewrite over a coinitial flat rewrite* $(\rho \mathbin{/^3} \sigma)$ *by induction on* σ*, as follows:*

$$\begin{array}{llll} \mu \mathbin{/^1} \nu \stackrel{\text{def}}{=} \mu/\nu & & \mu \mathbin{/^1} (\rho_1 \mathbin{;} \rho_2) \stackrel{\text{def}}{=} (\mu \mathbin{/^1} \rho_1) \mathbin{/^1} \rho_2 \\ \nu \mathbin{/^2} \mu \stackrel{\text{def}}{=} \nu/\mu & & (\rho_1 \mathbin{;} \rho_2) \mathbin{/^2} \mu \stackrel{\text{def}}{=} (\rho_1 \mathbin{/^2} \mu) \mathbin{;} (\rho_2 \mathbin{/^2} (\mu \mathbin{/^1} \rho_1)) \\ \rho \mathbin{/^3} \mu \stackrel{\text{def}}{=} \rho \mathbin{/^2} \mu & & \rho \mathbin{/^3} (\sigma_1 \mathbin{;} \sigma_2) \stackrel{\text{def}}{=} (\rho \mathbin{/^3} \sigma_1) \mathbin{/^3} \sigma_2 \end{array}$$

Note that $/^3$ generalizes $/^2$ and $/^1$ in the sense that $\mu \mathbin{/^1} \rho = \mu \mathbin{/^3} \rho$ and $\rho \mathbin{/^2} \mu = \rho \mathbin{/^3} \mu$. With these definitions, the key equation $(\rho_1 \mathbin{;} \rho_2) \mathbin{/^3} \sigma = (\rho_1 \mathbin{/^3} \sigma) \mathbin{;} (\rho_2 \mathbin{/^3} (\sigma \mathbin{/^3} \rho_1))$ can be shown to hold.

From this point on, we overload ρ/σ to stand for either of these projection operators. The key equation ensures that this abuse of notation is harmless. In the following, we mention some important properties of projection for flat rewrites. First, projection of a rewrite over a sequence, and of a sequence over a rewrite, obey the expected equations $\rho/(\sigma_1 \mathbin{;} \sigma_2) = (\rho/\sigma_1)/\sigma_2$ and $(\rho_1 \mathbin{;} \rho_2)/\sigma = (\rho_1/\sigma) \mathbin{;} (\rho_2/(\sigma/\rho_1))$. Second, flat permutation equivalence is a congruence with respect to projection: more precisely, if $\rho \sim \sigma$ then $\tau/\rho = \tau/\sigma$ and $\rho/\tau \sim \sigma/\tau$. Third, the projection of a rewrite over itself is always empty; specifically $\rho/\rho \sim (\rho^{\mathsf{tgt}})^\flat$. Finally, an important property is that $\rho \mathbin{;} (\sigma/\rho) \sim \sigma \mathbin{;} (\rho/\sigma)$, corresponding to a strong form of confluence. The proof of these properties is technical, by induction on the structure of the rewrites. We do not develop the full theory of projection for flat rewrites here for lack of space (see Section E in [2] for more details).

Projection for Arbitrary Rewrites. As a final step, the projection operator of Def. 28 may be extended to arbitrary rewrites by flattening first. The proof of Prop. 30 relies crucially on the properties of projection for flat rewrites and on Thm. 20; it may be found in Section G in [2].

▶ **Definition 29** (Projection operator for arbitrary rewrites). *Let* ρ, σ *be arbitrary coinitial rewrites. Their projection is defined as* $\rho /\!\!/ \sigma \stackrel{\text{def}}{=} \rho^\flat / \sigma^\flat$.

▶ **Proposition 30** (Properties of projection for arbitrary rewrites).

1. *Projection of a rewrite over a sequence and of a sequence over a rewrite obey the expected equations* $\rho /\!\!/ (\sigma_1 \mathbin{;} \sigma_2) = (\rho /\!\!/ \sigma_1) /\!\!/ \sigma_2$ *and* $(\rho_1 \mathbin{;} \rho_2) /\!\!/ \sigma = (\rho_1 /\!\!/ \sigma) \mathbin{;} (\rho_2 /\!\!/ (\sigma /\!\!/ \rho_1))$.
2. *Projection commutes with abstraction and application, that is:*
   - 2.1 $(\lambda x.\rho) /\!\!/ (\lambda x.\sigma) \approx \lambda x.(\rho /\!\!/ \sigma)$*, and more precisely* $(\lambda x.\rho) /\!\!/ (\lambda x.\sigma) \overset{\flat}{\leftarrow}{}^* \lambda x.(\rho /\!\!/ \sigma)$.
   - 2.2 *If* ρ_1, σ_1 *are coinitial and* ρ_2, σ_2 *are coinitial, then* $(\rho_1\, \rho_2) /\!\!/ (\sigma_1\, \sigma_2) \approx (\rho_1 /\!\!/ \sigma_1)\, (\rho_2 /\!\!/ \sigma_2)$*, and more precisely* $(\rho_1\, \rho_2) /\!\!/ (\sigma_1\, \sigma_2) \overset{\flat}{\leftarrow}{}^* (\rho_1 /\!\!/ \sigma_1)\, (\rho_2 /\!\!/ \sigma_2)$.
3. *The projection of a rewrite over itself is always empty,* $\rho /\!\!/ \rho \approx \rho^{\mathsf{tgt}}$.
4. *Permutation equivalence is a congruence with respect to projection, namely if* $\rho \approx \sigma$ *then* $\tau /\!\!/ \rho = \tau /\!\!/ \sigma$ *and* $\rho /\!\!/ \tau \approx \sigma /\!\!/ \tau$.
5. *The key equation* $\rho \mathbin{;} (\sigma /\!\!/ \rho) \approx \sigma \mathbin{;} (\rho /\!\!/ \sigma)$ *holds.*

Characterization of Permutation Equivalence in Terms of Projection. Finally, we are able to characterize permutation equivalence $\rho \approx \sigma$ as the condition that the projections $\rho /\!\!/ \sigma$ and $\sigma /\!\!/ \rho$ are both empty. Indeed:

▶ **Theorem 31** (Projection equivalence). *Let ρ, σ be arbitrary coinitial rewrites. Then $\rho \approx \sigma$ if and only if $\rho /\!\!/ \sigma \approx \sigma^{\mathsf{tgt}}$ and $\sigma /\!\!/ \rho \approx \rho^{\mathsf{tgt}}$.*

Proof. ($\Rightarrow$) Suppose that $\rho \approx \sigma$. Then, by Prop. 30, $\rho /\!\!/ \sigma \approx \sigma /\!\!/ \sigma \approx \sigma^{\mathsf{tgt}}$. Symmetrically, $\sigma /\!\!/ \rho \approx \rho^{\mathsf{tgt}}$. ($\Leftarrow$) Let $\rho /\!\!/ \sigma \approx \sigma^{\mathsf{tgt}}$ and $\sigma /\!\!/ \rho \approx \rho^{\mathsf{tgt}}$. Then, by Prop. 30, $\rho \approx \rho \,;\, \rho^{\mathsf{tgt}} \approx \rho \,;\, (\sigma /\!\!/ \rho) \approx \sigma \,;\, (\rho /\!\!/ \sigma) \approx \sigma \,;\, \sigma^{\mathsf{tgt}} \approx \sigma$. ◀

Since flattening and projection are computable, Thm. 20 and Thm. 31 together provide an **effective method to decide permutation equivalence** $\rho \approx \sigma$ for arbitrary rewrites. Indeed, to test whether $\rho /\!\!/ \sigma \approx \sigma^{\mathsf{tgt}}$, note by Thm. 20 that this is equivalent to testing whether $\rho /\!\!/ \sigma \sim (\sigma^{\mathsf{tgt}})^\flat$, so it suffices to check that $\rho /\!\!/ \sigma$ is *empty*, *i.e.* it contains no rule symbols. This is justified by the fact that if μ has no rule symbols and $\mu \sim \rho$, then ρ has no rule symbols (See Lem. 162 in [2]).

7 Related Work and Conclusions

As mentioned in the introduction, proof terms were introduced by van Oostrom and de Vrijer for first-order left-linear rewrite systems to study equivalence of reductions in [17] and [15, Chapter 9]. They are inspired in Rewriting Logic [13]. In the setting of HORs, Hilken [6] introduces rewrites for $\beta\eta$-reduction together with a notion of permutation equivalence for those rewrites. He does not study permutation equivalence for arbitrary HORs nor formulate notions of projection. Hilken does, however, justify his equations through a categorical semantics. We have already discussed Bruggink's work extensively [4, 3]. Another attempt at devising proof terms for HOR by the authors of the present paper is [1]. The latter uses a term assignment for a minimal modal logic called Logic of Proofs (LP), to model rewrites. LP is a refinement of S4 in which the modality $\Box A$ is refined to $[s]A$, where s is said to be a witness to the proof of A. The intuition is that terms and rewrites may be seen to belong to different stages of discourse; rewrites verse about terms. Terms are typed with simple types and rewrites are typed with a modal type $[s]A$ where the term s is the source term of the rewrite. However, the notion of substitution that is required for subject reduction is arguably ad-hoc. In particular, substitution of a rewrite $\rho : s \twoheadrightarrow s' : A$ for x in another rewrite $\sigma : t \twoheadrightarrow t' : A$ is *defined* as the composed rewrite $\rho\{x\backslash t\} \,;\, s'\{x\backslash\!\backslash\sigma\}$, where ρ is substituted for x in $\underline{t}$ followed by σ where s' is substituted for x.

Future work. It would be of interest to develop tools based on the work presented here for reasoning about computations in higher-order rewriting, as has recently been explored for first-order rewriting [9, 10]. One downside is that our rewrites cannot be treated as terms in a higher-order rewrite system. Indeed, rewrites are not defined modulo $\beta\eta$ (for good reason since an expression such as $(\lambda x.\rho)\,\sigma$ should not be subject to β reduction).

One problem that should be addressed is that of formulating *standardization* (see *e.g.* [15, Section 8.5]) using rewrites. This amounts to giving a procedure that reorders the steps of a rewrite ρ, yielding a rewrite ρ^* in which outermost steps are performed before innermost ones. Standardization finds canonical representatives of $\approx$-equivalence classes, in the sense that $\rho \approx \sigma$ if and only if $\rho^* = \sigma^*$. The flattening rewrite system of Section 5 is a first approximation to standardization, since $\rho \approx \sigma$ if and only if $\rho^\flat \sim \sigma^\flat$. In a preliminary version of this work, we proposed a procedure to compute canonical representatives of $\approx$-equivalence classes, based on the idea of repeatedly converting $\mu \,;\, \nu$ into $\mu' \,;\, \nu'$ whenever $\nu \Leftrightarrow \xi \,;\, \nu'$ and $\mu' \Leftrightarrow \mu \,;\, \xi$, an idea reminiscent of *greedy decompositions* [5]. Unfortunately, this procedure does not always terminate, due to the fact that rewrites may have infinitely long "unfoldings";

for instance, if $\varrho : \mathbf{c} \twoheadrightarrow \mathbf{c}$ and $\vartheta : \mathbf{f}(x) \twoheadrightarrow \mathbf{d}$ then $\vartheta(\mathbf{c}) : \mathbf{f}(\mathbf{c}) \twoheadrightarrow \mathbf{d}$ is equivalent to arbitrarily long rewrites of the form $\mathbf{f}(\varrho)\ \mathbf{;}\ \ldots\ \mathbf{;}\ \mathbf{f}(\varrho)\ \mathbf{;}\ \vartheta(\mathbf{c})$. A terminating procedure should probably rely on a measure based on the notion of *essential development* [16, Definition 11].

Another avenue to pursue is to characterize permutation equivalence via *labelling*. The application of a rewrite step leaves a witness in the term itself, manifested as a decoration (a label). These labels thus collect and record the history of a computation. By comparing them one can determine whether two computations are equivalent. Labelling equivalence for first-order rewriting is studied by van Oostrom and de Vrijer in [17] and [15, Chapter 9].

We have given semantics to rewrites via Higher-Order Rewriting Logic. A categorical semantics for a similar notion of rewrite and permutation equivalence was presented by Hirshowitz [7] (projection equivalence and flattening are not studied though). Our $s\{x \backslash\backslash \rho\}$ is called *left whiskering* and $\rho\{x \backslash s\}$ *right whiskering*, using the terminology of 2-category theory. These are then used to define $\rho\{x \backslash\backslash\backslash \sigma\}$. A precise relation between the two notions of rewrite should be investigated.

References

1. Pablo Barenbaum and Eduardo Bonelli. Rewrites as terms through justification logic. In *PPDP '20: 22nd International Symposium on Principles and Practice of Declarative Programming, Bologna, Italy, 9-10 September, 2020*, pages 11:1–11:13. ACM, 2020.
2. Pablo Barenbaum and Eduardo Bonelli. Reductions in higher-order rewriting and their equivalence. *CoRR*, 2022.
3. Sander Bruggink. Residuals in higher-order rewriting. In Robert Nieuwenhuis, editor, *Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings*, volume 2706 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2003.
4. Sander Bruggink. *Equivalence of reductions in higher-order rewriting*. PhD thesis, Utrecht University, 2008. Available from: `http://www.ti.inf.uni-due.de/publications/bruggink/thesis.pdf`.
5. P. Dehornoy, F. Digne, E. Godelle, D. Krammer, and J. Michel. *Foundations of Garside Theory*. EMS tracts in mathematics. European Mathematical Society, 2015. Available from: `https://books.google.com.ar/books?id=7ec_SGVzNhEC`.
6. Barney P. Hilken. Towards a proof theory of rewriting: The simply typed 2λ-calculus. *Theor. Comput. Sci.*, 170(1-2):407–444, 1996.
7. Tom Hirschowitz. Cartesian closed 2-categories and permutation equivalence in higher-order rewriting. *Log. Methods Comput. Sci.*, 9(3), 2013.
8. Jan Willem Klop. *Combinatory Reduction Systems*. PhD thesis, Utrecht University, 1980.
9. Christina Kohl and Aart Middeldorp. Protem: A proof term manipulator (system description). In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 31:1–31:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
10. Christina Kohl and Aart Middeldorp. Composing proof terms. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 337–353. Springer, 2019.
11. Jean-Jacques Lévy. *Réductions correctes et optimales dans le lambda-calcul*. PhD thesis, Université de Paris 7, 1978.
12. Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.
13. José Meseguer. Conditioned rewriting logic as a united model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.

14 Tobias Nipkow. Higher-order critical pairs. In *Proceedings 1991 Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 342–343. IEEE Computer Society, 1991.

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

16 Vincent van Oostrom. Normalisation in weakly orthogonal rewriting. In Paliath Narendran and Michaël Rusinowitch, editors, *Rewriting Techniques and Applications, 10th International Conference, RTA-99, Trento, Italy, July 2-4, 1999, Proceedings*, volume 1631 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 1999.

17 Vincent van Oostrom and Roel C. de Vrijer. Four equivalent equivalences of reductions. *Electron. Notes Theor. Comput. Sci.*, 70(6):21–61, 2002.

Proofs and Refutations for Intuitionistic and Second-Order Logic

Pablo Barenbaum
University of Buenos Aires, Argentina
National University of Quilmes (CONICET), Bernal, Argentina

Teodoro Freund
University of Buenos Aires, Argentina

Abstract

The λ^{PRK}-calculus is a typed λ-calculus that exploits the duality between the notions of proof and refutation to provide a computational interpretation for classical propositional logic. In this work, we extend λ^{PRK} to encompass classical second-order logic, by incorporating parametric polymorphism and existential types. The system is shown to enjoy good computational properties, such as type preservation, confluence, and strong normalization, which is established by means of a reducibility argument. We identify a syntactic restriction on proofs that characterizes exactly the intuitionistic fragment of second-order λ^{PRK}, and we study canonicity results.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Type theory

Keywords and phrases lambda-calculus, propositions-as-types, classical logic, proof normalization

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.9

Related Version *Extended Version*: `https://arxiv.org/abs/2210.07274` [5]

Funding This work was partially supported by project grant PUNQ 2247/22.

1 Introduction

Constructivism in logic is closely related with the notion of *algorithm* in computer science. The reason is that a constructive proof of existence of a mathematical object fulfilling certain properties should provide an *effective construction* of such an object. For example, a constructive proof of $\forall x \in \mathbb{N}. \exists y \in \mathbb{N}. P(x, y)$ may be understood as an algorithm that takes as input a natural number x and produces as output a natural number y that verifies $P(x, y)$. The close relationship that exists between *proofs* and *computer programs*, and between *logical propositions* and *program specifications* (or *types*), can be taken to its maximum consequences in the form of the **propositions-as-types correspondence**.

This correspondence has given rise to a broad and active area of research, guided by the principle that *each proof-theoretical notion has a computational counterpart and vice-versa*. These ideas allow logic and computer science to feed back on each other, and they have been extended to such settings as *first-order logic* [15, 30, 8], *second-order logic* [21, 42], *linear logic* [22], *modal logic* [6, 14] and *classical logic* [23, 10, 3, 37]. The question of what kind of computational system would constitute a reasonable counterpart for **classical logic**, from the point of view of the propositions-as-types correspondence, is far from being definitely settled. This work is part of the quest for a satisfactory answer to this problem.

The proofs and refutations calculus (λ^{PRK}). Until the late 1980s, it was widely thought that it was not possible to extend the propositions-as-types correspondence to encompass classical logic. This view changed when Griffin [23] remarked that the classical principle of double negation elimination ($\neg\neg A \to A$) can be understood as the typing rule for a control

operator $\mathcal{C}$, closely related to Felleisen's $\mathcal{C}$ operator [19] and to SCHEME's `call/cc`. Since then, many other calculi for classical logic have been proposed. Significant examples are Parigot's $\lambda\mu$ [37], Barbanera and Berardi's calculus [3], and Curien and Herbelin's $\bar{\lambda}\mu\tilde{\mu}$ calculus [10].

The starting point of this paper is the logical system PRK, introduced recently by the authors [4] and extending Nelson's constructive negation [34]. In PRK, propositions become classified along two dimensions: their *sign*, which may be positive or negative, and their *strength*, which may be strong or weak. This results into four possible *modes*:

	Positive	Negative
Strong	A^+	A^-
Weak	$A^\oplus$	$A^\ominus$

Positive ($A^+/A^\oplus$) and negative ($A^-/A^\ominus$) propositions correspond to affirmations and denials. Strong (A^+/A^-) and weak ($A^\oplus/A^\ominus$) propositions impose restrictions on the shape of canonical proofs: a canonical proof of a strong affirmation (A^+) must always be constructed with an introduction rule for the corresponding logical connective, whereas a canonical proof of a weak affirmation ($A^\oplus$) must always proceed by *reductio ad absurdum*, by assuming the weak denial $A^\ominus$ and proving the strong affirmation A^+.

We summarize some important characteristics of PRK. First, PRK is a **refinement** of classical logic: $A_1, \ldots, A_n \vdash B$ holds in classical propositional logic if and only if $A_1^\oplus, \ldots, A_n^\oplus \vdash B^\oplus$ holds in PRK. In fact PRK is "finer" than classical logic: for example, the law of excluded middle holds *weakly*, *i.e.* $(A \vee \neg A)^\oplus$ is valid in PRK, whereas it does not hold *strongly*, *i.e.* $(A \vee \neg A)^+$ is not valid (in general) in PRK. Second, the λ^{PRK}-calculus, which results from assigning proof terms to PRK proofs and endowing it with rewrite rules, turns out to be **confluent** and **strongly normalizing**, besides enjoying **subject reduction**. Third, as a result, PRK enjoys **canonicity**: a proof of a sequent $\vdash P$ without assumptions can always be normalized to a *canonical* proof, headed by an introduction rule.

Contributions and structure of this paper. The PRK logical system of [4] only treats three propositional connectives: conjunction, disjunction, and negation.

- In Section 2, we **extend the $\lambda^{\mathbf{PRK}}$ calculus** to propositional second-order logic. We incorporate **second-order universal and existential quantification**, as well as two propositional connectives, implication and co-implication. The system is shown to **refine classical second-order logic**, and to enjoy good computational properties: **subject reduction** and **confluence**. This extension increases the expressivity of the system, allowing to encode inductive datatypes such as natural numbers, lists, and trees.
- In Section 3, we study **Böhm–Berarducci encodings**, that is, we study how the logical connectives of second-order λ^{PRK} may be encoded in terms of universal quantification and implication only ($\{\forall, \to\}$). The encoding turns out to be only partially satisfactory: it simulates proof normalization for an introduction rule followed by an elimination rule in the *positive* case but, unfortunately, not in the negative case.
- In Section 4 we prove **strong normalization** for the second-order λ^{PRK}-calculus. This is the most technically challenging part of the work. In [4], normalization of the propositional fragment of λ^{PRK} is attained by means of a translation to System F with non-strictly positive recursion. This technique does not carry over to the second-order case. To prove strong normalization, we use a variant of Girard's technique of reducibility candidates and, in particular, we resort to a non-trivial adaptation of Mendler's proof of strong normalization for System F with non-strictly positive recursion [31].

- In Section 5, we define a subsystem of second-order λ^{PRK}, called λ^{PRJ}, by imposing a syntactic restriction on terms. We show that λ^{PRJ} **refines second-order intuitionistic logic**, in the sense that λ^{PRJ} is a conservative extension of second-order intuitionistic logic and, conversely, second-order intuitionistic logic can be embedded in λ^{PRJ}.
- In Section 6 we formulate **canonicity** results for λ^{PRK}. In particular, we strengthen the canonicity results of [4] to show that an explicit witness can be extracted from a proof of P.
- Finally, in Section 7 we conclude and we discuss some related and future work.

2 Second-Order Proofs and Refutations

In this section we define a second-order extension of λ^{PRK}, including its syntax, typing rules, and rewriting rules. We show that the system enjoys **subject reduction**, it is **confluent**, and it **refines classical second-order logic** (Thm. 3).

Syntax of types. We assume given a denumerable set of *type variables* $\alpha, \beta, \gamma, \ldots$. The sets of *pure types* $(A, B, \ldots)$ and *types* $(P, Q, \ldots)$ are given by:

$$A ::= \alpha \mid A \wedge A \mid A \vee A \mid A \to A \mid A \ltimes A \mid \neg A \mid \forall \alpha.\, A \mid \exists \alpha.\, A \qquad P ::= A^{+} \mid A^{-} \mid A^{\oplus} \mid A^{\ominus}$$

where $A \ltimes B$ represents *co-implication*, the dual connective to implication, to be understood (roughly) as $\neg A \wedge B$. The four modes represent strong affirmation (A^{+}), strong denial (A^{-}), weak affirmation $(A^{\oplus})$, and weak denial $(A^{\ominus})$. Note that *modes* $({}^{+}, {}^{-}, {}^{\oplus}, {}^{\ominus})$ can only decorate the root of a type, *i.e.* they cannot be nested.

Sometimes one may be interested in *fragments* of the system. For instance, the λ^{PRK}-calculus of [4] corresponds to the $\{\wedge, \vee, \neg\}$ fragment. In this paper we are usually interested in the full $\{\wedge, \vee, \to, \ltimes, \neg, \forall, \exists\}$ fragment. As long as there is little danger of confusion we still speak of λ^{PRK} without further qualifications.

Syntax of terms. Terms of λ^{PRK} are given by the following grammar. The letter i ranges over $\{1, 2\}$. Some terms are decorated with either "$^{+}$" or "$^{-}$". In the grammar we write "$^{\pm}$" to stand for either "$^{+}$" or "$^{-}$".

$t, s, \ldots ::=$	x^{P}	variable	$\mid$	$t \bowtie_{P} s$	absurdity
$\mid$	$\bigcirc^{\pm}_{(x:P)}.\, t$	$^{\oplus}/^{\ominus}$ introduction	$\mid$	$t \bullet^{\pm} s$	$^{\oplus}/^{\ominus}$ elimination
$\mid$	$\langle t, s \rangle^{\pm}$	$\wedge^{+}/\vee^{-}$ introduction	$\mid$	$\pi_i^{\pm}(t)$	$\wedge^{+}/\vee^{-}$ elimination
$\mid$	$\mathsf{in}_i^{\pm}(t)$	$\vee^{+}/\wedge^{-}$ introduction	$\mid$	$\delta^{\pm} t\, [_{x:P}.s][_{y:Q}.u]$	$\vee^{+}/\wedge^{-}$ elimination
$\mid$	$\lambda^{\pm}_{(x:P)}.t$	$\to {}^{+}/\ltimes^{-}$ introduction	$\mid$	$t@^{\pm} s$	$\to {}^{+}/\ltimes^{-}$ elimination
$\mid$	$(t \,;^{\pm} s)$	$\ltimes^{+}/\to {}^{-}$ introduction	$\mid$	$\varrho^{\pm} t[_{x:P;y:Q}.s]$	$\ltimes^{+}/\to {}^{-}$ elimination
$\mid$	$\mathsf{N}^{\pm} t$	$\neg^{+}/\neg^{-}$ introduction	$\mid$	$\mathsf{M}^{\pm} t$	$\neg^{+}/\neg^{-}$ elimination
$\mid$	$\lambda^{\pm}_{\alpha}.\, t$	$\forall^{+}/\exists^{-}$ introduction	$\mid$	$t@^{\pm} A$	$\forall^{+}/\exists^{-}$ elimination
$\mid$	$\langle A, t \rangle^{\pm}$	$\exists^{+}/\forall^{-}$ introduction	$\mid$	$\nabla^{\pm} t\, [_{(\alpha, x:P)}.s]$	$\exists^{+}/\forall^{-}$ elimination

The notions of free and bound occurrences of variables are defined as expected, with the typographical convention that subscripted variable occurrences are binding. Terms are considered up to α-renaming of bound variables. We write $\mathsf{fv}(t)$ for the set of free variables of t and $\mathsf{ftv}(t)$ for the set of type variables occurring free in t. By $t\{x := s\}$ we mean the capture-avoiding substitution of the free occurrences of x in t by s.

Variables are annotated with their type, which we usually omit. Sometimes we also omit the types of bound variables if they are clear from the context, as well as the name of unused bound variables, writing "_" instead. For example, if $x \notin \mathsf{fv}(t)$ we may write $\bigcirc^{+}_{\_}.\, t$ rather

than $\bigcirc^{+}_{(x:A^{-})}.\,t$. Application-like operators are assumed to be left-associative; for example, $t@^{+}s \bullet^{+} u@^{+}A$ stands for $((t@^{+}s) \bullet^{+} u)@^{+}A$. In a term of the form $\bigcirc^{\pm}_{(x:P)}.\,t$, the variable x is called the *counterfactual*, and more specifically a *negative counterfactual* in a term of the form $\bigcirc^{+}_{(x:A^{\ominus})}.\,t$. In a term of the form $t \bullet^{\pm} s$, we call t the *subject* and s the *argument*. We write $\mathtt{C}$ for arbitrary term *contexts*, *i.e.* terms with a single free occurrence of a distinguished variable $\Box$ called a *hole*, and $\mathtt{C}\langle t\rangle$ for the variable-capturing substitution of the hole of $\mathtt{C}$ by t.

The λ^{PRK} type system. A *typing context*, ranged over by $\Gamma, \Delta, \ldots$, is a finite assignment of variables to types, written as $x_1 : P_1, \ldots, x_n : P_n$. We write $\mathsf{dom}(\Gamma)$ for the *domain* of Γ, *i.e.* the finite set $\{x_1, \ldots, x_n\}$. Typing judgments in λ^{PRK} are of the form $\Gamma \vdash t : P$, meaning that t has type P under the context Γ.

We write $\Gamma \vdash_{\mathrm{PRK}} t : P$ if the typing judgment $\Gamma \vdash t : P$ is derivable in λ^{PRK}. When we wish to emphasize the logical point of view, we may write sequents as $P_1, \ldots, P_n \vdash Q$, and we may write $P_1, \ldots, P_n \vdash_{\mathrm{PRK}} Q$ to mean that there exists a term t such that $x_1 : P_1, \ldots, x_n : P_n \vdash_{\mathrm{PRK}} t : Q$. Derivable judgments are given inductively by the typing rules below.

Basic rules

$$\frac{}{\Gamma, x : P \vdash x : P}\,\textsc{Ax} \qquad \frac{\Gamma \vdash t : A^{+} \qquad \Gamma \vdash s : A^{-}}{\Gamma \vdash t \bowtie_{P} s : P}\,\textsc{Abs} \qquad \frac{\Gamma, x : A^{\ominus} \vdash t : A^{+}}{\Gamma \vdash \bigcirc^{+}_{x:A^{\ominus}}.\,t : A^{\oplus}}\,\mathrm{I}^{+}_{\circ}$$

$$\frac{\Gamma, x : A^{\oplus} \vdash t : A^{-}}{\Gamma \vdash \bigcirc^{+}_{x:A^{\oplus}}.\,t : A^{\ominus}}\,\mathrm{I}^{-}_{\circ} \qquad \frac{\Gamma \vdash t : A^{\oplus} \quad \Gamma \vdash s : A^{\ominus}}{\Gamma \vdash t \bullet^{+} s : A^{+}}\,\mathrm{E}^{+}_{\circ} \qquad \frac{\Gamma \vdash t : A^{\ominus} \quad \Gamma \vdash s : A^{\oplus}}{\Gamma \vdash t \bullet^{-} s : A^{-}}\,\mathrm{E}^{-}_{\circ}$$

Conjunction and disjunction

$$\frac{\Gamma \vdash t : A^{\oplus} \qquad \Gamma \vdash s : B^{\oplus}}{\Gamma \vdash \langle t, s\rangle^{+} : (A \wedge B)^{+}}\,\mathrm{I}^{+}_{\wedge} \qquad \frac{\Gamma \vdash t : A^{\ominus} \qquad \Gamma \vdash s : B^{\ominus}}{\Gamma \vdash \langle t, s\rangle^{-} : (A \vee B)^{-}}\,\mathrm{I}^{-}_{\vee} \qquad \frac{\Gamma \vdash t : (A_1 \wedge A_2)^{+}}{\Gamma \vdash \pi_i^{+}(t) : {A_i}^{\oplus}}\,\mathrm{E}^{+}_{\wedge i}$$

$$\frac{\Gamma \vdash t : (A_1 \vee A_2)^{-}}{\Gamma \vdash \pi_i^{-}(t) : {A_i}^{\ominus}}\,\mathrm{E}^{-}_{\vee i} \qquad \frac{\Gamma \vdash t : {A_i}^{\oplus} \qquad i \in \{1, 2\}}{\Gamma \vdash \mathsf{in}_i^{+}(t) : (A_1 \vee A_2)^{+}}\,\mathrm{I}^{+}_{\vee i} \qquad \frac{\Gamma \vdash t : {A_i}^{\ominus} \qquad i \in \{1, 2\}}{\Gamma \vdash \mathsf{in}_i^{-}(t) : (A_1 \wedge A_2)^{-}}\,\mathrm{I}^{-}_{\wedge i}$$

$$\frac{\Gamma \vdash t : (A \vee B)^{+} \qquad \Gamma, x : A^{\oplus} \vdash s : P \qquad \Gamma, y : B^{\oplus} \vdash u : P}{\Gamma \vdash \delta^{+} t\,[_{x:A^{\oplus}}.s][_{y:B^{\oplus}}.u] : P}\,\mathrm{E}^{+}_{\vee}$$

$$\frac{\Gamma \vdash t : (A \wedge B)^{-} \qquad \Gamma, x : A^{\ominus} \vdash s : P \qquad \Gamma, y : B^{\ominus} \vdash u : P}{\Gamma \vdash \delta^{-} t\,[_{x:A^{\ominus}}.s][_{y:B^{\ominus}}.u] : P}\,\mathrm{E}^{-}_{\wedge}$$

Implication and co-implication

$$\frac{\Gamma, x : A^{\oplus} \vdash t : B^{\oplus}}{\Gamma \vdash \lambda^{+}_{x:A^{\oplus}}.t : (A \to B)^{+}}\,\mathrm{I}^{+}_{\to} \qquad \frac{\Gamma, x : A^{\ominus} \vdash t : B^{\ominus}}{\Gamma \vdash \lambda^{-}_{x:A^{\ominus}}.t : (A \ltimes B)^{-}}\,\mathrm{I}^{-}_{\ltimes} \qquad \frac{\Gamma \vdash t : (A \to B)^{+} \quad \Gamma \vdash s : A^{\oplus}}{\Gamma \vdash t@^{+}s : B^{\oplus}}\,\mathrm{E}^{+}_{\to}$$

$$\frac{\Gamma \vdash t : (A \ltimes B)^{-} \quad \Gamma \vdash s : A^{\ominus}}{\Gamma \vdash t@^{-}s : B^{\ominus}}\,\mathrm{E}^{-}_{\ltimes} \qquad \frac{\Gamma \vdash t : A^{\ominus} \qquad \Gamma \vdash s : B^{\oplus}}{\Gamma \vdash (t\,;^{+}s) : (A \ltimes B)^{+}}\,\mathrm{I}^{+}_{\ltimes} \qquad \frac{\Gamma \vdash t : A^{\oplus} \quad \Gamma \vdash s : B^{\ominus}}{\Gamma \vdash (t\,;^{-}s) : (A \to B)^{-}}\,\mathrm{I}^{-}_{\to}$$

$$\frac{\Gamma \vdash t : (A \ltimes B)^{+} \quad \Gamma, x : A^{\ominus}, y : B^{\oplus} \vdash s : P}{\Gamma \vdash \varrho^{+} t[_{x:A^{\ominus}; y:B^{\oplus}}.s] : P}\,\mathrm{E}^{+}_{\ltimes} \qquad \frac{\Gamma \vdash t : (A \to B)^{-} \quad \Gamma, x : A^{\oplus}, y : B^{\ominus} \vdash s : P}{\Gamma \vdash \varrho^{-} t[_{x:A^{\oplus}; y:B^{\ominus}}.s] : P}\,\mathrm{E}^{-}_{\to}$$

Negation

$$\frac{\Gamma \vdash t : A^{\ominus}}{\Gamma \vdash \mathsf{N}^{+} t : (\neg A)^{+}}\,\mathrm{I}^{+}_{\neg} \qquad \frac{\Gamma \vdash t : A^{\oplus}}{\Gamma \vdash \mathsf{N}^{-} t : (\neg A)^{-}}\,\mathrm{I}^{-}_{\neg} \qquad \frac{\Gamma \vdash t : (\neg A)^{+}}{\Gamma \vdash \mathsf{M}^{+} t : A^{\ominus}}\,\mathrm{E}^{+}_{\neg} \qquad \frac{\Gamma \vdash t : (\neg A)^{-}}{\Gamma \vdash \mathsf{M}^{-} t : A^{\oplus}}\,\mathrm{E}^{-}_{\neg}$$

Second-order quantification

$$\frac{\Gamma \vdash t : A^{\oplus} \qquad \alpha \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \lambda^{+}_{\alpha}.\, t : (\forall \alpha.\, A)^{+}}\,\mathrm{I}^{+}_{\forall} \qquad \frac{\Gamma \vdash t : A^{\ominus} \qquad \alpha \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \lambda^{-}_{\alpha}.\, t : (\exists \alpha.\, A)^{-}}\,\mathrm{I}^{-}_{\exists} \qquad \frac{\Gamma \vdash t : (\forall \alpha.\, B)^{+}}{\Gamma \vdash t @^{+} A : B^{\oplus}\{\alpha := A\}}\,\mathrm{E}^{+}_{\forall}$$

$$\frac{\Gamma \vdash t : (\exists \alpha.\, B)^{-}}{\Gamma \vdash t @^{-} A : B^{\ominus}\{\alpha := A\}}\,\mathrm{E}^{-}_{\exists} \qquad \frac{\Gamma \vdash t : B^{\oplus}\{\alpha := A\}}{\Gamma \vdash \langle A, t \rangle^{+} : (\exists \alpha.\, B)^{+}}\,\mathrm{I}^{+}_{\exists} \qquad \frac{\Gamma \vdash t : B^{\ominus}\{\alpha := A\}}{\Gamma \vdash \langle A, t \rangle^{-} : (\forall \alpha.\, B)^{-}}\,\mathrm{I}^{-}_{\forall}$$

$$\frac{\Gamma \vdash t : (\exists \alpha.\, A)^{+} \qquad \Gamma, x : A^{\oplus} \vdash s : P \qquad \alpha \notin \mathsf{ftv}(\Gamma, P)}{\Gamma \vdash \nabla^{+} t [_{(\alpha, x : A^{\oplus})}.s] : P}\,\mathrm{E}^{+}_{\exists}$$

$$\frac{\Gamma \vdash t : (\forall \alpha.\, A)^{-} \qquad \Gamma, x : A^{\ominus} \vdash s : P \qquad \alpha \notin \mathsf{ftv}(\Gamma, P)}{\Gamma \vdash \nabla^{-} t [_{(\alpha, x : A^{\ominus})}.s] : P}\,\mathrm{E}^{-}_{\forall}$$

The typing rules may be informally explained as follows. Ax is the standard axiom. The *absurdity rule* (Abs) allows to derive any conclusion from a strong proof and a strong refutation of A. Introduction and elimination rules for weak affirmation and denial ($\mathrm{I}^{\pm}_{\circ}$, $\mathrm{E}^{\pm}_{\circ}$) follow the principle that a weak affirmation $A^{\oplus}$ behaves like an implication "$A^{\ominus} \to A^{+}$". Indeed, $\mathrm{I}^{+}_{\circ}$ and $\mathrm{E}^{+}_{\circ}$ have the same structure as the introduction and the elimination rule for an implication "$A^{\ominus} \to A^{+}$", where $\bigcirc^{+}_{x}.\, t$ and $t \bullet^{+} s$ are akin to λ-abstraction and application. The intuition behind this is that $A^{\oplus}$ is the type of weak proofs of a proposition A, where a weak proof proceeds by *reductio ad absurdum*, assuming a weak refutation ($A^{\ominus}$), and providing a *strong* proof (A^{+}). Dually, a weak denial $A^{\ominus}$ behaves like "$A^{\oplus} \to A^{-}$".

The remaining rules are introduction and elimination rules for positive and negative strong connectives. These rules come in dual pairs: *for each rule for a connective with positive sign there is a symmetric rule for the dual connective with negative sign.* For example, the introduction rule for positive conjunction ($\mathrm{I}^{+}_{\wedge}$) states that to strongly prove $A \wedge B$ it suffices to weakly prove A and weakly prove B. Dually, the introduction rule for negative disjunction ($\mathrm{I}^{-}_{\vee}$) states that to strongly refute $A \vee B$ it suffices to weakly refute A and weakly refute B.

The introduction and elimination rules for most logical connectives ($\wedge$, $\vee$, $\to$, $\ltimes$, $\forall$, $\exists$) are mechanically derived from the standard natural deduction rules following this methodology, taking in account that the dual pairs of connectives are $(\wedge, \vee)$, $(\to, \ltimes)$, and $(\forall, \exists)$. In general, *introduction rules have weak premises and strong conclusions*, whereas *elimination rules have strong premises and weak conclusions.*

The typing rules for conjunction ($\mathrm{I}^{+}_{\wedge}, \mathrm{E}^{+}_{\wedge i}, \mathrm{I}^{-}_{\wedge i}, \mathrm{E}^{-}_{\wedge}$), disjunction ($\mathrm{I}^{+}_{\vee i}, \mathrm{E}^{+}_{\vee}, \mathrm{I}^{-}_{\vee}, \mathrm{E}^{-}_{\vee i}$), positive implication ($\mathrm{I}^{+}_{\to}, \mathrm{E}^{+}_{\to}$), negative co-implication ($\mathrm{I}^{-}_{\ltimes}, \mathrm{E}^{-}_{\ltimes}$), universal ($\mathrm{I}^{+}_{\forall}, \mathrm{E}^{+}_{\forall}, \mathrm{I}^{-}_{\forall}, \mathrm{E}^{-}_{\forall}$) and existential quantification ($\mathrm{I}^{+}_{\exists}, \mathrm{E}^{+}_{\exists}, \mathrm{I}^{-}_{\exists}, \mathrm{E}^{-}_{\exists}$) are typical, so for instance $\langle t, s \rangle^{\pm}$ forms a pair, $\pi^{\pm}_{i}(t)$ is the i-th projection, $\mathsf{in}^{\pm}_{i}(t)$ is the i-th injection into a disjoint union type, $\delta^{\pm} t\, [_{x}.s][_{y}.u]$ is a pattern matching construct, and so on. The rules differ from usual typed λ-calculi only in the signs and strengths that decorate premises and conclusions.

The typing rules for positive co-implication ($\mathrm{I}^{+}_{\ltimes}, \mathrm{E}^{+}_{\ltimes}$), sometimes called *subtraction* [9], and negative implication ($\mathrm{I}^{-}_{\to}, \mathrm{E}^{-}_{\to}$) follow the rough interpretation of $A \ltimes B$ as $\neg A \wedge B$, so a strong proof of a co-implication $A \ltimes B$ is given by a pair $(t \,;^{+} s)$ comprising a weak refutation of A and a weak proof of B. Dually, a strong refutation of $A \to B$ is given by a pair $(t \,;^{-} s)$ comprising a weak proof of A and a weak refutation of B. The eliminators $\varrho^{\pm} t[_{x;y}.s]$ are presented as generalized elimination rules, in multiplicative style.

The typing rules for negation $(\mathrm{I}^+_\neg, \mathrm{E}^+_\neg, \mathrm{I}^-_\neg, \mathrm{E}^-_\neg)$ express that to strongly prove $\neg A$ is the same as to weakly refute A, and dually for strong refutations of $\neg A$.

▶ **Example 1.** Let $\top \stackrel{\text{def}}{=} \forall\alpha.\,(\alpha \to \alpha)$ and $\bot \stackrel{\text{def}}{=} \forall\alpha.\,\alpha$. Recall from [4] that the weak non-contradiction principle $\Gamma \vdash (A \wedge \neg A)^\ominus$ holds in λ^{PRK}. Then $\vdash \top^\oplus$ and $\bot^\oplus \vdash A^\oplus$ hold, where A stands for any pure type:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\top^\ominus, (\alpha\to\alpha)^\ominus, \alpha^\oplus \vdash \alpha^\oplus}\,\mathrm{Ax}}{\top^\ominus, (\alpha\to\alpha)^\ominus \vdash (\alpha\to\alpha)^+}\,\mathrm{I}^+_\to}{\top^\ominus \vdash (\alpha\to\alpha)^\oplus}\,\mathrm{I}^+_\circ}{\top^\ominus \vdash \top^+}\,\mathrm{I}^+_\forall}{\vdash \top^\oplus}\,\mathrm{I}^+_\circ$$

$$\dfrac{\dfrac{}{\bot^\oplus \vdash \bot^\oplus}\,\mathrm{Ax} \qquad \dfrac{\dfrac{\dfrac{\text{(By weak non-contradiction.)}}{\bot^\oplus, \bot^\oplus \vdash (B \wedge \neg B)^\ominus}\,\mathrm{I}^-_\forall}{\bot^\oplus, \bot^\oplus \vdash \bot^-}}{\bot^\oplus \vdash \bot^\ominus}\,\mathrm{I}^-_\circ}{\bot^\oplus \vdash A^\oplus}\,\mathrm{Abs}'$$

The λ^{PRK}-calculus. The opposite type $P^\sim$ of a given type P is defined by flipping the sign, *i.e.* $(A^\oplus)^\sim \stackrel{\text{def}}{=} A^\ominus$; $(A^\ominus)^\sim \stackrel{\text{def}}{=} A^\oplus$; $(A^+)^\sim \stackrel{\text{def}}{=} A^-$; and $(A^-)^\sim \stackrel{\text{def}}{=} A^+$. If $\Gamma \vdash_{\mathrm{PRK}} t : P$ and $\Gamma \vdash_{\mathrm{PRK}} s : P^\sim$ then a term $t \bowtie_Q s$ may be constructed such that $\Gamma \vdash_{\mathrm{PRK}} t \bowtie_Q s : Q$, as follows:

$$t \bowtie_Q s \stackrel{\text{def}}{=} t \bowtie_Q s \quad \text{if } P = A^+ \qquad\qquad t \bowtie_Q s \stackrel{\text{def}}{=} s \bowtie_Q t \quad \text{if } P = A^-$$
$$t \bowtie_Q s \stackrel{\text{def}}{=} (t \bullet^+ s) \bowtie_Q (s \bullet^- t) \quad \text{if } P = A^\oplus \qquad t \bowtie_Q s \stackrel{\text{def}}{=} (s \bullet^+ t) \bowtie_Q (t \bullet^- s) \quad \text{if } P = A^\ominus$$

We endow typable PRK terms with a notion of reduction, defining the **λ^{PRK}-calculus** by a binary rewriting relation $\to$ on typable PRK terms, given by the rewriting rules below, and closed by compatibility under arbitrary contexts. Rules are presented following the convention that, if many occurrences of "$\pm$" appear in the same expression, they are all supposed to stand for the same sign:

$$(\bigcirc^\pm_x.\,t)\bullet^\pm s \xrightarrow{\beta^+_\circ\,/\,\beta^-_\circ} t\{x := s\} \qquad \pi^\pm_i(\langle t_1, t_2\rangle^\pm) \xrightarrow{\beta^+_\wedge\,/\,\beta^-_\vee} t_i$$
$$(\lambda^\pm_x.t)@^\pm s \xrightarrow{\beta^+_\to\,/\,\beta^-_\ltimes} t\{x := s\} \qquad \mathsf{M}^\pm(\mathsf{N}^\pm t) \xrightarrow{\beta^+_\neg\,/\,\beta^-_\neg} t$$
$$(\lambda^\pm_\alpha.\,t)@^\pm A \xrightarrow{\beta^+_\forall\,/\,\beta^-_\exists} t\{\alpha := A\} \qquad \langle t_1, t_2\rangle^+ \bowtie \mathsf{in}^-_i(s) \xrightarrow{\bowtie_\wedge} t_i \bowtie s$$
$$\lambda^+_x.t \bowtie (s\,;^- u) \xrightarrow{\bowtie_\to} t\{x := s\} \bowtie u \qquad (\mathsf{N}^+ t) \bowtie (\mathsf{N}^- s) \xrightarrow{\bowtie_\neg} t \bowtie s$$
$$(\lambda^+_\alpha.\,t) \bowtie \langle A, s\rangle^- \xrightarrow{\bowtie_\forall} t\{\alpha := A\} \bowtie s$$

$$\delta^\pm(\mathsf{in}^\pm_i(t))\,[_x.s_1][_x.s_2] \xrightarrow{\beta^+_\vee\,/\,\beta^-_\wedge} s_i\{x := t\}$$
$$\varrho^\pm(t\,;^\pm s)[_{x;y}.u] \xrightarrow{\beta^+_\ltimes\,/\,\beta^-_\to} u\{x := t\}\{y := s\}$$
$$\nabla^\pm\,\langle A, t\rangle^\pm[_{(\alpha,x)}.s] \xrightarrow{\beta^+_\exists\,/\,\beta^-_\forall} s\{\alpha := A\}\{x := t\}$$
$$\mathsf{in}^+_i(t) \bowtie \langle s_1, s_2\rangle^- \xrightarrow{\bowtie_\vee} t \bowtie s_i$$
$$(t\,;^+ s) \bowtie \lambda^-_x.u \xrightarrow{\bowtie_\ltimes} s \bowtie u\{x := t\}$$
$$\langle A, t\rangle^+ \bowtie (\lambda^-_\alpha.\,s) \xrightarrow{\bowtie_\exists} t \bowtie s\{\alpha := A\}$$

The λ^{PRK}-calculus has two kinds of rules: "β" rules, akin to proof normalization rules in natural deduction, and "$\bowtie$" rules, akin to cut elimination rules in sequent calculus. The $\beta^+_\circ\,/\,\beta^-_\circ$ rules are exactly like the standard β-rule of the λ-calculus, with the difference that the abstraction $\bigcirc^+_{x:A^\ominus}.\,t$ is not an introduction of an implication "$A^\ominus \to A^+$" but rather the introduction of a weak affirmation $A^\oplus$. The $\beta^+_\to\,/\,\beta^-_\ltimes$ rules also describe a similar behavior, where $\lambda^+_{x:A^\oplus}.t$ is of type $(A \to B)^\oplus$. The remaining β rules are straightforward, encoding projection $(\beta^+_\wedge\,/\,\beta^-_\vee)$, pattern matching $(\beta^+_\vee\,/\,\beta^-_\wedge)$, etcetera.

The $\bowtie$ rules simplify an absurdity $(t \bowtie s)$ as much as possible, but they are never able to get rid of the absurdity. Indeed, note that the right-hand side of all the $\bowtie$ rules include the generalized absurdity operator $(\bowtie)$, which is in turn defined in terms of the absurdity operator

($\bowtie$). Recall, for example, that if $t : A^{\oplus}$ and $s : A^{\ominus}$ then $t \bowtie s = (t \bullet^{+} s) \bowtie (s \bullet^{-} t)$. This means that an instance of the absurdity is a proof which provides *no relevant information*. For example, if $P = (B \wedge C)^{+}$ then $\pi_1^{+}(t \bowtie_P s)$ is a well-formed term of type $B^{\oplus}$; but the argument of the projection is a term $t \bowtie_P s$ which may never become a pair $\langle p, q\rangle^{+}$. This means that the normal form will be a "stuck" term of the form $\pi_1^{+}(t' \bowtie_P s')$.

▶ **Example 2** (Reduction in λ^{PRK}). Let $\Gamma \vdash_{\mathrm{PRK}} t : A^{\oplus}$ and $\Gamma \vdash_{\mathrm{PRK}} s : A^{\ominus}$. Then:

$$
\begin{array}{ll}
 & (\lambda^{+}_{\alpha}.\, \bigcirc^{+}_{\_:(\alpha\to\alpha)^{\ominus}}.\, \lambda^{+}_{x:\alpha^{\oplus}}.x) \bowtie \langle A, \bigcirc^{-}_{\_:(\alpha\to\alpha)^{\oplus}}.\, (t\,;^{-}s)\rangle^{-} \\
\xrightarrow{\bowtie_{\forall}} & (\bigcirc^{+}_{\_:(A\to A)^{\ominus}}.\, \lambda^{+}_{x:A^{\oplus}}.x) \bowtie (\bigcirc^{-}_{\_:(A\to A)^{\oplus}}.\, (t\,;^{-}s)) \\
= & ((\bigcirc^{+}_{\_:(A\to A)^{\ominus}}.\, \lambda^{+}_{x:A^{\oplus}}.x) \bullet^{+} (\bigcirc^{-}_{\_:(A\to A)^{\oplus}}.\, (t\,;^{-}s))) \bowtie \\
 & ((\bigcirc^{-}_{\_:(A\to A)^{\oplus}}.\, (t\,;^{-}s)) \bullet^{-} (\bigcirc^{+}_{\_:(A\to A)^{\ominus}}.\, \lambda^{+}_{x:A^{\oplus}}.x)) \\
\xrightarrow{\beta^{\pm}_{\circ}}{}^{*} & (\lambda^{+}_{x:A^{\oplus}}.x) \bowtie (t\,;^{-}s) \xrightarrow{\bowtie_{\to}} t \bowtie s = (t \bullet^{+} s) \bowtie (s \bullet^{-} t)
\end{array}
$$

An η-like rewriting rule can also be incorporated to λ^{PRK} as done in [4, Thm. 37], declaring that $\bigcirc^{\pm}_{x}.\, (t \bullet^{\pm} x) \xrightarrow{\eta_{\circ}} t$ if $x \notin \mathsf{fv}(t)$. The λ^{PRK}-calculus (with or without the $\eta_{\circ}$ rule) enjoys the following properties:

▶ **Theorem 3.**

1. Subject reduction. *If $\Gamma \vdash_{\mathrm{PRK}} t : P$ and $t \to s$, then $\Gamma \vdash_{\mathrm{PRK}} s : P$.*
2. Confluence. *The λ^{PRK}-calculus has the Church–Rosser property.*
3. Classical refinement. *$A_1^{\oplus}, \ldots, A_n^{\oplus} \vdash B^{\oplus}$ holds in λ^{PRK} if and only if $A_1, \ldots, A_n \vdash B$ holds in the classical second-order natural deduction system* NK.

Proof. Subject reduction is a straightforward extension of [4, Prop. 24], with minor adaptations to account for implication, co-implication, and second-order quantification. Confluence follows from the fact that λ^{PRK} can be modeled as an orthogonal higher-order rewriting system in the sense of Nipkow [35]. Classical refinement is an extension of Prop. 38 and Thm. 39 from [4]; this theorem has two parts:

- The "only if" direction ($A_1^{\oplus}, \ldots, A_n^{\oplus} \vdash_{\mathrm{PRK}} B^{\oplus}$ implies $A_1, \ldots, A_n \vdash_{\mathsf{NK}} B$) means that PRK is a *conservative extension* of classical second-order logic. To prove this statement, we generalize the statement as follows: if $P_1, \ldots, P_n \vdash_{\mathrm{PRK}} Q$ then $\iota(P_1), \ldots, \iota(P_n) \vdash_{\mathsf{NK}} \iota(Q)$, where $\iota(A^{\oplus}) = \iota(A^{+}) = A$ and $\iota(A^{\ominus}) = \iota(A^{-}) = \neg A$. This can be shown by a straightforward induction on the derivation of the first judgment.
- The "if" direction ($A_1, \ldots, A_n \vdash_{\mathsf{NK}} B$ implies $A_1^{\oplus}, \ldots, A_n^{\oplus} \vdash_{\mathrm{PRK}} B^{\oplus}$) means that classical logic can be *embedded* into PRK. The essence of the proof is showing that all the inference rules of classical second-order natural deduction are admissible in λ^{PRK}, taking the weak affirmation of all propositions (*i.e.* decorating all formulas with "$^{\oplus}$"). Some cases are subtle, especially elimination rules. Here we show the introduction and elimination rules for quantifiers:
  1. *Universal introduction.* Let $\Gamma \vdash_{\mathrm{PRK}} t : (\forall\alpha.\, B)^{\oplus}$, and define $t \,@^{\mathcal{C}}\, A$ as the following term: $\bigcirc^{+}_{(x:(B\{\alpha:=A\})^{\ominus})}.\, ((t \bullet^{+} \bigcirc^{+}_{(\_:(\forall\alpha.\, B)^{\oplus})}.\, \langle A, x\rangle^{-})@^{+}A \bullet^{+} x)$. Then we have that $\Gamma \vdash_{\mathrm{PRK}} t \,@^{\mathcal{C}}\, A : B\{\alpha := A\}^{\oplus}$.
  2. *Universal elimination.* Let $\Gamma \vdash_{\mathrm{PRK}} t : (\forall\alpha.\, B)^{\oplus}$. Define $t \,@^{\mathcal{C}}\, A$ as the following term: $\bigcirc^{+}_{(x:(B\{\alpha:=A\})^{\ominus})}.\, ((t \bullet^{+} \bigcirc^{+}_{(\_:(\forall\alpha.\, B)^{\oplus})}.\, \langle A, x\rangle^{-})@^{+}A \bullet^{+} x)$. Then we have that $\Gamma \vdash_{\mathrm{PRK}} t \,@^{\mathcal{C}}\, A : B\{\alpha := A\}^{\oplus}$.
  3. *Existential introduction.* Let $\Gamma \vdash_{\mathrm{PRK}} t : (B\{\alpha := A\})^{\oplus}$. Define $\langle A, t\rangle^{\mathcal{C}}$ as the following term: $\bigcirc^{+}_{(\_:(\exists\alpha.\, B)^{\ominus})}.\, \langle A, t\rangle^{+}$. Then we have that $\Gamma \vdash_{\mathrm{PRK}} \langle A, t\rangle^{\mathcal{C}} : (\exists\alpha.\, B)^{\oplus}$.

4. *Existential elimination.* Let $\Gamma \vdash_{\mathrm{PRK}} t : (\exists\alpha.\, A)^{\oplus}$ and $\Gamma, x : A^{\oplus} \vdash_{\mathrm{PRK}} s : B^{\oplus}$ with $\alpha \notin \mathsf{ftv}(\Gamma, P)$. Define $\nabla^{\mathcal{C}}\, t[_{(\alpha,x)}.s]$ as the following term: $\bigcirc^{+}_{(y:B^{\ominus})}.\, (\nabla^{+}\, t'[_{(\alpha,x)}.s] \bullet^{+} y)$ where $t' \stackrel{\text{def}}{=} t \bullet^{+} \bigcirc^{-}_{(\_:(\exists\alpha.\, A)^{\oplus})}.\, \lambda^{-}_{\alpha}.\, \bigcirc^{-}_{(x:A^{\oplus})}.\, (s \bowtie_{A^{-}} y)$. Then $\Gamma \vdash_{\mathrm{PRK}} \nabla^{\mathcal{C}}\, t[_{(\alpha,x)}.s] : B^{\oplus}$. ◀

3 Böhm–Berarducci Encodings

It is well-known that, in System F, logical connectives such as $\top$, $\bot$, $\wedge$, $\vee$, $\exists$, as well as inductive data types, can be represented using only $\forall$ and $\to$ by means of their *Böhm–Berarducci encodings* [7], which can be understood as *universal properties* or *structural induction principles*. Böhm–Berarducci encodings can be reproduced in λ^{PRK}. In the following subsections we study the encoding of connectives in terms of universal quantification and implication.

The encoding of conjunction, for instance, can be taken to be $A \wedge B \stackrel{\text{def}}{=} \forall\alpha.\, ((A \to B \to \alpha) \to \alpha)$. Then **positive typing rules for conjunction**, analogous to $\mathrm{I}^{+}_{\wedge}$ and $\mathrm{E}^{+}_{\wedge i}$ are derivable, and their constructions simulate the $\beta^{+}_{\wedge}$ rule. Indeed, let $X := (A_1 \to A_2 \to A_i) \to \alpha$ and $Y := A_1 \to A_2 \to \alpha$. Moreover, let $X_i := (A_1 \to A_2 \to A_i) \to A_i$ and $Y_i := A_1 \to A_2 \to A_i$. Given $\Gamma \vdash t_1 : A_1^{\oplus}$ and $\Gamma \vdash t_2 : A_2^{\oplus}$ and $\Gamma \vdash s : (A_1 \wedge A_2)^{+}$, define:

- $\langle t_1, t_2\rangle^{+} \stackrel{\text{def}}{=} \lambda^{+}_{\alpha}.\, \bigcirc^{+}_{(\_:X^{\ominus})}.\, \lambda^{+}_{(x:Y^{\oplus})}.\bigcirc^{+}_{(y:\alpha^{\ominus})}.\, x \bullet^{+} (\bigcirc^{-}_{(\_:Y^{\oplus})}.\, (t_1\, ;^{-} u))@^{+} t_1 \bullet^{+} u@^{+} t_2 \bullet^{+} y$,
  where $u \stackrel{\text{def}}{=} \bigcirc^{-}_{(\_:(B\to\alpha)^{\oplus})}.\, (t_2\, ;^{-} y)$.
- $\pi^{+}_{i}(s) \stackrel{\text{def}}{=} \bigcirc^{+}_{(x:A_i^{\ominus})}.\, (t_1@^{+}A_i \bullet^{+} (\bigcirc^{+}_{(\_:X_i^{\oplus})}.\, (r\, ;^{-} x))@^{+} r \bullet x)$,
  where $r \stackrel{\text{def}}{=} \bigcirc^{+}_{(\_:Y_i^{\ominus})}.\, \lambda^{+}_{(y_1:A_1^{\oplus})}.\bigcirc^{+}_{(\_:(A_2\to A_i)^{\ominus})}.\, \lambda^{+}_{(y_2:A_2^{\oplus})}.y_i$.

Then $\Gamma \vdash \langle t_1, t_2\rangle^{+} : (A_1 \wedge A_2)^{+}$ and $\Gamma \vdash \pi^{+}_{i}(s) : A_i^{\oplus}$ and it can be easily checked that $\pi^{+}_{i}(\langle t_1, t_2\rangle^{+}) \to^{*} t_i$ (using $\eta_{\circ}$).

On the other hand, **negative typing rules for conjunction**, analogous to $\mathrm{I}^{-}_{\wedge i}$ and a weak variant of $\mathrm{E}^{-}_{\wedge}$ can also be derived. First note that, given terms $\Gamma \vdash p : (A \to B)^{\oplus}$ and $\Gamma \vdash q : A^{\oplus}$, a term $p \mathbin{@^{\mathcal{C}}} q$ may be defined in such a way that $\Gamma \vdash_{\mathrm{PRK}} p \mathbin{@^{\mathcal{C}}} q : B^{\oplus}$. An explicit construction is $p \mathbin{@^{\mathcal{C}}} q \stackrel{\text{def}}{=} \bigcirc^{+}_{(x:B^{\ominus})}.\, (p \bullet^{+} (\bigcirc^{-}_{(\_:(A\to B)^{\oplus})}.\, (q\, ;^{-} x))@^{+} q \bullet^{+} x)$. Then we may encode $\mathrm{I}^{-}_{\wedge i}$ and a weak variant of $\mathrm{E}^{-}_{\wedge}$ as follows:

- $\mathsf{in}^{-}_{i}(t) \stackrel{\text{def}}{=} \langle A_i, \bigcirc^{-}_{(\_:X_i^{\oplus})}.\, (r\, ;^{-} t)\rangle^{-}$,
  where $r \stackrel{\text{def}}{=} \bigcirc^{+}_{(\_:Y_i^{\ominus})}.\, \lambda^{+}_{(y_1:A_1^{\oplus})}.\bigcirc^{+}_{(\_:(A_2\to A_i)^{\ominus})}.\, \lambda^{+}_{(y_2:A_2^{\oplus})}.y_i$.
- $\delta^{-} t\, [_{a_1}.s_1][_{a_2}.s_2] \stackrel{\text{def}}{=} \nabla^{-}\, t[_{(\alpha,x:X^{\ominus})}.\bigcirc^{-}_{(c:C^{\oplus})}.\, s_1' \bullet^{-} c]$,
  where $s_1' \stackrel{\text{def}}{=} s_1\{a_1 := \bigcirc^{-}_{(z_1:A_1^{\oplus})}.\, (s_2' \bowtie_{A^{-}} c)\}$, and $s_2' \stackrel{\text{def}}{=} s_2\{a_2 := \bigcirc^{-}_{(z_2:A_2^{\oplus})}.\, (u \bowtie_{A^{-}} x)\}$,
  and $u \stackrel{\text{def}}{=} \bigcirc^{+}_{:X^{\ominus}}.\, \lambda^{+}_{y:Y^{\oplus}}.(y \mathbin{@^{\mathcal{C}}} z_1 \mathbin{@^{\mathcal{C}}} z_2)$.

Note that $\Gamma \vdash \mathsf{in}^{-}_{i}(t) : (A_1 \wedge A_2)^{-}$ and $\Gamma \vdash \delta^{-} t\, [_{a_1}.s_1][_{a_2}.s_2] : C^{\ominus}$. However, unfortunately, it is **not** the case that $\delta^{-}\mathsf{in}^{-}_{i}(t)\, [_{a_1}.s_1][_{a_2}.s_2] \to^{*} s_i\{a_i := t\}$; in fact the computation becomes stuck.

In general, these kinds of encodings are able to simulate reduction for the positive half of the system but not for the negative half[1]. This seems to suggest that λ^{PRK} cannot be fully simulated by the $\{\forall, \to\}$ fragment, although we do not know of a proof of this fact and there might exist other encodings which allow simulating the full λ^{PRK} calculus.

[1] Naturally, one may consider dual encodings in terms of $\exists$ and $\ltimes$, for example $A \vee B = \exists\alpha.\, ((A \ltimes B \ltimes \alpha) \ltimes \alpha)$, which behave well only for the negative half of the system.

4 Normalization of Second-Order λ^{PRK}

In this section we construct a **reducibility model** for λ^{PRK} and we prove **adequacy** (Thm. 5) of the model, from which **strong normalization** of second-order λ^{PRK} follows. We only discuss the proof of strong normalization for the calculus without the $\eta_\circ$ rule[2].

In [4], strong normalization for the propositional fragment of λ^{PRK} is shown via a translation to System F extended with recursive type constraints enjoying a (non-strict) positivity condition. This technique does not seem to extend to the second-order case. The problem is that the translation given in [4] is *not closed under type substitution*. More precisely, if we denote the translation by $\{\!\{-\}\!\}$, an equality such that $\{\!\{t\{\alpha := A\}\}\!\} = \{\!\{t\}\!\}\{\alpha := \{\!\{A\}\!\}\}$ does not hold in general, making the proof fail.

Our proof of strong normalization is based on an adaptation of Girard's technique of reducibility candidates. Specifically, we adapt Mendler's proof of strong normalization for the extended System F given in [31]. We begin by defining an *untyped* version of λ^{PRK}:

The untyped λ^{PRK}-calculus ($\lambda^{\mathrm{PRK}}_{\mathbf{U}}$). By $\mathbf{U}$ we denote the set of *untyped terms*, given by the following grammar:

$$\begin{array}{rcl} a, b, c, \dots & ::= & x \mid a \bowtie b \mid \langle a, b\rangle \mid \pi_i(a) \mid \mathsf{in}_i(a) \mid \delta a\,[x.b][y.c] \mid \lambda_x.\, a \mid a@b \mid (a\,;b) \mid \varrho a[_{x;y}.b] \\ & \mid & \mathsf{N}a \mid \mathsf{M}a \mid \lambda_\diamond.\, a \mid a@\diamond \mid \langle \diamond, a\rangle \mid \nabla\, a[_{(\diamond,x)}.b] \end{array}$$

The reduction relation $\to_{\mathbf{U}} \subseteq \mathbf{U} \times \mathbf{U}$ of the $\lambda^{\mathrm{PRK}}_{\mathbf{U}}$-calculus is defined by the following reduction rules, closed by compatibility under arbitrary reduction contexts:

$$\begin{array}{rcl@{\qquad}rcl} \pi_i(\langle a_1, a_2\rangle) & \to_{\mathbf{U}} & a_i & \delta(\mathsf{in}_i(a))\,[x.b_1][x.b_2] & \to_{\mathbf{U}} & b_i\{x := a\} \\ (\lambda_x.\, a)@b & \to_{\mathbf{U}} & a\{x := b\} & \varrho(a_1\,;a_2)[_{x;y}.b] & \to_{\mathbf{U}} & b\{x := a_1\}\{y := a_2\} \\ \mathsf{M}(\mathsf{N}a) & \to_{\mathbf{U}} & a & & & \\ (\lambda_\diamond.\, a)@\diamond & \to_{\mathbf{U}} & a & \nabla\,\langle \diamond, a\rangle[_{(\diamond,x)}.b] & \to_{\mathbf{U}} & b\{x := a\} \\ \langle a_1, a_2\rangle \bowtie \mathsf{in}_i(b) & \to_{\mathbf{U}} & a_i \bowtie b & \mathsf{in}_i(a) \bowtie \langle b_1, b_2\rangle & \to_{\mathbf{U}} & a \bowtie b_i \\ \lambda_x.\, a \bowtie (b\,;c) & \to_{\mathbf{U}} & a\{x := b\} \bowtie c & (a\,;b) \bowtie \lambda_x.\, c & \to_{\mathbf{U}} & b \bowtie c\{x := a\} \\ (\mathsf{N}a) \bowtie (\mathsf{N}b) & \to_{\mathbf{U}} & b \bowtie a & & & \\ \lambda_\diamond.\, a \bowtie \langle \diamond, b\rangle & \to_{\mathbf{U}} & a \bowtie b & \langle \diamond, a\rangle \bowtie \lambda_\diamond.\, b & \to_{\mathbf{U}} & a \bowtie b \end{array}$$

where $a \bowtie b \stackrel{\text{def}}{=} (a@b) \bowtie (b@a)$. The set $\mathbf{CAN} \subseteq \mathbf{U}$ of *canonical terms* is the set of terms built with a constructor, *i.e.* of any of the forms: $\langle a, b\rangle$, $\mathsf{in}_i(a)$, $\lambda_x.\, a$, $(a\,;b)$, $\mathsf{N}a$, $\lambda_\diamond.\, a$, $\langle \diamond, a\rangle$.

Note that the untyped calculus $\lambda^{\mathrm{PRK}}_{\mathbf{U}}$ is obtained from λ^{PRK} by erasing all signs and type annotations from terms, replacing types and type variables by a placeholder "$\diamond$" in introductors and eliminators for quantifiers, and identifying[3] "weak" abstraction and application ($\bigcirc_x.\, t$ and $t \bullet s$) with regular abstraction and application ($\lambda_x.\, t$ and $t@s$). It is easy to note that the $\to_{\mathbf{U}}$ reduction is confluent, observing that it can be modeled as an orthogonal higher-order rewriting system [35].

One difficult aspect of the strong normalization proof is that terms of type $A^{\oplus}$ behave as functions "$A^{\ominus} \to A^{+}$" and, dually, terms of $A^{\ominus}$ behave as functions "$A^{\oplus} \to A^{-}$". Consequently, sets of *reducible terms* cannot be defined by straightforward recursion, as this would lead to a non-well-founded mutual dependency between reducible terms of types $A^{\oplus}$ and $A^{\ominus}$. To address this difficulty, we follow Mendler's approach of taking fixed points in the *complete lattice of reducibility candidates*.

[2] Strong normalization for the full λ^{PRK}-calculus with the $\eta_\circ$ rule comes out as a relatively easy corollary by postponing $\eta_\circ$ steps (see *e.g.* [4, Theorem 37] for a similar result).

[3] This identification is not essential, but just a matter of syntactic economy.

4.1 A Reducibility Model for λ^{PRK}

We begin by recalling a few standard notions from order theory. A *complete lattice* is a partially ordered set $(A, \leq)$ such that every subset $B \subseteq A$ has a least upper bound and a greatest lower bound, denoted respectively by $\bigvee B$ and $\bigwedge B$. Then (see [12, Thm. 2.35]):

▶ **Theorem 4** (Knaster–Tarski fixed point theorem). *If $(A, \leq)$ is a complete lattice and $f : A \to A$ is an order-preserving map,* i.e. $a \leq a' \implies f(a) \leq f(a')$*, then f has a* least fixed point *and a* greatest fixed point*, given respectively by: $\boldsymbol{\mu}(f) = \bigwedge\{a \in A \mid f(a) \leq a\}$ and $\boldsymbol{\nu}(f) = \bigvee\{a \in A \mid a \leq f(a)\}$.*

We write $\boldsymbol{\mu}(\xi.f(\xi))$ for $\boldsymbol{\mu}(f)$ and $\boldsymbol{\nu}(\xi.f(\xi))$ for $\boldsymbol{\nu}(f)$.

Reducibility candidates. Let $\mathbf{SN} \subseteq \mathbf{U}$ denote the set of *strongly normalizing* terms, with respect to $\to_{\mathbf{U}}$. A set $\xi \subseteq \mathbf{SN}$ is *closed by reduction* if for every $a, b \in \mathbf{U}$ such that $a \in \xi$ and $a \to_{\mathbf{U}} b$, one has that $b \in \xi$. A set $\xi \subseteq \mathbf{SN}$ is *complete* if for every $a \in \mathbf{SN}$ the following holds:

$$(\forall b \in \mathbf{CAN}.\ ((a \to_{\mathbf{U}}^{*} b) \implies b \in \xi)) \qquad \text{implies} \qquad a \in \xi$$

A set $\xi \subseteq \mathbf{SN}$ is a *reducibility candidate* (or a r.c. for short) if it is closed by reduction and complete. We write $\mathbf{RC}$ for the set of all r.c.'s, that is, $\mathbf{RC} \stackrel{\text{def}}{=} \{\xi \subseteq \mathbf{SN} \mid \xi \text{ is a r.c.}\}$.

It is easy to see that reducibility candidates are non-empty. In particular, for every $\xi \in \mathbf{RC}$ we have that any variable $x \in \xi$ is strongly normalizing and it vacuously verifies the property $\forall c \in \mathbf{CAN}.\ ((x \to_{\mathbf{U}}^{*} c) \implies c \in \xi)$ so, since ξ is complete, we have that $x \in \xi$. Moreover, the set $\mathbf{RC}$ forms a complete lattice ordered by inclusion $\subseteq$. Following Mendler [31, Prop. 2], the greatest lower bound of $\{\xi_i\}_{i \in I}$ is given by the intersection $\cap_{i \in I} \xi_i$, and the bottom element is the set $\bot = \{a \in \mathbf{SN} \mid \nexists b \in \mathbf{CAN}.\, a \to_{\mathbf{U}}^{*} b\}$ of terminating terms that have no canonical reduct.

Operations on reducibility candidates. For each set of canonical terms $X \subseteq \mathbf{CAN}$, we define its *closure* $\mathbb{C}X$ as the set of all strongly normalizing terms whose canonical reducts are in X. More precisely, $\mathbb{C}X \stackrel{\text{def}}{=} \{a \in \mathbf{SN} \mid \forall b \in \mathbf{CAN}.((a \to_{\mathbf{U}}^{*} b) \implies b \in X)\}$. If ξ_1, ξ_2 are r.c.'s and if $\{\xi_i\}_{i \in I}$ is a set of r.c.'s, we define the following operations:

$$\begin{array}{rcl@{\qquad}rcl}
(\xi_1 \times \xi_2) & \stackrel{\text{def}}{=} & \mathbb{C}\{\langle a_1, a_2\rangle \mid a_1 \in \xi_1, a_2 \in \xi_2\} & (\xi_1 + \xi_2) & \stackrel{\text{def}}{=} & \mathbb{C}\{\mathsf{in}_i(a) \mid i \in \{1,2\}, a \in \xi_i\} \\
(\xi_1 \to \xi_2) & \stackrel{\text{def}}{=} & \{a \in \mathbf{SN} \mid \forall b \in \xi_1.\ a@b \in \xi_2\} & (\xi_1 \ltimes \xi_2) & \stackrel{\text{def}}{=} & \mathbb{C}\{(a_1\,;a_2) \mid a_1 \in \xi_1, a_2 \in \xi_2\} \\
\sim\!\xi & \stackrel{\text{def}}{=} & \mathbb{C}\{\mathsf{N}a \mid a \in \xi\} & & & \\
\mathbf{\Pi}_{i \in I}\, \xi_i & \stackrel{\text{def}}{=} & \{a \in \mathbf{SN} \mid \forall i \in I.\ a@\Diamond \in \xi_i\} & \mathbf{\Sigma}_{i \in I}\, \xi_i & \stackrel{\text{def}}{=} & \mathbb{C}\{\langle \Diamond, a\rangle \mid \exists i \in I.\ a \in \xi_i\}
\end{array}$$

It can be checked that all these operations map r.c.'s to r.c.'s.
A straightforward observation is that the arrow operator is order-reversing on the left, *i.e.* that if $\xi_1 \subseteq \xi_1'$ then $(\xi_1' \to \xi_2) \subseteq (\xi_1 \to \xi_2)$.

Orthogonality. The idea of the normalization proof is, as usual, to associate, to each type P, a set of *reducible terms* $[\![P]\!] \in \mathbf{RC}$. The interpretation of a type variable, such as $[\![\alpha^+]\!]$ or $[\![\alpha^-]\!]$ shall be given by an *environment* ρ, mapping type variables to r.c.'s. However, the sets $[\![\alpha^+]\!]$ and $[\![\alpha^-]\!]$ *should not be chosen independently of each other*: we require them to be orthogonal in the following sense.

Two reducibility candidates $\xi_1, \xi_2 \in \mathbf{RC}$ are *orthogonal*, if for all $a_1 \in \xi_1$ and $a_2 \in \xi_2$ we have that $(a_1 \bowtie a_2) \in \mathbf{SN}$. We write $\perp\!\!\!\perp$ for the set of all pairs $(\xi_1, \xi_2) \in \mathbf{RC}^2$ such that ξ_1 and ξ_2 are orthogonal.

Reducible terms. The set of reducible terms is defined by induction on the following notion of *measure* $\#(-)$ of a type P, given by $\#(A^+) = \#(A^-) \stackrel{\text{def}}{=} 2|A|$ and $\#(A^\oplus) = \#(A^\ominus) \stackrel{\text{def}}{=} 2|A| + 1$, where $|A|$ denotes the *size*, *i.e.* the number of symbols, of the pure type A. Note for example that $\#((A \wedge B)^\oplus) > \#((A \wedge B)^+) > \#(A^\oplus)$.

An *environment* is a function ρ mapping each type variable $\alpha^\pm$, with either positive or negative sign, to a reducibility candidate $\rho(\alpha^\pm) \in \mathbf{RC}$, in such a way that $(\rho(\alpha^+), \rho(\alpha^-)) \in \perp\!\!\!\perp$. If $(\xi^+, \xi^-) \in \perp\!\!\!\perp$, we write $\rho[\alpha := \xi^+, \xi^-]$ for the environment ρ' that extends ρ in such a way that $\rho'(\alpha^+) = \xi^+$ and $\rho'(\alpha^-) = \xi^-$ and $\rho'(\beta^\pm) = \rho(\beta^\pm)$ for any other type variable $\beta \neq \alpha$.

Given an environment ρ, we define the set of *reducible terms* of type P under the environment ρ, written $[\![P]\!]_\rho$, by induction on the measure $\#(P)$ as follows:

$$
\begin{array}{rcl@{\qquad}rcl}
[\![\alpha^+]\!]_\rho & \stackrel{\text{def}}{=} & \rho(\alpha^+) & [\![\alpha^-]\!]_\rho & \stackrel{\text{def}}{=} & \rho(\alpha^-) \\
[\![(A \wedge B)^+]\!]_\rho & \stackrel{\text{def}}{=} & [\![A^\oplus]\!]_\rho \times [\![B^\oplus]\!]_\rho & [\![(A \wedge B)^-]\!]_\rho & \stackrel{\text{def}}{=} & [\![A^\ominus]\!]_\rho + [\![B^\ominus]\!]_\rho \\
[\![(A \vee B)^+]\!]_\rho & \stackrel{\text{def}}{=} & [\![A^\oplus]\!]_\rho + [\![B^\oplus]\!]_\rho & [\![(A \vee B)^-]\!]_\rho & \stackrel{\text{def}}{=} & [\![A^\ominus]\!]_\rho \times [\![B^\ominus]\!]_\rho \\
[\![(A \to B)^+]\!]_\rho & \stackrel{\text{def}}{=} & [\![A^\oplus]\!]_\rho \to [\![B^\oplus]\!]_\rho & [\![(A \to B)^-]\!]_\rho & \stackrel{\text{def}}{=} & [\![A^\oplus]\!]_\rho \ltimes [\![B^\ominus]\!]_\rho \\
[\![(A \ltimes B)^+]\!]_\rho & \stackrel{\text{def}}{=} & [\![A^\ominus]\!]_\rho \ltimes [\![B^\oplus]\!]_\rho & [\![(A \ltimes B)^-]\!]_\rho & \stackrel{\text{def}}{=} & [\![A^\ominus]\!]_\rho \to [\![B^\ominus]\!]_\rho \\
[\![(\neg A)^+]\!]_\rho & \stackrel{\text{def}}{=} & \sim [\![A^\ominus]\!]_\rho & [\![(\neg A)^-]\!]_\rho & \stackrel{\text{def}}{=} & \sim [\![A^\oplus]\!]_\rho \\
[\![(\forall \alpha.\, A)^+]\!]_\rho & \stackrel{\text{def}}{=} & \mathbf{\Pi}_{(\xi^+,\xi^-) \in \perp\!\!\!\perp} [\![A^\oplus]\!]_{\rho[\alpha := \xi^+, \xi^-]} & [\![(\forall \alpha.\, A)^-]\!]_\rho & \stackrel{\text{def}}{=} & \mathbf{\Sigma}_{(\xi^+,\xi^-) \in \perp\!\!\!\perp} [\![A^\ominus]\!]_{\rho[\alpha := \xi^+, \xi^-]} \\
[\![(\exists \alpha.\, A)^+]\!]_\rho & \stackrel{\text{def}}{=} & \mathbf{\Sigma}_{(\xi^+,\xi^-) \in \perp\!\!\!\perp} [\![A^\oplus]\!]_{\rho[\alpha := \xi^+, \xi^-]} & [\![(\exists \alpha.\, A)^-]\!]_\rho & \stackrel{\text{def}}{=} & \mathbf{\Pi}_{(\xi^+,\xi^-) \in \perp\!\!\!\perp} [\![A^\ominus]\!]_{\rho[\alpha := \xi^+, \xi^-]} \\
[\![A^\oplus]\!]_\rho & \stackrel{\text{def}}{=} & \boldsymbol{\mu}(\xi.((\xi \to [\![A^-]\!]_\rho) \to [\![A^+]\!]_\rho)) & [\![A^\ominus]\!]_\rho & \stackrel{\text{def}}{=} & \boldsymbol{\nu}(\xi.((\xi \to [\![A^+]\!]_\rho) \to [\![A^-]\!]_\rho))
\end{array}
$$

It is straightforward to check for each type P and each environment ρ that $[\![P]\!]_\rho$ is a reducibility candidate, by induction on the measure $\#(P)$. In the case of $[\![A^\oplus]\!]_\rho$, by the Knaster–Tarski theorem (Thm. 4), to see that the least fixed point exists, it suffices to observe that the mapping $f(\xi) = ((\xi \to [\![A^-]\!]_\rho) \to [\![A^+]\!]_\rho)$ is order-preserving. The case of $[\![A^\ominus]\!]_\rho$ is similar.

Adequacy of the reducibility model. For each term t of λ^{PRK}, we define an untyped term $|t|$ of $\lambda^{\text{PRK}}_{\mathbf{U}}$ via the obvious forgetful map. For instance $|\bigcirc^+_{(x:(\exists \alpha.\, \alpha)^\ominus)}.\, \langle B, z^{B^\oplus} \rangle^+| = \lambda_x.\, \langle \diamond, z \rangle$. Note that each reduction step $t \to s$ in λ^{PRK} is mapped to a reduction step $|t| \to_{\mathbf{U}} |s|$ in $\lambda^{\text{PRK}}_{\mathbf{U}}$. Hence, if $|t|$ is strongly normalizing with respect to $\to_{\mathbf{U}}$, then t is strongly normalizing with respect to $\to$.

A *substitution* is a function σ mapping each variable to a term in $\mathbf{U}$. We write a^σ for the term that results from the capture-avoiding substitution of each free occurrence of each variable x in a by $\sigma(x)$. We say that the substitution σ is *adequate* for the typing context Γ under the environment ρ, and we write $\sigma \vDash_\rho \Gamma$, if for each type assignment $(x : P) \in \Gamma$ we have that $\sigma(x) \in [\![P]\!]_\rho$. We are finally able to state the key result:

▶ **Theorem 5** (Adequacy). *If* $\Gamma \vdash t : P$ *and* $\sigma \vDash_\rho \Gamma$ *then* $|t|^\sigma \in [\![P]\!]_\rho$.

The proof of the adequacy theorem relies on a number of auxiliary lemmas stating properties such as $[\![A^\oplus]\!]_\rho = [\![A^\ominus]\!]_\rho \to [\![A^+]\!]_\rho$ and $([\![A^+]\!]_\rho, [\![A^-]\!]_\rho) \in \perp\!\!\!\perp$. From this we obtain as an easy corollary that the λ^{PRK}-calculus is strongly normalizing, taking ρ as the environment that maps all type variables to the bottom reducibility candidate, and σ as the identity substitution.

5 Intuitionistic Proofs and Refutations

In natural deduction, it is well-known that classical logic can be obtained from the intuitionistic system by adding a single classical axiom, such as excluded middle or double negation elimination. In sequent calculus, it is well-known that intuitionistic logic can be obtained by restricting sequents $A_1, \dots, A_n \vdash B_1, \dots, B_m$ to have at most one formula on the right. As we have seen, λ^{PRK} refines classical logic. It is a natural question to ask what subsystem of λ^{PRK} corresponds to intuitionistic logic.

In this section we characterize a restricted subsystem of λ^{PRK} that corresponds to intuitionistic logic, called λ^{PRJ}, by imposing a syntactic restriction on the shape of λ^{PRK} proofs, that forbids certain specific patterns of reasoning. In particular, a variable x introduced by a positive weak introduction $\bigcirc^+_x.\, t$ can only occur free in t inside the arguments of weak eliminations. The main result in this section is that **λ^{PRJ} refines intuitionistic second-order logic** (Thm. 9).

As mentioned before, proofs of *strong* propositions in PRK must be constructive. However, this is only true for the toplevel logical connective in the formula. In general, a proof of A^+ in PRK does not necessarily correspond to an intuitionistic proof of A. For example, a canonical proof of $(A \wedge B)^+$ is given by a proof of $A^\oplus$ and a proof of $B^\oplus$, but these subproofs may resort to classical reasoning principles.

The key to identify an intuitionistic subset of λ^{PRK} is to disallow inference rules which embody classical principles. One example is the $\text{E}^-_\neg$ rule, which derives $A^\oplus$ from $(\neg A)^-$. This rule embodies the classical principle of double negation elimination ($\neg\neg A \to A$). Another important example is the $\text{I}^+_\circ$ rule, which derives $A^\oplus$ from $A^\ominus \vdash A^+$. This rule embodies the classical principle of *consequentia mirabilis* ($(\neg A \to A) \to A$).

The analysis of the $\text{I}^+_\circ$ rule suggests that, in the intuitionistic fragment, $A^\oplus$ should not be identified with "$A^\ominus \to A^+$", but directly with A^+. One natural idea would be to impose an invariant over terms of the form $\bigcirc^+_{(x:A^\ominus)}.\, t$, in such a way that the body t may have no free occurrences of the negative counterfactual x. With this invariant, all instances of the $\text{I}^+_\circ$ rule are actually instances of the following variant of $\text{I}^+_\circ$:

$$\frac{\Gamma \vdash t : A^+ \qquad x \notin \mathsf{fv}(t)}{\Gamma \vdash \bigcirc^+_{(x:A^\ominus)}.\, t : A^\oplus}\,\text{I}^+_\circ(\text{variant})$$

This in turn means that, in an application $t \bullet^+ s$, the argument s is *useless*. Indeed, if t becomes $\bigcirc^+_{(x:A^\ominus)}.\, t'$, the invariant ensures that $x \notin \mathsf{fv}(t')$, so $(\bigcirc^+_{(x:A^\ominus)}.\, t') \bullet^+ s \to t'$, which does not depend on the specific choice of s.

Rather than completely forbidding classical reasoning principles, we relax this condition so that classical principles are allowed as long as they are useless, *i.e.* inside the argument of an application $t \bullet^+ s$. Furthermore, the invariant over terms of the form $\bigcirc^+_{(x:A^\ominus)}.\, t$, requesting that $x \notin \mathsf{fv}(t)$, can also be relaxed, in such a way that x is allowed to occur in t as long as all of its free occurrences are useless. Formally:

▶ **Definition 6** (Intuitionistic terms). *A subterm of a term t is said to be* **useless** *if it lies inside the argument of a positive weak elimination. More precisely, given a term $t = \mathtt{C}\langle s\rangle$, we say that the subterm s under the context $\mathtt{C}$ is useless if and only if there exist contexts $\mathtt{C}_1, \mathtt{C}_2$ and a term u such that $\mathtt{C}$ is of the form $\mathtt{C}_1\langle u \bullet^+ \mathtt{C}_2\langle\Box\rangle\rangle$. A subterm of t is* **useful** *if it is not useless. A term is said to be* **intuitionistic** *if and only if the two following conditions hold:*

1. **Useless negative eliminations ($\mathbf{E}^-_\wedge$, $\mathbf{E}^-_\to$, $\mathbf{E}^-_\neg$, $\mathbf{E}^-_\forall$).** *There are no useful subterms of any of the following forms:* $\delta^- t\,[_{(x:A^\ominus)}.s][_{(y:B^\ominus)}.u]$, $\varrho^- t[_{(x:A^\oplus);(y:B^\ominus)}.s]$, $\mathsf{M}^- t$, $\nabla^-\, t[_{(\alpha,x)}.s]$.
2. **Useless negative counterfactuals.** *In every useful subterm of the form* $\bigcirc^+_{(x:A^\ominus)}.\,t$, *there are no useful occurrences of* x *in* t.

The λ^{PRJ} type system. The type system λ^{PRJ} is defined by imposing the restriction on λ^{PRK} that terms be intuitionistic. More precisely, we say that a judgment $\Gamma \vdash t : P$ holds in PRJ, and in this case we write $\Gamma \vdash_{\mathrm{PRJ}} t : P$, if the judgment holds in PRK and furthermore t is an intuitionistic term. We also write $P_1, \dots, P_n \vdash_{\mathrm{PRJ}} Q$ if there exists a term t such that $x_1 : P_1, \dots, x_n : P_n \vdash_{\mathrm{PRJ}} t : Q$.

▶ **Example 7.** The weak variant of the law of excluded middle, $(A \vee \neg A)^\oplus$, can be proven in PRK. For example, if we define $\Cap^+_A$:

$$\Cap^+_A \stackrel{\text{def}}{=} \bigcirc^+_{(x:(A\vee\neg A)^\ominus)}.\,\mathsf{in}_2^+(\bigcirc^+_{(y:\neg A^\ominus)}.\,\mathsf{N}^+\pi_1^-(x \bullet^- \Delta^+_{y,A}))$$
$$\text{where } \Delta^+_{y,A} \stackrel{\text{def}}{=} \bigcirc^+_{(\_:(A\vee\neg A)^\ominus)}.\,\mathsf{in}_1^+(\bigcirc^+_{(z:A^\ominus)}.\,(y \bowtie_{A^+} \bigcirc^+_{(\_:\neg A^\ominus)}.\,\mathsf{N}^+ z))$$

we can note that $\vdash_{\mathrm{PRK}} \Cap^+_A : (A \vee \neg A)^\oplus$ holds. However, $\Cap^+_A$ is not intuitionistic, due to the fact that *there is a useful occurrence of the negative counterfactual* x.

The intuitionistic fragment is stable by reduction:

▶ **Proposition 8** (Subject reduction for PRJ). *Let* $\Gamma \vdash_{\mathrm{PRJ}} t : P$ *and* $t \to s$. *Then* $\Gamma \vdash_{\mathrm{PRJ}} s : P$.

Proof. If X is a set of variables, we say that a term t is X*-intuitionistic* if it is intuitionistic and, furthermore, it has no *useful* free occurrences of variables in X. We write $\Gamma \vdash^X_{\mathrm{PRJ}} t : P$, if the judgment is derivable in PRK and t is X-intuitionistic. The statement of subject reduction is generalized as follows: $\Gamma \vdash^X_{\mathrm{PRJ}} t : P$ and $t \to s$, then $\Gamma \vdash^X_{\mathrm{PRJ}} s : P$.

The interesting case is the β rule for positive weak proofs, $(\bigcirc^+_{x:A^\ominus}.\,t) \bullet^+ s \xrightarrow{\beta^+_\circ} t\{x := s\}$. By hypothesis, $\Gamma \vdash^X_{\mathrm{PRJ}} (\bigcirc^+_{x:A^\ominus}.\,t) \bullet^+ s : A^+$. This judgment can only be derived from the $\mathrm{I}^+_\circ$ rule, so $\Gamma, x : A^\ominus \vdash^X_{\mathrm{PRJ}} t : A^+$. Moreover, x is a negative counterfactual, so there cannot be useful free occurrences of x in t, which means that $\Gamma, x : A^\ominus \vdash^{X\cup\{x\}}_{\mathrm{PRJ}} t : A^+$. On the other hand, s lies inside a positive application, so it is not necessarily X-intuitionistic, *i.e.* we only know $\Gamma \vdash_{\mathrm{PRK}} s : A^\ominus$. The key observation is that all the copies of s on the right-hand side $t\{x := s\}$ must be useless, because all the occurrences of x in t are useless. More precisely, from $\Gamma, x : A^\ominus \vdash^{X\cup\{x\}}_{\mathrm{PRJ}} t : A^+$ and $\Gamma \vdash_{\mathrm{PRK}} s : A^\ominus$ one concludes $\Gamma \vdash^X_{\mathrm{PRJ}} t\{x := s\} : P$ by induction on t. ◀

Reasoning principles in PRJ differ from those of PRK. For example, if P is a weak formula, a sequent $\Gamma, x : P \vdash_{\mathrm{PRK}} t : Q$ valid in PRK can always be *contraposed* to a sequent of the form $\Gamma, y : Q^\sim \vdash_{\mathrm{PRK}} t' : P^\sim$. The analogous of this contraposition principle in PRJ depends on the sign of P. If P is positive, *i.e.* $P = A^\oplus$ the sequent $\Gamma, x : A^\oplus \vdash_{\mathrm{PRJ}} t : Q$ can always be contraposed to $\Gamma, y : Q^\sim \vdash_{\mathrm{PRJ}} t' : A^\ominus$. But if P is negative, *i.e.* $P = A^\ominus$ the sequent $\Gamma, x : A^\ominus \vdash_{\mathrm{PRJ}} t : Q$ can only be contraposed to $\Gamma, y : Q^\sim \vdash_{\mathrm{PRJ}} t' : A^\oplus$ if there are no useful occurrences of x in t.

The following theorem is an analog of Thm. 3 for λ^{PRJ}.

▶ **Theorem 9** (Intuitionistic refinement). $A_1^\oplus, \dots, A_n^\oplus \vdash B^\oplus$ *holds in* λ^{PRJ} *if and only if* $A_1, \dots, A_n \vdash B$ *holds in intuitionistic second-order logic.*

6 Canonicity

In sequent calculus and natural deduction, an *indirect proof* (*e.g.* with cuts), can always be mechanically converted into a *canonical proof* (*e.g.* cut-free), in which the justification for the conclusion is immediately available, as is known from the works of Gentzen [20] and Prawitz [41]. Its philosophical importance is that the validity of an indirect proof can thus be justified by understanding it as a notation for describing a canonical proof. A practical consequence is that an explicit witness may be extracted from a proof of existence.

In this section, we formulate a **canonicity** result strengthening those of [4]. We start by introducing some nomenclature. *Neutral terms* $(e, \ldots)$ and *normal terms* $(f, \ldots)$ are given by $e ::= x \mid e \bowtie_P f \mid f \bowtie_P e \mid \pi_i^{\pm}(e) \mid \delta^{\pm} e\,[x.f][x.f] \mid e@^{\pm} f \mid \varrho^{\pm} e[_{x;y}.f] \mid \mathsf{M}^{\pm} e \mid e@^{\pm} A \mid \nabla^{\pm} e[_{(x,\alpha)}.f] \mid e \bullet^{\pm} f$ and $f ::= e \mid \langle f, f\rangle^{\pm} \mid \mathsf{in}_i^{\pm}(f) \mid \lambda_x^{\pm}.f \mid (f \,;^{\pm} f) \mid \mathsf{N}^{\pm} f \mid \lambda_\alpha^{\pm}.f \mid \langle A, f\rangle^{\pm} \mid \bigcirc_x^{\pm}.f$. Terms built with an introduction rule: $\langle t, s\rangle^{\pm}$, $\mathsf{in}_i^{\pm}(t)$, $\lambda_x^{\pm}.t$, $(t \,;^{\pm} s)$, $\mathsf{N}^{\pm} t$, $\lambda_\alpha^{\pm}.t$, $\langle A, t\rangle^{\pm}$, $\bigcirc_x^{\pm}.t$ are called *canonical*. Then:

▶ **Theorem 10** (Canonicity).

1. *If* $\vdash_{\mathrm{PRK}} t : P$*, then* t *reduces to a canonical normal form* f *such that* $\vdash_{\mathrm{PRK}} f : P$*.*
2. *If* $\vdash_{\mathrm{PRK}} t : P$*, where* P *is weak, then a canonical normal form* f *can be effectively found such that* $\vdash_{\mathrm{PRK}} \bigcirc^{\pm}_{(x:P^{\sim})}.\, f : P$

Note that this canonicity theorem applies to closed terms only, so there is no need to include commutative conversions, such as $\delta^{+} x\,[_y.t][_z.s] \bullet^{+} u \to \delta^{+} x\,[_y.t \bullet^{+} u][_z.s \bullet^{+} u]$, to unblock redexes. The preceding canonicity result extends and strengthens Thm. 35 of [4]. The first part of the theorem confirms the intuition, mentioned in the introduction, that canonical proofs of strong propositions are always constructed with an introduction rule for the corresponding logical connective, while canonical proofs of weak propositions proceed by *reductio ad absurdum*. For example, if $\vdash t : (A_1 \wedge A_2)^{+}$ is the *strong* proof of a conjunction, its normal form must be a pair $\langle t_1, t_2\rangle^{+}$ and we know that $\vdash t_i : A_i{}^{\oplus}$ must hold for $i \in \{1, 2\}$. On the other hand, if $\vdash s : (A_1 \wedge A_2)^{+}$ is the *weak* proof of a conjunction, we can only assure that its normal form is of the form $\bigcirc^{+}_{(x:(A_1 \wedge A_2)^{\ominus})}.\, s'$, where $x : (A_1 \wedge A_2)^{\ominus} \vdash s' : (A_1 \wedge A_2)^{+}$. The second part of the theorem provides the stronger guarantee that, in such case, one can compute a *canonical* s'' such that $x : (A_1 \wedge A_2)^{\ominus} \vdash s'' : (A_1 \wedge A_2)^{+}$, so in particular one has that $s'' = \langle s_1, s_2\rangle^{+}$ and that $x : (A_1 \wedge A_2)^{\ominus} \vdash s_i : A_i{}^{\oplus}$ for all $i \in \{1, 2\}$.

Canonicity can also be used to obtain a (weak) form of *disjunctive property*. In particular, from $\vdash t : (A_1 \vee A_2)^{\oplus}$ one can always find an $i \in \{1, 2\}$ and a term $\vdash \bigcirc^{+}_{(x:(A_1 \vee A_2)^{\ominus})}.\, \mathsf{in}_i^{+}(t') : (A_1 \vee A_2)^{\oplus}$ such that $x : (A_1 \vee A_2)^{\ominus} \vdash t' : A_i{}^{\oplus}$. Similarly, a (weak) form of *witness extraction* can be obtained: given $\vdash t : (\exists \alpha.\, A)^{\oplus}$ one can always find a term $\vdash \bigcirc^{+}_{(x:(\exists \alpha.\, A)^{\ominus})}.\, \langle B, t'\rangle^{+} : (\exists \alpha.\, A)^{\oplus}$ where $x : (\exists \alpha.\, A)^{\ominus} \vdash t' : A\{\alpha := B\}^{\oplus}$.

Furthermore, canonicity provides a purely syntactic proof of the consistency of PRK[4]. Note, for example, that if α is a base type, there is no canonical term t such that $\vdash_{\mathrm{PRK}} t : \alpha^{+}$.

7 Conclusion

In this paper we have extended the λ^{PRK}-calculus of [4] to incorporate implication and co-implication, as well as second-order quantifiers. From the logical point of view, this extension of λ^{PRK} **refines classical second order logic** (Thm. 3). From the computational point of view, it is **confluent** (Thm. 3) and **strongly normalizing** (Section 4). These

[4] Another way to prove consistency is using Thm. 3, noting that $\vdash_{\mathrm{PRK}} \alpha^{+}$ implies $\vdash_{\mathsf{NK}} \alpha$.

ingredients constitute a computational interpretation for second-order classical logic. We have identified a well-behaved subset of the system, called λ^{PRJ}, that **refines intuitionistic second-order logic** (Thm. 9). We have also formulated a **canonicity** (Thm. 10) result that strengthens results of previous works. One noteworthy property of λ^{PRK} is that both typing and reduction rules are fully symmetric with respect to the operation that flips signs and exchanges the roles of dual connectives, while still being confluent.

Related work. The "$\mathrm{I}_\circ^+$"/"$\mathrm{I}_\circ^-$" rules in λ^{PRK} encode a primitive variant of *consequentia mirabilis* ($(\neg A \to A) \to A$), while the "λ" rule in Barbanera and Berardi's calculus [3], as well as the "μ" rule in Parigot's $\lambda\mu$-calculus [37], encode a primitive variant of *double negation elimination* ($\neg\neg A \to A$). To prove A using double negation elimination one may assume $\neg A$ and then provide a proof of $\bot$. This proof *cannot be canonical*, as there are no introduction rules for the empty type. This motivates that we instead rely on *consequentia mirabilis*.

Strong normalization proofs are often based on reducibility candidates. Yamagata proves strong normalization for second-order formulations of classical calculi [48, 49], via reducibility candidates. Our proof is inspired by ideas known from the literature of logical relations and biorthogonality: for instance, the notions of *orthogonal* r.c.'s and *closure* of a r.c. can be traced back to Krivine's work on classical realizability [28], Pitts' $\top\top$-closed logical relations [39], and related notions (see *e.g.* [18]). The challenging aspect of λ^{PRK} is the mutually recursive dependency between $A^\oplus$ and $A^\ominus$, for which our key reference is Mendler's work [31].

The problem of finding a good calculus for classical logic has not been unquestionably settled. Current proof assistants based on type theory, such as Coq, allow classical reasoning by postulating axioms with no computational content, which breaks canonicity. An established classical calculus is Parigot's $\lambda\mu$, whose metatheory has been thoroughly developed; see for instance [16, 13, 29, 44, 45, 38, 27, 26]. One difference between $\lambda\mu$ and λ^{PRK} is that λ^{PRK} computational rules are based on the standard operation of substitution, while $\lambda\mu$ is based on an *ad hoc* substitution operator. Another difference is that the embedding of the λ-calculus into $\lambda\mu$ is an inclusion, whereas the embedding into λ^{PRK} is much more convoluted.

Another established classical calculus is Curien and Herbelin's [10] $\bar{\lambda}\mu\tilde{\mu}$, whose study is also quite mature; see for instance [40, 25, 17, 47, 11, 2, 1, 32]. One difference between $\bar{\lambda}\mu\tilde{\mu}$ and λ^{PRK} is that $\bar{\lambda}\mu\tilde{\mu}$ is not confluent unless a reduction strategy is fixed in the presence of a specific critical pair, whereas λ^{PRK} is orthogonal. Another difference is that $\bar{\lambda}\mu\tilde{\mu}$ is derived from a proof term assignment for classical sequent calculus, while λ^{PRK} is defined in natural deduction style with four forms of judgment (given by the modes A^+, A^-, $A^\oplus$, $A^\ominus$). Munch-Maccagnoni [33] proposes a classical calculus by *polarizing* Curien and Herbelin's calculus, in such a way that the reduction strategy becomes determined by the polarities.

As mentioned in the introduction, λ^{PRK} is related to Nelson's constructible falsity [34]. Parigot [36] studies *free deduction*, a system for classical logic in which natural deduction and sequent calculus can both be embedded. Rumfitt [43] proposes *bilateral* logical systems, in which assertion and denial judgments, with dual rules, are formulated. Zeilberger [50] studies a polarized logical system with proofs and refutations distinguishing between verificationist and pragmatist connectives. These systems, however, do not distinguish between weak and strong propositions, and they do not have rules analogous to $\mathrm{I}_\circ^\pm/\mathrm{E}_\circ^\pm$.

Future work. The merely *logical* correspondence between λ^{PRK} and other classical calculi is immediate, given the fact that λ^{PRK} refines second-order classical logic (Thm. 3). However, it is not obvious what their relation is from the *computational* point of view.

In order to be able to build programming languages and proof assistants based on the principles of λ^{PRK}, it would be convenient to study dependently typed extensions of the system. It is not *a priori* clear what such an extension would look like.

It is known that, in classical logic, *reductio ad absurdum* can be postponed, in such a way that it is used at most once, as the last rule in the derivation [46, 24]. It would be interesting to see if this result can be reproduced in PRK.

References

1 Zena M. Ariola, Paul Downen, Hugo Herbelin, Keiko Nakata, and Alexis Saurin. Classical call-by-need sequent calculi: The unity of semantic artifacts. In Tom Schrijvers and Peter Thiemann, editors, *Functional and Logic Programming – 11th International Symposium, FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, volume 7294 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2012.

2 Zena M Ariola, Hugo Herbelin, and Alexis Saurin. Classical call-by-need and duality. In *International Conference on Typed Lambda Calculi and Applications*, pages 27–44. Springer, 2011.

3 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*, pages 495–515, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

4 Pablo Barenbaum and Teodoro Freund. A constructive logic with classical proofs and refutations. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 – July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470649.

5 Pablo Barenbaum and Teodoro Freund. Proofs and refutations for intuitionistic and second-order logic (extended version), 2022. doi:10.48550/ARXIV.2210.07274.

6 Gavin M. Bierman and Valeria CV de Paiva. On an intuitionistic modal logic. *Studia Logica*, 65(3):383–416, 2000.

7 Corrado Böhm and Alessandro Berarducci. Automatic synthesis of typed lambda-programs on term algebras. *Theor. Comput. Sci.*, 39:135–154, 1985.

8 Thierry Coquand and Gérard P. Huet. The calculus of constructions. *Inf. Comput.*, 76(2/3):95–120, 1988. doi:10.1016/0890-5401(88)90005-3.

9 Tristan Crolard. Subtractive logic. *Theor. Comput. Sci.*, 254(1-2):151–185, 2001. doi:10.1016/S0304-3975(99)00124-3.

10 Pierre-Louis Curien and Hugo Herbelin. The duality of computation, 2000.

11 Pierre-Louis Curien and Guillaume Munch-Maccagnoni. The duality of computation under focus. In Cristian S. Calude and Vladimiro Sassone, editors, *Theoretical Computer Science – 6th IFIP TC 1/WG 2.2 International Conference, TCS 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 165–181. Springer, 2010.

12 Brian A. Davey and Hilary A. Priestley. *Introduction to lattices and order*. Cambridge University Press, Cambridge, 1990.

13 René David and Walter Py. $\lambda\mu$-calculus and böhm's theorem. *The Journal of Symbolic Logic*, 66(1):407–413, 2001.

14 Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *J. ACM*, 48(3):555–604, 2001. doi:10.1145/382780.382785.

15 Nicolaas Govert De Bruijn. The mathematical language automath, its usage, and some of its extensions. In *Symposium on automatic demonstration*, pages 29–61. Springer, 1970.

16 Philippe de Groote. An environment machine for the lambda-mu-calculus. *Math. Struct. Comput. Sci.*, 8(6):637–669, 1998. URL: http://journals.cambridge.org/action/displayAbstract?aid=44787.

17 Daniel J. Dougherty, Silvia Ghilezan, and Pierre Lescanne. Characterizing strong normalization in the curien-herbelin symmetric lambda calculus: Extending the coppo-dezani heritage. *Theor. Comput. Sci.*, 398(1-3):114–128, 2008.

18 Paul Downen, Philip Johnson-Freyd, and Zena M. Ariola. Abstracting models of strong normalization for classical calculi. *J. Log. Algebraic Methods Program.*, 111:100512, 2020. doi:10.1016/j.jlamp.2019.100512.

19 Matthias Felleisen, Daniel P. Friedman, Eugene E. Kohlbecker, and Bruce F. Duba. A syntactic theory of sequential control. *Theor. Comput. Sci.*, 52:205–237, 1987. doi:10.1016/0304-3975(87)90109-5.

20 Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische zeitschrift*, 39(1):176–210, 1935.

21 Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris 7, 1972.

22 Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

23 Timothy G Griffin. A formulae-as-type notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 47–58, 1989.

24 Giulio Guerrieri and Alberto Naibo. Postponement of raa and Glivenko's theorem, revisited. *Stud Logica*, 107(1):109–144, 2019. doi:10.1007/s11225-017-9781-5.

25 Hugo Herbelin and Silvia Ghilezan. An approach to call-by-name delimited continuations. *SIGPLAN Not.*, 43(1):383–394, January 2008. doi:10.1145/1328897.1328484.

26 Delia Kesner, Eduardo Bonelli, and Andrés Viso. Strong bisimulation for control operators (invited talk). In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, volume 152 of *LIPIcs*, pages 4:1–4:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

27 Delia Kesner and Pierre Vial. Non-idempotent types for classical calculi in natural deduction style. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:3)2020.

28 Jean-Louis Krivine. Realizability in classical logic. *Panoramas et synthèses*, 27, January 2009.

29 Olivier Laurent. Polarized proof-nets and lambda-μ-calculus. *Theor. Comput. Sci.*, 290(1):161–188, 2003.

30 Per Martin-Löf. A theory of types, 1971.

31 Nax Paul Mendler. Inductive types and type constraints in the second-order lambda calculus. *Annals of pure and Applied logic*, 51(1-2):159–172, 1991.

32 Étienne Miquey. A classical sequent calculus with dependent types. *ACM Trans. Program. Lang. Syst.*, 41(2):8:1–8:47, 2019.

33 Guillaume Munch-Maccagnoni. Focalisation and classical realisability. In Erich Grädel and Reinhard Kahle, editors, *Computer Science Logic, 23rd international Workshop, CSL 2009, 18th Annual Conference of the EACSL, Coimbra, Portugal, September 7-11, 2009. Proceedings*, volume 5771 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 2009. doi:10.1007/978-3-642-04027-6_30.

34 David Nelson. Constructible falsity. *The Journal of Symbolic Logic*, 14(1):16–26, 1949.

35 Tobias Nipkow. Higher-order critical pairs. In *Proceedings 1991 Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 342–343. IEEE Computer Society, 1991.

36 Michel Parigot. Free deduction: An analysis of "computations" in classical logic. In Andrei Voronkov, editor, *Logic Programming, First Russian Conference on Logic Programming, Irkutsk, Russia, September 14-18, 1990 – Second Russian Conference on Logic Programming, St. Petersburg, Russia, September 11-16, 1991, Proceedings*, volume 592 of *Lecture Notes in Computer Science*, pages 361–380. Springer, 1991. doi:10.1007/3-540-55460-2_27.

37 Michel Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning*, pages 190–201, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

38 Pierre-Marie Pédrot and Alexis Saurin. Classical by-need. In *European Symposium on Programming*, pages 616–643. Springer, 2016.

39 Andrew M. Pitts. Parametric polymorphism and operational equivalence. *Math. Struct. Comput. Sci.*, 10(3):321–359, 2000. URL: `http://journals.cambridge.org/action/displayAbstract?aid=44651`.

40 Emmanuel Polonovski. Strong normalization of $\lambda\mu\mu$-calculus with explicit substitutions. *Lecture Notes in Computer Science*, pages 423–437, 2004.

41 Dag Prawitz. *Natural deduction: a proof-theoretical study*. PhD thesis, Almqvist & Wiksell, 1965.

42 John C Reynolds. Towards a theory of type structure. In *Programming Symposium*, pages 408–425. Springer, 1974.

43 Ian Rumfitt. "Yes" and "No". *Mind*, 109(436):781–823, 2000.

44 Alexis Saurin. Separation with streams in the $\lambda\mu$-calculus. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*, pages 356–365. IEEE, 2005.

45 Alexis Saurin. On the relations between the syntactic theories of lambda-mu-calculi. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, volume 5213 of *Lecture Notes in Computer Science*, pages 154–168. Springer, 2008. `doi:10.1007/978-3-540-87531-4_13`.

46 Jonathan P. Seldin. On the proof theory of the intermediate logic MH. *J. Symb. Log.*, 51(3):626–647, 1986. `doi:10.2307/2274019`.

47 Steffen van Bakel. Completeness and partial soundness results for intersection and union typing for $\overline{\lambda}\mu\tilde{\mu}$. *Ann. Pure Appl. Log.*, 161(11):1400–1430, 2010.

48 Yoriyuki Yamagata. Strong normalization of a symmetric lambda calculus for second-order classical logic. *Arch. Math. Log.*, 41(1):91–99, 2002.

49 Yoriyuki Yamagata. Strong normalization of the second-order symmetric lambda mu -calculus. *Inf. Comput.*, 193(1):1–20, 2004.

50 Noam Zeilberger. On the unity of duality. *Ann. Pure Appl. Log.*, 153(1-3):66–96, 2008. `doi:10.1016/j.apal.2008.01.001`.

The Functional Machine Calculus II: Semantics

Chris Barrett
Department of Computer Science, University of Bath, UK

Willem Heijltjes
Department of Computer Science, University of Bath, UK

Guy McCusker
Department of Computer Science, University of Bath, UK

Abstract

The Functional Machine Calculus (FMC), recently introduced by the second author, is a generalization of the lambda-calculus which may faithfully encode the effects of higher-order mutable store, I/O and probabilistic/non-deterministic input. Significantly, it remains confluent and can be simply typed in the presence of these effects.

In this paper, we explore the denotational semantics of the FMC. We have three main contributions: first, we argue that its syntax – in which both effects and lambda-calculus are realised using the same syntactic constructs – is semantically natural, corresponding closely to the structure of a Scott-style domain theoretic semantics. Second, we show that simple types confer strong normalization by extending Gandy's proof for the lambda-calculus, including a small simplification of the technique. Finally, we show that the typed FMC (without considering the specifics of encoded effects), modulo an appropriate equational theory, is a complete language for Cartesian closed categories.

2012 ACM Subject Classification Theory of computation

Keywords and phrases lambda-calculus, computational effects, denotational semantics, strong normalization

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.10

Related Version *Full Version*: `https://arxiv.org/abs/2211.13140`

Funding *Willem Heijltjes*: EPSRC Grant EP/R029121/1 *Typed lambda-calculi with sharing and unsharing.*

Acknowledgements Pierre Clairambault, Ugo Dal Lago, Anupam Das, Giulio Guerrieri, Jim Laird, Paul Levy, Simon Peyton Jones, John Power, Alex Simpson, Vincent van Oostrom.

1 Introduction

Almost without exception, modern programming languages support a combination of computational effects and higher-order mechanisms. Programmers and programming language theorists recognise that the effects and higher-order mechanisms are fundamentally different constructs, with radically different syntax and semantics: compare assignment and dereferencing operations with function definition and invocation, for example. In both operational and denotational accounts, the higher-order mechanism – typically expressed in some variant of λ-calculus – and the effects mechanisms are treated using distinct approaches, and indeed the combination of the two, not to mention the combination of multiple kinds of effects, requires careful handling.

In a previous paper [6] the second author introduced the Functional Machine Calculus (FMC), a compact programming language which eliminates these distinctions and supports higher-order effectful programming with a streamlined yet natural syntax and operational semantics. In this paper, we reprise the definition of the FMC and attempt to explain and

explore its construction and behaviour from a denotational perspective. Beginning with a domain-theoretic analysis of computation with stacks, we discover that the λ-calculus and effectful aspects of programming can be viewed as being of exactly the same kind, in fact entirely interchangeable. The syntax and operational semantics of the FMC embodies the operations naturally supported by the denotational semantics, while remaining programmer-friendly: the previous paper [6] demonstrates by example the wide range of effectful programs that can straightforwardly be expressed in this syntax.

The connection we exploit between domain-theoretic semantics of λ-calculus and an operational semantics based on stacks is not new. In [19], Streicher and Reus show that Scott's well-known D_∞ models motivate and explain the definition of Krivine's abstract machine for evaluating λ-terms. Scott's construction takes a domain D and builds a model of the λ-calculus as an appropriate limit of a sequence of domains, shown on the left below. Streicher and Reus observed that this construction may alternatively be regarded as taking the limit of the sequence on the right, where each D_n is recovered as $C_n \to D$, and D_∞ is $C_\infty \to D$.

$$\begin{array}{rcl@{\qquad\qquad}rcl} D_0 & = & D & C_0 & = & 1 \\ D_{n+1} & = & D_n \to D_n & C_{n+1} & = & C_n \times (C_n \to D) \end{array}$$

With this view, the limit C_∞ is a stream or (potentially infinite) stack of elements of D_∞, i.e. denotations of λ-terms; and such terms consume a stack. The equations defining the interpretations of terms in this model show that the operations of application and abstraction correspond directly to pushing and popping from the stack. The return domain D plays a very minor role in the semantics of λ-terms: in Streicher and Reus's view, it is the result type of continuations, and ordinary λ-terms never return. Our development in this paper adapts this in two ways. First, we choose the return domain D to be the domain of stacks. Thus a term can be regarded as a *stack transformer*, which immediately supports an operation of sequencing between terms, leading to a sublanguage of the FMC called the *sequential λ-calculus*. Second, we enrich the domain equation with the familiar monad for global state, and observe that – if states are also stacks – the resulting domain equation is one of multiple-stack transformers, with *no special status* afforded to the stack that implements λ-abstraction and application. Thus we arrive at the promised semantic explanation of the FMC, treating reader–writer style effects and higher-order mechanisms identically, and giving a denotational motivation for the stack-machine semantics.

We go on to study this calculus from a type-theoretic and category-theoretic perspective. The domain-theoretic model is untyped but, generalising the usual development for λ-calculus, a simple type system can be imposed which describes the shapes of the stacks being operated upon. Adapting Gandy's proof technique for λ-calculus, we show that the well-typed terms of the FMC are strongly normalizing. Further, we study the categorical properties of the calculus, viewing terms as morphisms between types, and discover that, up to a notion of contextual equivalence, well-typed programs form a Cartesian closed category. An equational theory is presented which refines this equivalence, and well-typed terms modulo this theory are shown to be a sound and complete language for CCCs.

The properties identified above – strong normalization, Cartesian closure, and the operational property of confluence established in [6] – are entirely expected of languages which are variants of typed λ-calculus, but perhaps surprising, or even shocking, in the setting of the effectful FMC. How can the combination of higher-order programming and effects, including state, input/output, nondeterminism and probability, retain properties of confluence and referential transparency?

We offer the following explanation. These properties are of the FMC as a general calculus, which remains close to the λ-calculus, independent of the encoding of effects. As explored in [6], the FMC is confluent, and for encoded reader/writer effects this manifests as reduction following the algebraic laws of Plotkin and Power [13]. The CCC semantics presented in this paper pertains to the FMC itself, and, we emphasize, is *not* a semantics of effects: when particular properties of the encoded effects are taken into account, the semantics will no longer be a CCC. We explore this in more detail in Section 6.

2 The Functional Machine Calculus

The Functional Machine Calculus (FMC) arises from an operational perspective on the λ-calculus, taking a simple call–by–name stack machine in the style of Krivine [7] as primary. The machine evaluates a term in the context of a stack, where application is a *push* (of the argument) and abstraction is a *pop*, binding the popped value to the abstracted variable. Since the stack machine is intended as an operational semantics, and not for implementation, for simplicity we use substitution rather than an environment. The FMC then introduces two natural generalizations.

Locations. The machine is generalized from one to multiple stacks or streams, indexed by a global set of *locations* A. In the calculus, abstraction and application are parameterized in A as pop- and push-actions on the associated stack. Stacks are homogeneous, but may be used to encode different reader/writer effects: an input stream (which may be non-deterministic or probabilistic), an output stream, or a global mutable variable.

Sequencing. The calculus is extended with sequential composition of terms, which gives their consecutive execution on the machine, and an identity term as the unit to composition, which is the empty instruction sequence on the machine, analogous to imperative *skip*. This generalizes the calculus from one of stack *consumers* to stack *transformers*, where a term may return multiple outputs to the stack, and gives control over execution, as demonstrated by the encoding of various reduction strategies including call–by–value and call–by–push–value (see [6]).

Locations are an innovation of the FMC [6], while sequencing is familiar from Hasegawa's κ-calculus [5, 17] and higher-order stack programming (see e.g. [10]). In the latter case, there are also certain similarities with Milner's action calculi [9]. These two innovations are implemented technically as follows. To emphasize the operational intuition as *push* and *pop*, application $M\,N$ will be written as $[N].\,M$, and abstraction $\lambda x.\,M$ as $\langle x\rangle.\,M$. These are subsequently parameterized in *locations* $a, b, c \in A$. *Sequencing* introduces a *nil* or *skip* term $\star$ and makes the variable construct a prefix $x.\,M$; sequential composition $M\,;N$ will be a defined operation.

▶ **Definition 1.** *The* Functional Machine Calculus *(*FMC*) is given by the grammar*

$$M,\, N,\, P \quad ::= \quad \star \;\mid\; x.\,M \;\mid\; [N]a.\,M \;\mid\; a\langle x\rangle.\,M$$

where from left to right the term *constructors are* nil*, a* (sequential) variable*, an* application *or* push action *on the location* a*, and an* abstraction *or* pop action *on the location* a *which binds* x *in* M*. Terms are considered modulo* α*-equivalence.*

We may omit the trailing $.\star$ from terms for readability. We will use a *main* location λ, omitted from the term notation, as the computation stack (as opposed to the stacks to interpret effects), on which (call–by–name) λ-terms embed. We will use constants as free variables; constant operators such as addition $+$ and conditionals if will operate on λ as well.

▶ **Example 2.** Consider the following example terms.

$$\mathsf{rnd}\langle x\rangle.\,[x].\,c\langle y\rangle.\,[y].\;+\,.\,\langle z\rangle.\,[z]c \qquad [\langle x\rangle.\,[x]\mathsf{out}.\,[x].\,[1].\,+].\,\langle f\rangle.\,[0].\,f.\,f.\,f$$

The first term increments a memory cell c with a random number. It pops x from the random generator stream rnd and pushes it to the main stack; pops y from the cell c and pushes that to the main stack as well; then $+$ adds the top two elements of the main stack x and y pushing back the result $x+y$; and this is popped as z and pushed back onto the cell c.

The second term counts up from zero to three, outputting $0, 1, 2$ and leaving 3 on the main stack. The subterm $\langle x\rangle.\,[x]\mathsf{out}.\,[x].\,[1].\,+$ pops x from the main stack and sends it to the output location out, and then $[x].\,[1].\,+$ leaves $x+1$ on the main stack. In the example, this term is pushed, popped as f, and called three times on zero.

▶ **Definition 3.** Composition $N\,;M$ *and* substitution $\{N/x\}M$ *are given by*

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}l}
\star\,;M & = & M & [P]a.\,N\,;M & = & [P]a.\,(N\,;M) & \\
x.\,N\,;M & = & x.\,(N\,;M) & a\langle x\rangle.\,N\,;M & = & a\langle x\rangle.\,(N\,;M) & (x\notin \mathsf{fv}(M))
\end{array}$$

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}l}
\{P/x\}\star & = & \star & \{P/x\}[N]a.\,M & = & [\{P/x\}N]a.\,\{P/x\}M & \\
\{P/x\}x.\,M & = & P\,;\{P/x\}M & \{P/x\}a\langle x\rangle.\,M & = & a\langle x\rangle.\,M & \\
\{P/x\}y.\,M & = & y.\,\{P/x\}M & \{P/x\}a\langle y\rangle.\,M & = & a\langle y\rangle.\,\{P/x\}M & (y\notin \mathsf{fv}(P))
\end{array}$$

where, in the last two cases, $x\neq y$*; both are capture-avoiding by the conditions* $x\notin\mathsf{fv}(M)$ *and* $y\notin\mathsf{fv}(P)$*, which can be satisfied by* α*-conversion.*

▶ **Lemma 4.** *Composition is associative and has unit* $\star$*.*

▶ **Definition 5.** *The* functional abstract machine *is given by the following data. A* stack *of terms* S *is defined below left, and written with the top element to the right. A* memory S_A *is a family of stacks or streams in* A*, defined below right.*

$$S \;::=\; \varepsilon \;\mid\; S{\cdot}M \qquad S_A \;::=\; \{\,S_a \mid a\in A\}$$

The notation $S_A;S_a$ *identifies the stack* S_a *within* S_A*. A* state *is a pair* (S_A,M) *of a memory and a term. The* transitions *or* steps *of the machine are given below left as top–to–bottom rules. A* run *of the machine is a sequence of steps, written as* $(S_A,M)\Downarrow(T_A,N)$ *or with a double line as below right.*

$$\frac{(\;S_A\;;\;S_a\;,\;[N]a.\,M\;)}{(\;S_A\;;\;S_a{\cdot}N\;,\;M\;)} \qquad \frac{(\;S_A\;;\;S_a{\cdot}N\;,\;a\langle x\rangle.\,M\;)}{(\;S_A\;;\;S_a\;,\;\{N/x\}M\;)} \qquad \frac{(\;S_A\;,\;M\;)}{(\;T_A\;,\;N\;)}$$

Constant operations such as addition $+$ and conditional if pop the required number of items from the main stack and reinstate their result, as per the rule given below left. The FMC then operates as a standard stack calculus: e.g. an arithmetic expression $1+((2+3)\times 4)$ is given as a term $[4].\,[3].\,[2].\,+\,.\,\times\,.\,[1].\,+$ which indeed returns 21. Formally, a constant operator $c^{n,m}$ of arity n,m is defined by a (partial) function $c^{n,m}$ from n input terms to m output terms, which generates the machine rule schema below right, where the output terms are put on the stack.

$$\frac{(\;S_A;S_\lambda\cdot 2\cdot 3\;,\;+\,.\,M\;)}{(\;S_A;S_\lambda\cdot 5\;\;,\;\;M\;)} \qquad \frac{(\;S_A;S_\lambda\cdot N_n\cdots N_1\;\;,\;c^{n,m}.\,M\;)}{(\;S_A;S_\lambda\cdot c^{n,m}(N_1,\ldots,N_n)\;,\;M\;)}$$

▶ **Example 6.** The first term of Example 2 has the following run of the machine, where the rnd location is initialized with a stream with 3 at the head, and c with the value 5.

$$
\begin{array}{llr}
(S_{\mathsf{rnd}} \cdot 3 \; ; \varepsilon_c \cdot 5 \; ; \varepsilon_\lambda & , & \mathsf{rnd}\langle x\rangle.\,[x].\,c\langle y\rangle.\,[y].\,+.\,\langle z\rangle.\,[z]c\ ) \\
(S_{\mathsf{rnd}} \; ; \varepsilon_c \cdot 5 \; ; \varepsilon_\lambda & , & [3].\,c\langle y\rangle.\,[y].\,+.\,\langle z\rangle.\,[z]c\ ) \\
(S_{\mathsf{rnd}} \; ; \varepsilon_c \cdot 5 \; ; \varepsilon_\lambda \cdot 3 & , & c\langle y\rangle.\,[y].\,+.\,\langle z\rangle.\,[z]c\ ) \\
(S_{\mathsf{rnd}} \; ; \varepsilon_c \; ; \varepsilon_\lambda \cdot 3 & , & [5].\,+.\,\langle z\rangle.\,[z]c\ ) \\
(S_{\mathsf{rnd}} \; ; \varepsilon_c \; ; \varepsilon_\lambda \cdot 3 \cdot 5 & , & +.\,\langle z\rangle.\,[z]c\ ) \\
(S_{\mathsf{rnd}} \; ; \varepsilon_c \; ; \varepsilon_\lambda \cdot 8 & , & \langle z\rangle.\,[z]c\ ) \\
(S_{\mathsf{rnd}} \; ; \varepsilon_c \; ; \varepsilon_\lambda & , & [8]c\ ) \\
(S_{\mathsf{rnd}} \; ; \varepsilon_c \cdot 8 \; ; \varepsilon_\lambda & , & \star\ )
\end{array}
$$

Beta-reduction in the λ-calculus, from the perspective of the machine, lets consecutive push and pop actions interact directly. With multiple stacks, these must be actions on the same location, while other locations may be accessed in-between. Informally, the reduction step will then be as follows, where the argument $[N]a$ and abstraction $a\langle x\rangle$ may be separated by actions $[P]b$ and $b\langle y\rangle$ with $a \neq b$: $[N]a \ldots a\langle x\rangle.\,M \;\to_\beta\; \ldots\{N/x\}M$. In addition, it must be avoided that any intervening $b\langle y\rangle$ captures y in N.

▶ **Definition 7.** *A* head context H *is a sequence of applications and abstractions terminating in a hole:*

$$H \;::=\; \{\} \;\mid\; [M]a.\,H \;\mid\; a\langle x\rangle.\,H$$

The term denoted $H.\,M$ *is given by replacing the hole* $\{\}$ *in* H *with* M*, where a binder* $a\langle x\rangle$ *in* H *captures in* M*. The* binding variables $\mathsf{bv}(H)$ *of* H *are those variables* x *where* H *is constructed over* $a\langle x\rangle$*. The set of* locations *used in a term or context is denoted* $\mathsf{loc}(M)$ *respectively* $\mathsf{loc}(H)$*. Then* beta-reduction *and* eta-reduction *are defined respectively by the rewrite rule schemas below, where* $a \notin \mathsf{loc}(H)$ *for both reduction rules,* $\mathsf{bv}(H) \cap \mathsf{fv}(N) = \varnothing$ *for the* β*-rule, and* $x \notin \mathsf{bv}(H)$ *for the* η*-rule. Both reductions are closed under all contexts.*

$$[N]a.\,H.\,a\langle x\rangle.\,M \;\to_\beta\; H.\{N/x\}M \qquad\qquad a\langle x\rangle.\,H.\,[x]a.\,M \;\to_\eta\; H.\,M \quad (x \notin \mathsf{fv}(M))$$

We now clarify the relationship between beta reduction and the machine. Evaluation of a term M on the machine, given sufficient inputs in the form of a stack of terms $N_1 \cdots N_n$, begins in the state $(N_1 \cdots N_n, M)$. The machine then implements a particular (weak) reduction strategy, with each *pop* transition corresponding to a beta-reduction of the term $[N_1] \ldots [N_n].\,M$ corresponding to the machine state under evaluation.

In Section 6, we further introduce a notion of observational equivalence based on the machine, dubbed *machine equivalence*, where terms are considered equivalent if they send equivalent inputs to equivalent outputs. This is shown to validate the beta and eta equations in general.

The two generalizations *locations* and *sequencing* are independent, and the two calculi that have one but not the other are of independent interest.

- The *poly* λ*-calculus* has only *locations*, and is given by the fragment below.

$$M, N \;::=\; x.\star \;\mid\; [N]a.\,M \;\mid\; a\langle x\rangle.\,M$$

- The *sequential* λ*-calculus* has only *sequencing*, and is given by the fragment below.

$$M, N \;::=\; \star \;\mid\; x.\,M \;\mid\; [N].\,M \;\mid\; \langle x\rangle.\,M$$

▶ **Example 8.** To demonstrate how the FMC encodes both effects and evaluation strategies, we consider the following (standard) call–by–value λ-calculus with reader/writer effects. (We assume familiarity with the operational semantics of effects and call–by–value λ-calculus; for an introduction see Winskel [20].)

$$\begin{array}{rll} M, N, P ::= x \mid M\,N \mid \lambda x.M & & \textit{λ-calculus} \\ \mid \mathsf{read} \mid \mathsf{write}\ N; M & & \textit{input/output} \\ \mid c := N; M \mid\ !c & & \textit{state update and lookup} \\ \mid N \oplus M \mid N + M & & \textit{probabilistic and non-deterministic sum} \end{array}$$

The full calculus is encoded into the FMC as follows. We let A comprise the main location λ, a location in for input, out for output, rnd and nd for probabilistically and non-deterministically generated streams of boolean values $(\top, \bot)$, and one location for each global memory cell. A term M encodes as M_v below, where $N + M$ encodes like $N \oplus M$ but with the stream nd.

$$\begin{array}{rcl} x_v = [x] & \qquad & \mathsf{read}_v = \mathsf{in}\langle x\rangle.\,[x] \\ (\lambda x.\,M)_v = [\langle x\rangle.\,M_v] & & !c_v = c\langle x\rangle.\,[x]c.\,[c] \\ (M\,N)_v = N_v\,;M_v\,;\langle x\rangle.\,x & & \end{array} \qquad \begin{array}{rcl} (\mathsf{write}\ N\,;M)_v = N_v.\,\langle x\rangle.\,[x]\mathsf{out}.\,M_v \\ (c := N\,;M)_v = N_v.\,\langle x\rangle.\,c\langle\_\rangle.\,[x]c.\,M_v \\ (N \oplus M)_v = \mathsf{rnd}\langle x\rangle.\,[N].\,[M].\,[x].\,\mathsf{if} \end{array}$$

We leave it to the reader to confirm that the interpretation generates the correct evaluation behaviour, and conclude the example with the encoding and reduction of the following term.

$$\begin{array}{rcl} ((\lambda f.f(f\,0))\,(\lambda x.\mathsf{write}\ x; !c))_v & = & [\langle x\rangle.\,\underline{[x].\,\langle v\rangle.}\,[v]\mathsf{out}.\,c\langle y\rangle.\,[y]c.\,[y]].\,\langle f\rangle.\,[0].\,[f].\,\langle z\rangle.\,z.\,[f].\,\langle w\rangle.\,w \\ & \to & [\langle x\rangle.\,[x]\mathsf{out}.\,c\langle y\rangle.\,[y]c.\,[y]].\,\langle f\rangle.\,[0].\,[f].\,\langle z\rangle.\,z.\,\underline{[f].\,\langle w\rangle.}\,w \\ & \to & [\langle x\rangle.\,[x]\mathsf{out}.\,c\langle y\rangle.\,[y]c.\,[y]].\,\langle f\rangle.\,[0].\,\underline{[f].\,\langle z\rangle.}\,z.\,f \\ & \to & \underline{[\langle x\rangle.\,[x]\mathsf{out}.\,c\langle y\rangle.\,[y]c.\,[y]].\,\langle f\rangle.}\,[0].\,f.\,f \\ & \to & [0].\,\langle x\rangle.\,[x]\mathsf{out}.\,c\langle y\rangle.\,\underline{[y]c.}\,[y].\,\langle z\rangle.\,[z]\mathsf{out}.\,\underline{c\langle w\rangle.}\,[w]c.\,[w] \\ & \to & [0].\,\langle x\rangle.\,[x]\mathsf{out}.\,c\langle y\rangle.\,\underline{[y].\,\langle z\rangle.}\,[z]\mathsf{out}.\,[y]c.\,[y] \\ & \to & \underline{[0].\,\langle x\rangle.}\,[x]\mathsf{out}.\,c\langle y\rangle.\,[y]\mathsf{out}.\,[y]c.\,[y] \\ & \to & [0]\mathsf{out}.\,c\langle y\rangle.\,[y]\mathsf{out}.\,[y]c.\,[y] \end{array}$$

3 Domain-theoretic semantics

Our aim in this section is to show that the syntax and stack-machine semantics of the FMC may be further justified by consideration of a simple domain-theoretic semantics. We work in the category **Cppo** of complete partial orders (with least-element) and continuous functions. We show that a domain equation for stack-transformers directly supports the operations of the sequential λ-calculus, directly extending a Scott-style semantics of λ-calculus. The step from sequential lambda-calculus to FMC is then nothing more than the incorporation of the state monad in the original domain equation. Our development has much in common with Streicher and Reus's work [19]; the key step in the move to the FMC is to allow computations to return a result – a new stack – which may be further operated upon by later computations, yielding the sequencing operation of the FMC.

We begin by constructing a domain D to interpret terms. A stack can be seen as an element of $D^{\mathbb{N}}$. A term takes a stack and, after computation, returns a new stack as its result. We suppose that computations are modelled using an (unspecified) strong monad T on **Cppo**; for now think of T as the lifting monad. Then a term would be an element of a domain satisfying $D \cong D^{\mathbb{N}} \to TD^{\mathbb{N}}$. This domain equation can be solved by standard

techniques. Kleisli function composition gives rise to a sequencing operation $D \times D \to D$ which is associative, and forms a monoid with unit element given by the unit of the monad. We will write $(d_1.d_2)$ for the composition of two elements of D using this operation.

Observe that D is a *reflexive object* in **Cppo** and hence provides a model of the λ-calculus:

$$D \;\cong\; D^{\mathbb{N}} \to TD^{\mathbb{N}} \;\cong\; (D^{\mathbb{N}} \times D) \to TD^{\mathbb{N}} \;\cong\; D \to (D^{\mathbb{N}} \to TD^{\mathbb{N}}) \;\cong\; D \to D.$$

We briefly spell out the semantics of λ-calculus induced by this model. Let ρ range over *environments*: functions from the set of variables to D. We use s to range over $D^{\mathbb{N}}$; $(s \cdot d)$ denotes the stack resulting from pushing d onto s. We shall elide the isomorphism $D \cong D^{\mathbb{N}} \to TD^{\mathbb{N}}$. For any term M and environment ρ we define $[\![M]\!]\rho \in D$ (equivalently, a function $D^{\mathbb{N}} \to TD^{\mathbb{N}}$) as follows.

$$[\![x.\star]\!]\rho = \rho x \qquad [\![[N].M]\!]\rho\ s = [\![M]\!]\rho\ (s \cdot [\![N]\!]\rho) \qquad [\![\langle x\rangle.M]\!]\rho\ (s \cdot d) = [\![M]\!]\rho'\ s$$

where $\rho'(x) = d$ and $\rho'(y) = \rho(y)$ for $y \neq x$.

These definitions show immediately that application is interpreted by pushing the argument onto the stack, and abstraction by popping a term from the stack. Thus this standard λ-calculus model directly justifies the machine semantics. It extends to the sequential λ-calculus by defining

$$[\![\star]\!]\rho = \eta \qquad [\![x.M]\!]\rho = (\rho(x)\ .\ [\![M]\!]\rho),$$

where η is the unit of the monad (and of the monoid).

Thanks to the compositionality of the semantics we can readily prove:

▶ **Lemma 9.** $[\![M]\!]\rho = [\![M]\!]\rho'$ *if* $\forall x \in \mathsf{fv}(M)\ \rho(x) = \rho'(x)$.

As a consequence of this Lemma, we may speak of $[\![M]\!]$, independent of ρ, when M is closed.

We extend the semantics to stacks of closed terms. Suppose the monad T is lifting. A stack S denotes an element of $D^{\mathbb{N}}$:

$$[\![\varepsilon]\!] = \bot \qquad [\![S \cdot M]\!] = \langle [\![S]\!], [\![M]\!]\rangle$$

where we elide the isomorphism $D^{\mathbb{N}} \cong D^{\mathbb{N}} \times D$.

Thanks to the direct correspondence between the semantic equations and the machine transitions, we have:

▶ **Lemma 10** (Soundness). *Whenever* $(S, M) \Downarrow (T, N)$ *(with all terms closed),* $[\![M]\!]([\![S]\!]) = [\![N]\!]([\![T]\!])$.

▶ **Theorem 11** (Adequacy). *If* $[\![M]\!]([\![S]\!]) \neq \bot$ *then there exists a* T *such that* $(S, M) \Downarrow (T, \star)$.

Proof. The proof of this statement follows readily from the existence of three relations: a relation between elements of D and closed terms; a relation between semantic streams in $D^{\mathbb{N}}$ and streams S; and a relation between computations in TD and machine states (S, M). We write $\lhd$ for each of these relations, relying on the types to disambiguate. The relations are required to satisfy the following conditions:

$$\begin{aligned} d \lhd M \quad &\text{iff} \quad \forall \sigma \in D^{\mathbb{N}}, \sigma \lhd S \Rightarrow d(\sigma) \lhd (S, M) \\ \sigma \lhd S \quad &\text{iff} \quad \forall i. \sigma_i \lhd S_i \\ k \lhd (S, M) \quad &\text{iff} \quad k = \mathsf{lift}(\sigma) \Rightarrow (S, M) \Downarrow (T, \star) \text{ and } \sigma \lhd T. \end{aligned}$$

$$\frac{}{\Gamma \vdash \star : \overleftarrow{\tau}_A \Rightarrow \vec{\tau}_A}\ \star \qquad \frac{}{\Gamma,\ x : \alpha \vdash x : \alpha}\ \text{base} \qquad \frac{\Gamma,\ x : \rho \vdash M : \overleftarrow{\sigma}_A \Rightarrow \vec{\tau}_A}{\Gamma \vdash a\langle x\rangle.\, M : a(\rho)\ \overleftarrow{\sigma}_A \Rightarrow \vec{\tau}_A}\ \text{abs}$$

$$\frac{\Gamma \vdash N : \rho \qquad \Gamma \vdash M : a(\rho)\ \overleftarrow{\sigma}_A \Rightarrow \vec{\tau}_A}{\Gamma \vdash [N]a.\, M : \overleftarrow{\sigma}_A \Rightarrow \vec{\tau}_A}\ \text{app} \qquad \frac{\Gamma,\ x : \overleftarrow{\rho}_A \Rightarrow \vec{\sigma}_A \vdash \quad M : \overleftarrow{\sigma}_A\ \overleftarrow{\tau}_A \Rightarrow \vec{\upsilon}_A}{\Gamma,\ x : \overleftarrow{\rho}_A \Rightarrow \vec{\sigma}_A \vdash x.\, M : \overleftarrow{\rho}_A\ \overleftarrow{\tau}_A \Rightarrow \vec{\upsilon}_A}\ \text{var}$$

Figure 1 Typing rules for the Functional Machine Calculus.

These conditions cannot be used as a definition of the relations $\lhd$, for example as a fixed point of an operator on such relations, because the first clause contains a negatively-occurring usage of $\lhd$. Nevertheless the existence of such relations can be established using standard techniques of denotational semantics. Pitts's work [11] gives an elegant general theory which enables the construction of such relations.

Once $\lhd$ has been shown to exist, a straightforward induction on syntax establishes that for any term M, and any finite stream S, we have

$$[\![M]\!] \lhd M \qquad [\![S]\!] \lhd S \qquad [\![M]\!]([\![S]\!]) \lhd (S, M)$$

from which computational adequacy immediately follows. ◀

Note that our soundness and adequacy results are expressed in terms of the stack-machine evaluation mechanism. It is also the case that the denotational semantics validates the beta- and eta-laws of Definition 7, but our point in this section is to emphasise that the stack-machine semantics can be seen as an implementation of a natural denotational model.

Our denotational semantics so far gives an account of the sequential λ-calculus. To extend to the FMC, we replace the lifting monad with the *state monad* $TX = St \to (St \times X)_\perp$, where St is a domain of states. Our domain equation becomes

$$D \cong D^{\mathbb{N}} \to (St \to (St \times D^{\mathbb{N}})_\perp) \cong St \times D^{\mathbb{N}} \to (St \times D^{\mathbb{N}})_\perp$$

If we let $St = D^{\mathbb{N}}$, so that the values in the state are stacks, this is a domain equation for "two-stack transformers". As above, this is a reflexive object, now in *two distinct ways* depending on which stack is used to interpret the arguments. This is exactly the FMC with two locations; extension to any finite set of locations is handled similarly, and the soundness and adequacy results may be proved in the same way. As we emphasized in the introduction, this semantics has the remarkable property that the stack used to interpret the operations of the λ-calculus has exactly the same status as that used to interpret state, and it is merely convention that distinguishes the two. This is precisely the point of view embodied by the novel syntax and operational semantics of the FMC.

4 Simple types

Simple types for the FMC [6] describe the input/output behaviour of the stack machine. The type system has three levels, mirroring the syntactic categories of the machine: types τ for terms M, type vectors $\vec{\tau}$ (or *stack types*) for stacks S, and location-indexed families of type vectors $\vec{\tau}_A$ (or *memory types*) for memories S_A. A function type is then an implication between an input memory type and an output memory type.

▶ **Definition 12.** FMC-types ρ, σ, τ, υ *over a set of* base types Σ *are given by:*

$$\tau \ ::= \ \alpha \in \Sigma \ \mid \ \vec{\sigma}_A \Rightarrow \vec{\tau}_A \qquad \vec{\tau}_A \ ::= \ \{\vec{\tau}_a \mid a \in A\} \qquad \vec{\tau} \ ::= \ \tau_1 \dots \tau_n$$

Equivalently, one may view a function type as an implication between two vectors of location-indexed types, considered modulo the permutation of types on different locations.

$$a_1(\sigma_1) \dots a_n(\sigma_n) \ \Rightarrow \ b_1(\tau_1) \dots b_m(\tau_m) \qquad\qquad a(\sigma)\, b(\tau) \sim b(\tau)\, a(\sigma)$$

We introduce the following notation, which will enable us to write types also in the manner above. The *empty* type vector is ε, and the empty memory type ε_A. A *singleton* memory type $a(\vec{\tau})$ is empty at every location except a, where it has $\vec{\tau}$: $a(\vec{\tau})_a = \vec{\tau}$ and $a(\vec{\tau})_b = \varepsilon$ for $a \neq b$. A singleton $\lambda(\vec{\tau})$ on the main location λ may be written as $\vec{\tau}$. *Concatenation* of type vectors is denoted by juxtaposition and the *reverse* of a type vector $\vec{\tau} = \tau_1 \dots \tau_n$ is written $\overleftarrow{\tau} = \tau_n \dots \tau_1$. This extends point-wise to families, so $\vec{\sigma}_A \vec{\tau}_A = \{\vec{\sigma}_a \vec{\tau}_a \mid a \in A\}$ and $\overleftarrow{\tau}_A = \{\overleftarrow{\tau}_a \mid a \in A\}$.

▶ **Definition 13.** *A* judgement $\Gamma \vdash M : \tau$ *is a typed term in a* context $\Gamma = x_1 : \tau_1, \dots, x_n : \tau_n$, *a finite function from variables to types. The typing rules for the FMC are given in Figure 1.*

▶ **Example 14.** The terms from Example 2 can be typed as follows, where $\mathbb{Z}$ is a base type of integers. Recall that the first term adds a random number to the cell c, and the second sends the numbers from zero to two to output, leaving the number three on the main stack.

$$\mathsf{rnd}\langle x\rangle.\,[x].\,c\langle y\rangle.\,[y].\ +\ .\,\langle z\rangle.\,[z]c : \mathsf{rnd}(\mathbb{Z})\, c(\mathbb{Z}) \Rightarrow c(\mathbb{Z})$$
$$[\langle x\rangle.\,[x]\mathsf{out}.\,[x].\,[1].\,+].\,\langle f\rangle.\,[0].\,f.\,f.\,f \ : \ \Rightarrow \mathsf{out}(\mathbb{Z}\,\mathbb{Z}\,\mathbb{Z})\,\mathbb{Z}$$

Note, there are two typing rules for variables: one for variables of base type, and one for variables of higher type. The simply-typed FMC satisfies the subject reduction property [6], which is implicitly used in the following section.

5 Strong normalization

We will show that reduction for the simply-typed FMC is strongly normalizing (SN). Our proof is a variant of Gandy's for the simply-typed λ-calculus [3]. Gandy's proof interprets terms in domains of strictly ordered, strict monotone functionals: the base domain is $\mathbb{N}^{<} = (\mathbb{N}, <_{\mathbb{N}})$, and if $X = (|X|, <_X)$ and $Y = (|Y|, <_Y)$ are domains then so is $X \to Y$ where

$$\begin{array}{rcl} |X \to Y| & = & \{f \in Y^X \mid \forall x, x' \in X.\ x <_X x' \implies f(x) <_Y f(x')\} \\ f <_{X\to Y} g & \iff & \forall x \in X.\ f(x) < g(x)\ . \end{array}$$

The interpretation takes types to domains and terms of a given type to elements of that domain. The domains are well-founded and the interpretation of terms is such that it decreases on reduction, giving SN. One may further collapse a functional to a natural number to give an overestimate of the longest reduction sequence of a term. The literature has several variants on this proof, including one by De Vrijer that calculates longest reduction sequences exactly [2]; see also [18, 15, 14].

We introduce a (to the best of our knowledge) new variant, that avoids the domain of *strict* functionals and instead interprets terms in the – more standard – domain of (non-strict) monotone functionals, as above but with $\leq$, generated from $\mathbb{N}^{\leq} = (\mathbb{N}, \leq_{\mathbb{N}})$ with $\to$. This domain is not well-founded, but our interpretation of terms ensures that when functionals are collapsed to a natural number, this strictly decreases upon reduction, giving SN.

The technical difference is small and subtle. Gandy's proof originates in ΛI, where abstracted variables must occur, and hence the interpretation of an abstraction $\lambda x.M$ is naturally strict: the argument to x always contributes to the overall interpretation. To generalize to the λ-calculus, where x need not occur in $\lambda x.M$, a construction is introduced to nevertheless measure the argument to x, so that the functional for $\lambda x.M$ remains strictly monotone. The literature has several further such constructions [4].

This solves the challenge of accounting for reduction in terms that will be discarded, common in SN proofs. In our proof, instead we account for such terms when they are supplied as arguments: for a term $M\,N$ we increment the overall measure with that for N, measuring potential reduction in N even if it will be discarded by M. An abstraction $\lambda x.M$ may then be interpreted as a standard monotone functional, avoiding strictness.

To build our domains, we use the $\to$ construction above, as well as the product of domains $X \times Y$ and an indexed product $\Pi_{a\in A}\, X_a$, defined in the expected way, as follows. Note, we will omit to work with base types in this section, so the base case is given by $(\Rightarrow)$.

$$|X \times Y| = |X| \times |Y| \qquad (x,y) \leq_{X\times Y} (x',y') \iff x \leq_X x' \wedge y \leq_Y y'$$
$$|\Pi_{a\in A}\, X_a| = \Pi_{a\in A}\, |X_a| \qquad x \leq_{\Pi_{a\in A} X_a} x' \iff \forall a \in A.\ x_a \leq_{X_a} x'_a$$

▶ **Definition 15.** *The* interpretation *of an FMC-type τ is the domain $[\![\tau]\!]$ given by:*

$$[\![\overleftarrow{\sigma_A} \Rightarrow \overrightarrow{\tau_A}]\!] = [\![\overrightarrow{\sigma_A}]\!] \to \mathbb{N}^{\leq} \times [\![\overrightarrow{\tau_A}]\!] \qquad [\![\tau_1 \ldots \tau_n]\!] = [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!] \qquad [\![\overrightarrow{\tau_A}]\!] = \Pi_{a\in A}\, [\![\overrightarrow{\tau}_a]\!]$$

It is worth observing that for the simple types of the λ-calculus, as embedded in the FMC, these domains are the natural ones. Briefly (see [6] for details), a simple type $\tau_1 \to \ldots \to \tau_n \to o$ embeds as the FMC-type $\tau_1 \ldots \tau_n \Rightarrow$ with the domain $[\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!] \to \mathbb{N}^{\leq}$, which is the expected one modulo Currying.

▶ **Definition 16.** *The least element of a domain $[\![\tau]\!]$ is written 0_τ. The* collapse *function $\lfloor - \rfloor_\tau : [\![\tau]\!] \to \mathbb{N}$ takes a functional to a natural number by providing a least element as input and discarding all other output: $\lfloor f \rfloor_{\overleftarrow{\sigma_A} \Rightarrow \overrightarrow{\tau_A}} = \pi_1(f(0_{\overrightarrow{\sigma_A}}))$.*

We will interpret terms such that if $M : \tau$ then $[\![M]\!] \in [\![\tau]\!]$, and if $M \twoheadrightarrow N$ at type τ then both $[\![M]\!] \geq_{[\![\tau]\!]} [\![N]\!]$ and $\lfloor [\![M]\!] \rfloor > \lfloor [\![N]\!] \rfloor$, to give SN. We introduce the following notation. To interpret terms in a context Γ, let a *valuation* v on Γ be a function assigning to each variable $x : \tau$ in Γ a value $v(x) \in [\![\tau]\!]$. The valuation $v\{x \leftarrow t\}$ assigns t to x and otherwise behaves as v. We write elements of product domains as vectors $(t_1, \ldots, t_n)$, and will elide the isomorphisms for associativity and unitality so that concatenation of s and t may be written (s,t). Concatenation lifts to indexed products pointwise: $(s,t)_a = (s_a, t_a)$. For $t \in [\![\tau]\!]$ we have a singleton $a(t) \in [\![a(\tau)]\!]$ where $a(t)_a = t$ and $a(t)_b = ()$ for $b \neq a$.

▶ **Definition 17.** *For a term $\Gamma \vdash M : \tau$ and valuation v on Γ, we inductively define the* interpretation *$[\![\Gamma \vdash M : \tau]\!]_v \in [\![\tau]\!]$ as follows, omitting Γ for compactness.*

$$\begin{aligned}
[\![\star : \overleftarrow{\tau_A} \Rightarrow \overrightarrow{\tau_A}]\!]_v(t) &= (0,t) \\
[\![x.\, M : \overleftarrow{\rho_A}\, \overleftarrow{\sigma_A} \Rightarrow \overrightarrow{\tau_A}]\!]_v(s,r) &= (n+m,t) \quad \textit{where } (n,u) = v(x : \overleftarrow{\rho_A} \Rightarrow \overrightarrow{\upsilon_A})(r) \\
&\qquad\qquad\qquad\quad (m,t) = [\![M : \overrightarrow{\upsilon_A}\, \overleftarrow{\sigma_A} \Rightarrow \overrightarrow{\tau_A}]\!]_v(s,u) \\
[\![a\langle x\rangle.\, M : a(\rho)\, \overleftarrow{\sigma_A} \Rightarrow \overrightarrow{\tau_A}]\!]_v(s, a(r)) &= (1+m,t) \quad \textit{where } (m,t) = [\![M : \overleftarrow{\sigma_A} \Rightarrow \overrightarrow{\tau_A}]\!]_{v\{x\leftarrow r\}}(s) \\
[\![[N]a.\, M : \overleftarrow{\sigma_A} \Rightarrow \overrightarrow{\tau_A}]\!]_v(s) &= (1+m+\lfloor f \rfloor, t) \quad \textit{where } f = [\![N : \rho]\!]_v \\
&\qquad\qquad\qquad\quad (m,t) = [\![M : a(\rho)\, \overleftarrow{\sigma_A} \Rightarrow \overrightarrow{\tau_A}]\!]_v(s, a(f))
\end{aligned}$$

We write $[\![\Gamma \vdash M : \tau]\!]$ for $[\![\Gamma \vdash M : \tau]\!]_v$ with v the least valuation $v(x : \tau) = 0_\tau$, and may abbreviate $[\![\Gamma \vdash M : \tau]\!]_v$ to $[\![M : \tau]\!]_v$ or $[\![M]\!]_v$.

▶ Remark 18. In this definition, the application case $[\![[N]a.\,M]\!]_v(s) = (1{+}m{+}\lfloor f \rfloor)$ adds the value $\lfloor f \rfloor$ to account for reduction inside the argument N. Further, both it and the abstraction case $[\![a\langle x\rangle.\,M]\!]_v(s, a(r)) = (1{+}m, t)$ add 1 to count redexes, so that a reduction step reduces the overall measure by (at least) 2. It would suffice to count only abstractions or only applications, but the choice to count both is so that we count steps of the stack machine. We observe the following: for the alternative interpretation that omits to count $\lfloor f \rfloor$, and instead has $[\![[N]a.\,M]\!]_v(s) = (1{+}m)$, the collapsed interpretation $\lfloor [\![M]\!] \rfloor$ measures the exact length of machine runs for M. This observation provides the proof with an operational intuition: terms are strongly normalizing because a) types guarantee termination of the machine [6, Theorem 3.12], and b) reduction shortens the length of machine runs.

For the remainder of the proof, we will give an overview by stating the main lemmata. Each follows by a straightforward induction on typing derivations. First, for the interpretation $[\![-]\!]$ to be well-defined, the construction for each term must be shown to preserve monotonicity. We will do so in the following lemma. For valuations v and w over Γ, let $v \leq w$ denote that $v(x) \leq_{[\![\tau]\!]} w(x)$ for all $x : \tau$ in Γ.

▶ **Lemma 19.** *For all terms $\Gamma \vdash M : \tau$ and valuations $v \leq w$ over Γ, we have that:*

1. $[\![M]\!]_v \in [\![\tau]\!]$
2. $[\![M]\!]_v \leq_{[\![\tau]\!]} [\![M]\!]_w$.

For the next steps, we first need that the interpretation commutes with sequential composition $M\,;N$ and substitution $\{N/x\}M$. Then, we show that reduction (non-strictly) decreases the interpretation, and strictly decreases the collapsed interpretation.

▶ **Lemma 20.** *For terms $\Gamma \vdash M : \overleftarrow{\sigma}_A \overleftarrow{\tau}_A \Rightarrow \vec{v}_A$ and $\Gamma \vdash N : \overleftarrow{\rho}_A \Rightarrow \vec{\sigma}_A$ and valuation v on Γ,*

$$[\![N\,;M]\!]_v(t, r) = (i + j, u) \quad \text{where} \quad [\![N]\!]_v(r) = (i, s) \quad \text{and} \quad [\![M]\!]_v(t, s) = (j, u)\ .$$

▶ **Lemma 21.** *For terms $\Gamma \vdash N : \sigma$ and $\Gamma, x : \sigma \vdash M : \tau$ and valuation v on Γ,*

$$[\![\{N/x\}M]\!]_v = [\![M]\!]_{v\{x \leftarrow [\![N]\!]_v\}}\ .$$

▶ **Lemma 22.** *If $\Gamma \vdash M \longrightarrow N : \tau$ then $[\![M]\!]_v \geq_{[\![\tau]\!]} [\![N]\!]_v$ for every valuation v on Γ.*

▶ **Lemma 23.** *If $\Gamma \vdash M \longrightarrow N : \overleftarrow{\sigma}_A \Rightarrow \vec{\tau}_A$ then $\pi_1([\![M]\!]_v(s)) >_{\mathbb{N}} \pi_1([\![N]\!]_v(s))$ for every $s \in [\![\vec{\sigma}_A]\!]$ and valuation v on Γ.*

The last lemma then immediately gives the strong normalization result.

▶ **Theorem 24** (Strong Normalization). *Simply-typed FMC-terms are strongly normalizing with respect to beta-reduction.*

Proof. By Lemma 23 if $\Gamma \vdash M \longrightarrow N : \tau$ then $\lfloor [\![M]\!] \rfloor >_{\mathbb{N}} \lfloor [\![N]\!] \rfloor$, so that $\lfloor [\![M]\!] \rfloor$ gives a bound for the length of any reduction path from M. ◀

Note that it is easy to extend this result to include eta-reduction: since eta-reduction does not increase the measure, and is clearly strongly normalizing by itself (the size of the term decreases), we can interleave each beta-reduction step with an arbitrary number of eta-reduction steps without affecting strong normalization.

6 Categorical semantics

We give the categorical view on the FMC in three layers:

- terms with composition $N\,;M$ and unit $\star$ form a category;
- terms modulo $\beta\eta$-equivalence form a *premonoidal* category [16];
- terms modulo an appropriate equational theory form a complete language for Cartesian closed categories;

we then show that *machine equivalence*, where terms are equivalent if they display the same input/output behaviour on the machine, validates the final equational theory.

The idea that a calculus with effects should semantically be a CCC may be surprising, so we will first motivate what the semantics does and does not capture. Firstly, and most importantly, the semantics we give here is one of the pure FMC, and emphatically *not* a semantics *of effects*: it ignores that, for instance, *input* only has a *pop* operation but no *push*, and that *state* locations would be restricted to a stack of depth one (at most). Imposing these constraints will cause the CCC semantics to break down, as we will demonstrate later in the section.

Secondly, the situation is analogous to the encoding of monadic effects in simply-typed λ-calculus, where for instance *state* encodes as the monad $S \to (- \times S)$. In that case, too, the semantics remains a CCC, despite the possibility of encoding effects.

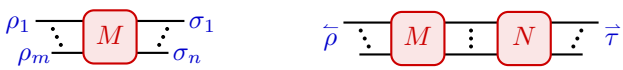
Thirdly, the two generalizations of the FMC, *locations* and *sequencing*, remain close enough to the λ-calculus that simple types allow to collapse them back onto a CCC semantics. *Locations* give multiple copies of the λ-calculus, but because the types give the entire memory, the semantics may combine the different indexed stacks into one, projecting the multiple copies onto a single λ-calculus. *Sequencing* gives control over evaluation and allows us to encode various reduction strategies, but the point of the denotational perspective is precisely to collapse any computational behaviour, and only consider the input/output behaviour of a term.

The purpose of our CCC semantics is to demonstrate that the simply-typed FMC is an *operational* refinement of the lambda-calculus, but not a *denotational* one. The FMC allows to express *how* computation takes place: what reduction strategy is used, whether inputs are passed as function arguments or via mutable store, when the random generator is consulted, *etc.* The denotational perspective then collapses these distinctions, demonstrating that we remain firmly in the domain of higher-order functional computation, despite the ability to encode effects.

The plain category

For simplicity we will work in the sequential λ-calculus. The arguments generalize straightforwardly to the case of the FMC, and the details of this case are to be given in the first author's Ph.D. thesis. The objects are then type vectors $\vec{\tau}$ and morphisms in $\vec{\sigma} \longrightarrow \vec{\tau}$ will be *closed* terms $M : \overleftarrow{\sigma} \Rightarrow \vec{\tau}$ modulo the given equivalence.

A term $M : \rho_1 \dots \rho_m \Rightarrow \sigma_n \dots \sigma_1$ may be represented by a string diagram as below, left. The wires represent the input and output stacks, with the first element at the top. Strict composition of terms $M : \overleftarrow{\rho} \Rightarrow \vec{\sigma}$ and $N : \overleftarrow{\sigma} \Rightarrow \vec{\tau}$ into $M\,;N : \overleftarrow{\rho} \Rightarrow \vec{\tau}$, given below, right.



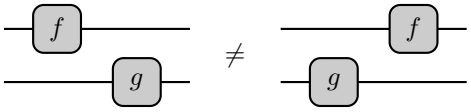
Analogous to the notation for type vectors, we introduce the following notation. We use vector notation for variables, $\vec{x} = x_1 \dots x_n$, and reverse a vector by a left-pointing arrow: if $\vec{x}$ is as before, then $\overleftarrow{x} = x_n \dots x_1$. Concatenation of vectors is by juxtaposition. We

lift the notation to sequences of abstractions and applications, but only for variables: if $\vec{x}$ is as above, then $\langle\vec{x}\rangle.\, N = \langle x_n\rangle \dots \langle x_1\rangle.\, N$ and $[\vec{x}].\, N = [x_1] \dots [x_n].\, N$. Vector notation is extended to contexts as $x_1 : \tau_1, \dots, x_n : \tau_n \;=\; \vec{x} : \vec{\tau}$ and simultaneous substitutions as $\{S/\vec{x}\} = \{M_1/x_1, \dots, M_n/x_n\}$, where $S = \varepsilon \cdot M_1 \cdots M_n$.

The first, plain category is then given by Lemma 4.

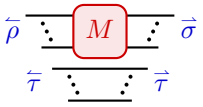
The premonoidal category

A *premonoidal* category [16], like a monoidal category, describes string diagrams, but with a *sequential* element: a premonoidal product $\otimes$ has no *parallel* composition $f \otimes g$, while $(f \otimes \mathsf{id}); (\mathsf{id} \otimes g)$ and $(\mathsf{id} \otimes g); (f \otimes \mathsf{id})$ are distinct.

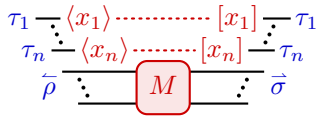


Formally, a premonoidal product is a binary operation on objects $X \otimes Y$ that is a functor in each argument, $- \otimes X$ and $X \otimes -$, but need not be a bifunctor $- \otimes -$. In the FMC, the action on objects is concatenation, $\vec{\sigma} \otimes \vec{\tau} \triangleq \vec{\sigma}\vec{\tau}$, with the first element at the top of the stack, and the unit given by ε. Both actions on morphisms are given below for $M : \overleftarrow{\rho} \Rightarrow \vec{\sigma}$, where $\overleftarrow{x} : \overleftarrow{\tau}$.

$$M \otimes \vec{\tau} :\; \vec{\tau} \otimes \vec{\rho} \longrightarrow \vec{\tau} \otimes \vec{\sigma} \;\triangleq\; M : \overleftarrow{\rho}\overleftarrow{\tau} \Rightarrow \vec{\tau}\vec{\sigma}$$



$$\vec{\tau} \otimes M :\; \vec{\rho} \otimes \vec{\tau} \longrightarrow \vec{\sigma} \otimes \vec{\tau} \;\triangleq\; \langle\overleftarrow{x}\rangle.\, M.\, [\vec{x}] : \overleftarrow{\tau}\overleftarrow{\rho} \Rightarrow \vec{\sigma}\vec{\tau}$$



The first is *expansion* (see Property 3.9 of the previous paper [6]). The second lifts the arguments for $\overleftarrow{\tau}$ from the stack as the variables $\overleftarrow{x}$, to restore them after evaluating M. We illustrate these above. A premonoidal product further has an *associator* and a *unitor*, and is called *strict* if these are identities, which they are here. The category is then formed by terms modulo $\beta\eta$-equivalence, where η-equivalence is generated by:

$$M : \rho\overleftarrow{\sigma} \Rightarrow \vec{\tau} \;=_\eta\; \langle x\rangle.\, [x].\, M : \rho\overleftarrow{\sigma} \Rightarrow \vec{\tau} \quad \text{where } x \notin \mathsf{fv}(M)$$

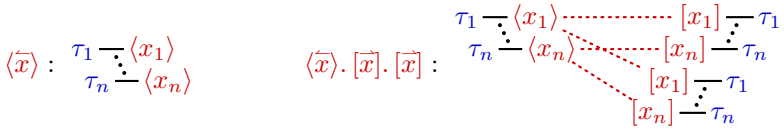
▶ **Proposition 25.** *Terms modulo $\beta\eta$-equivalence form a strict premonoidal category.*

We remark that terms modulo $\beta\eta$-equivalence do *not* form a *symmetric* pre-monoidal category, due to the failure of naturality of symmetry. Of course, one can add further equations to remedy this. In the sequel, we develop an extended equational theory which in fact makes the category of terms Cartesian closed.

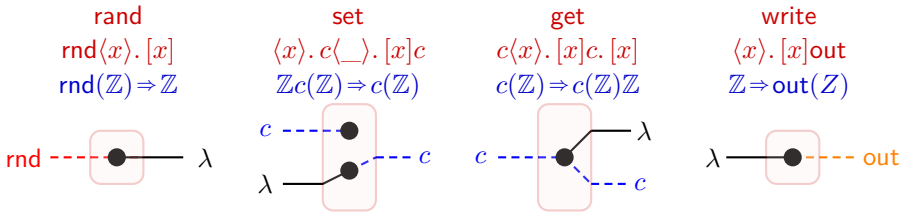
Premonoidal structure forces a notion of sequentiality, which has previously been employed to capture that of effects, as in the closed Freyd categories of Power, Thielecke, and Levy [17, 8], which are the premonoidal equivalent of Cartesian closed categories. However, this imposed sequentiality is only necessary if interactions through effects (such as state) are hidden from the type system. Because the FMC makes these explicit, they can instead be accounted for in the semantics, which then reverts to a Cartesian closed category.

The Cartesian closed category

We now give an example illustrating why we would expect the FMC to form a Cartesian (closed) category, despite its ability to encode effects. For this example, but not for the rest of the section, we will then consider terms *with* locations. Note, first, how the two following terms are illustrated in string diagrams below.

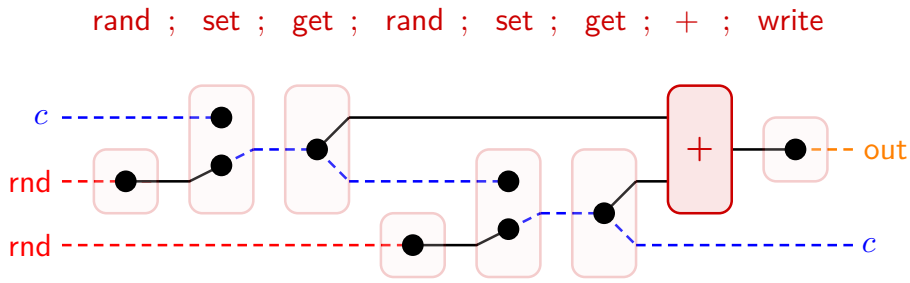


▶ **Example 26.** We introduce the following effectful operations as defined constructs ("sugar") into the FMC: reading from a stream of random integers ($\mathbb{Z}$), rand, a memory cell c with get and set operations, and writing to output, write. We give the definitions and types of these operations and illustrate these as as (nominal) string diagrams below, using colours and dashed lines to indicate non-main locations: red for rnd, blue for c and yellow for out. We use the black dot here to depict a transition from one location to another[1] (as in rnd, set and write), as well as to depict the terminal ! (as in set) and diagonal Δ (as in get), as is standard.



Note that, modulo renaming of wires, these effectful operations are encoded by the diagonal and terminal operations of a Cartesian category.

Consider the following term (reprised from Examples 4.4 and 4.8 of the previous paper [6]) and its string diagrammatic representation. Note that we give the diagram modulo symmetry.[2]



[1] This corresponds to the *renaming* a wire in the formalism of nominal string diagrams, *i.e.*, string diagrams where wires additionally have an associated *name*.

[2] Note that, because operations acting on different locations permute, we have, for example, no need to lift (and restore) the arguments from the stacks at locations λ and c prior to (and subsequent to) applying the second instances of rand, set or get – or, equivalently, doing so has the same result as not doing so. Technically, the term corresponding to the diagram would lift the result of the first instance of get off the main λ stack before applying the second instance of get, and then restore it afterwards; however, since $+$ is symmetric in its inputs, we omit this for simplicity.

Due to the strong type system, we see all the dependencies between operations. For example, the second call to rand may safely be made before the first calls to set and get. This would be illustrated by "sliding" the rand operation along the wire – something which is forbidden in general in a pre-monoidal category, but is permissible in a monoidal setting. We can also see that the second set is dependent on the first get. Indeed, their composition forms a beta-redex, corresponding to the expected interaction of ! with Δ in a Cartesian category.

Note, one can see in the above example that applying the naturality of the diagonal would result in a duplication on the location rnd. This relies on rnd being a location with no special status, and in particular, having an associated *push* action, similar to the main location λ. If we were to enforce that rnd was a read-only stream, then this duplication would no longer be possible and the semantics can no longer be Cartesian. Similar issues arise for memory cells, which ought to have depth (at most) one. We leave consideration of the particular properties of encoded effects for future work.

The following data will make $\otimes$ a Cartesian product $\times$, when considered modulo the following equational theory. The *exponent* $\vec{\sigma}\to\vec{\tau} \triangleq \overleftarrow{\sigma}\Rightarrow\vec{\tau}$ will then give Cartesian closure.

$$\begin{aligned}
! : \vec{\tau} \longrightarrow 1 &= \langle\overleftarrow{x}\rangle : \overleftarrow{\tau} \Rightarrow \\
\delta : \vec{\tau} \longrightarrow \vec{\tau}\times\vec{\tau} &= \langle\overleftarrow{x}\rangle.\,[\vec{x}].\,[\vec{x}] : \overleftarrow{\tau}\Rightarrow\vec{\tau}\vec{\tau} \\
\pi_1 : \vec{\upsilon}\times\vec{\tau} \longrightarrow \vec{\tau} &= \langle\overleftarrow{x}\rangle.\,\langle\overleftarrow{y}\rangle.\,[\vec{x}] : \overleftarrow{\tau}\overleftarrow{\upsilon}\Rightarrow\vec{\tau} \\
\pi_2 : \vec{\upsilon}\times\vec{\tau} \longrightarrow \vec{\upsilon} &= \langle\overleftarrow{x}\rangle.\,\langle\overleftarrow{y}\rangle.\,[\vec{y}] : \overleftarrow{\tau}\overleftarrow{\upsilon}\Rightarrow\vec{\upsilon} \\
\epsilon : \vec{\sigma}\times(\vec{\sigma}\to\vec{\tau}) \longrightarrow \vec{\tau} &= \langle z\rangle.\,z : (\overleftarrow{\sigma}\Rightarrow\vec{\tau})\overleftarrow{\sigma}\Rightarrow\vec{\tau} \\
\eta : \vec{\tau} \longrightarrow (\vec{\sigma}\to\vec{\sigma}\times\vec{\tau}) &= \langle\overleftarrow{x}\rangle.\,[[\vec{x}]] : \overleftarrow{\tau}\Rightarrow(\overleftarrow{\sigma}\Rightarrow\vec{\sigma}\vec{\tau}) \\
M\to N : (\vec{\sigma}\to\vec{\tau}) \longrightarrow (\vec{\rho}\to\vec{\upsilon}) &= \langle z\rangle.\,[M.\,z.\,N] : (\overleftarrow{\sigma}\Rightarrow\vec{\tau})\Rightarrow(\overleftarrow{\rho}\Rightarrow\vec{\upsilon})
\end{aligned}$$

where $\vec{x}\colon\vec{\tau}$, $\vec{y}\colon\vec{\upsilon}$, $z\colon\overleftarrow{\sigma}\Rightarrow\vec{\tau}$

▶ **Definition 27.** *We define the equational theory* $=_{eqn}$ *of the FMC to be the least equivalence generated by the following laws, closed under all contexts.*

Beta:	$[N].\,\langle x\rangle.\,M =_\beta M\{N/x\}$	$\overleftarrow{\sigma}\Rightarrow\vec{\tau}$
Interchange:	$\langle\overleftarrow{x}\rangle.\,N.\,[\vec{x}].\,M =_\iota M.\,\langle\overleftarrow{y}\rangle.\,N.\,[\vec{y}]$	$\overleftarrow{\sigma}\overleftarrow{\rho}\Rightarrow\vec{\upsilon}\vec{\tau}$
Diagonal:	$M\langle\overleftarrow{y}\rangle.\,[\vec{y}].\,[\vec{y}] =_\Delta \langle\overleftarrow{x}\rangle.\,[\vec{x}].\,M.\,[\vec{x}].\,M$	$\overleftarrow{\sigma}\Rightarrow\vec{\tau}\vec{\tau}$
Terminal:	$M.\,\langle\overleftarrow{y}\rangle =_! \langle\overleftarrow{x}\rangle$	$\overleftarrow{\sigma}\Rightarrow$
Eta (First-order):	$\star =_\eta \langle a\rangle.\,[a]$	$\alpha\Rightarrow\alpha$
Eta (Higher-order):	$P =_\epsilon \langle\overleftarrow{x}\rangle.\,[[\vec{x}].\,P.\,\langle z\rangle.\,z]$	$\overleftarrow{\rho}\Rightarrow(\overleftarrow{\sigma}\Rightarrow\vec{\tau})$

where $a\colon\alpha$, $\vec{x}\colon\vec{\sigma},\vec{y}\colon\vec{\tau}, M\colon\overleftarrow{\sigma}\Rightarrow\vec{\tau}$, $N\colon\overleftarrow{\rho}\Rightarrow\vec{\upsilon}$, $z\colon\overleftarrow{\sigma}\Rightarrow\vec{\tau}$ *and* $P\colon\overleftarrow{\rho}\Rightarrow(\overleftarrow{\sigma}\Rightarrow\vec{\tau})$, *and we do not allow abstractions to capture in* M, N, *or* P.

Note that this theory includes beta- and eta-reduction. To see it includes eta-reduction at higher-type, consider the higher-order eta equation with $P = \star$.[3]

▶ **Theorem 28.** *Terms modulo* $=_{eqn}$ *form a strict Cartesian closed category.*

[3] Following the definition of substitution, given a context $\{-\}.\,M$ with hole $\{-\}$, the substitution of a term N into the hole is given by $N\,;M$, in particular with N not binding in M. This means the eta equations together give $\langle x\rangle.\,[x].\,M = M$, where $x \notin \mathsf{fv}(M)$.

The proof of the theorem above provides a canonical functor from the free Cartesian closed category generated over a set of base types Σ, denoted $\mathsf{CCC}(\Sigma)$, to the category of FMC terms generated over th same signature, denoted $\Lambda S/\mathsf{eqn}$. We construct a left-inverse CCC-functor interpreting FMC-terms into λ-terms (with products and patterns), thus proving completeness. In general, all constructions also work with also with constants drawn from a monoidal signature, as well as simply with a signature given by a set of base types.

The interpretation $[\![-]\!] : \Lambda S/\mathsf{eqn} \to \mathsf{CCC}(\Sigma)$ preserves types. The top-level arrow $\Rightarrow$ of FMC-types becomes sequent entailment $\vdash$: the type of the input stack becomes the type of the λ-context and the type of the output stack becomes the type of the λ-term.

A *valuation* v is a function assigning to each FMC-variable $x : \tau$ a λ-term $v(x) \in [\![\tau]\!]$. Given a valuation v, let $v\{x \leftarrow t\}$ denote the valuation which assigns t to x and otherwise behaves as v. We write contexts and products as vectors and elide the isomorphisms for associativity and unitality so that concatenation of s and t may be written as $s \cdot t$.

▶ **Definition 29.** *For each valuation v, define on the type derivation of $\Gamma \vdash M : \overleftarrow{\sigma} \Rightarrow \vec{\tau}$ an open λ-term $[\![\Gamma \vdash M : \overleftarrow{\sigma} \Rightarrow \vec{\tau}]\!]_v$, given by its action on contexts:*

$$\begin{aligned}
[\![\Gamma \vdash \star : \overleftarrow{\sigma} \Rightarrow \vec{\sigma}]\!]_v(s) &= s \\
[\![\Gamma \vdash x : \alpha]\!]_v &= v(x) \\
[\![\Gamma \vdash \langle x \rangle . M : \rho\overleftarrow{\sigma} \Rightarrow \vec{\tau}]\!]_v(s \cdot r) &= [\![\Gamma, x : \rho \vdash M : \overleftarrow{\sigma} \Rightarrow \vec{\tau}]\!]_{v\{x \leftarrow r\}}(s) \\
[\![\Gamma \vdash [N] . M : \overleftarrow{\sigma} \Rightarrow \vec{\tau}]\!]_v(s) &= [\![\Gamma \vdash M : \rho\overleftarrow{\sigma} \Rightarrow \vec{\tau}]\!]_v(s \cdot [\![\Gamma \vdash N : \rho]\!]_v) \\
[\![\Gamma, x : \overleftarrow{\rho} \Rightarrow \vec{\upsilon} \vdash x . M : \overleftarrow{\rho}\overleftarrow{\sigma} \Rightarrow \vec{\tau}]\!]_v(s \cdot r) &= [\![\Gamma, x : \overleftarrow{\rho} \Rightarrow \vec{\upsilon} \vdash M : \vec{\upsilon}\overleftarrow{\sigma} \Rightarrow \vec{\tau}]\!]_v(s \cdot v(x)(r))
\end{aligned}$$

▶ **Theorem 30.** *Terms modulo $=_{\mathsf{eqn}}$ form a complete language for Cartesian closed categories.*

Machine Equivalence

A natural contextual equivalence on terms is given by *machine equivalence*, defined inductively on types below. It resembles the logical relation for program equivalence of Pitts and Stark [12]. We write $(S, M)\Downarrow$ for T if $(S, M)\Downarrow(T, \star)$ and take here the only constants to be of base type.

▶ **Definition 31.** *Closed terms $M : \overleftarrow{\sigma} \Rightarrow \vec{\tau}$ and $M' : \overleftarrow{\sigma} \Rightarrow \vec{\tau}$ are* machine equivalent at type $\overleftarrow{\sigma} \Rightarrow \vec{\tau}$ *if for equivalent inputs the machine gives equivalent outputs,*

$$M \sim M' : \overleftarrow{\sigma} \Rightarrow \vec{\tau} \triangleq \forall S \sim S' : \vec{\sigma}.\ (S, M)\Downarrow \sim (S', M')\Downarrow : \vec{\tau}$$

where two terms of base type are equivalent if they are equal, and two stacks are equivalent if their terms are pairwise equivalent. Equivalence extends to open terms $\Gamma \vdash M : \tau$ and $\Gamma \vdash M' : \tau$ as follows: $\vec{w} : \vec{\omega} \vdash M \sim M' : \tau$ if and only if

$$\forall W \sim W' : \vec{\omega}.\ \{W/\vec{w}\}M \sim \{W'/\vec{w}\}M' : \tau\ .$$

Machine equivalence validates the equational theory (and in particular the beta and eta equations). Thus we have the following result.

▶ **Theorem 32.** *Terms modulo machine equivalence form a Cartesian closed category.*

In fact, the category given by terms modulo machine equivalence is just the extensional collapse of the category of terms modulo the equational theory. Note that machine equivalence is strictly coarser than the equational theory: a situation analogous to that of the simply-typed λ-calculus with products, considered modulo an appropriate contextual equivalence.

7 Further work

The current type system for the FMC is *too strong* for practical programming: it captures such intensional (and unobservable) aspects of computation as the number of elements read from a random stream. We aim to investigate more abstract type systems, including dealing with the particular properties of effectful locations. The results in this paper concerning the type system, which is essentially a presentation of intuitionistic logic, the operational intuition and the close denotational relationship with the λ-calculus make a strong basis for future refinements which account properly for effects. There are several ways in which we already know how to weaken the type system: introducing a recursor, stream types τ^*, which type a stream of terms of type τ, and ignoring types on non-main locations. A close link with string diagrams is evident from the results presented, including with the recently introduced higher-order string diagrams for CCCs [1]. This is another avenue for investigation.

References

1 Mario Alvarez-Picallo, Dan R. Ghica, David Sprunger, and Fabio Zanasi. Functorial string diagrams for reverse-mode automatic differentiation. *Computer Science Logic (CSL) 2023*, 2023. arXiv:2107.13433.

2 Roel de Vrijer. Exactly estimating functionals and strong normalization. *Indagationes Mathematicae (Proceedings)*, 90(4):479–493, 1987.

3 Robin Gandy. Proofs of strong normalization. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 457–477. Academic Press, 1980.

4 Inge Gørtz, Signe Reuss, and Morten Sørensen. Strong normalization from weak normalization by translation into the Lambda-I-calculus. *Higher-Order and Symbolic Computation*, 16:253–285, 2003. doi:10.1023/A:1025693307470.

5 Masahito Hasegawa. Decomposing typed lambda-calculus into a couple of categorical programming languages. In *International Conference on Category Theory and Computer Science*, 1995.

6 Willem Heijltjes. The functional machine calculus. To appear in Mathematical Foundations of Programming Semantics (MFPS 2022). Preprint available at http://people.bath.ac.uk/wbh22/index.html#FMC2022, 2022.

7 Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20:199–207, 2007. doi:10.1007/s10990-007-9018-9.

8 Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call–by–value programming languages. *Information and Computation*, 185:182–210, 2003.

9 Robin Milner. Action calculi, or syntactic action structures. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *Mathematical Foundations of Computer Science 1993, 18th International Symposium, MFCS'93, Gdansk, Poland, August 30 - September 3, 1993, Proceedings*, volume 711 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 1993. doi:10.1007/3-540-57182-5_7.

10 Slava Pestov, Daniel Ehrenberg, and Joe Groff. Factor: A dynamic stack-based programming language. *ACM SIGPLAN Notices*, 45(12):43–58, 2010.

11 Andrew Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.

12 Andrew Pitts and Ian Stark. Operational reasoning for functions with local state. In *Higher order operational techniques in semantics*, pages 227–273. Isaac Newton Institute for Mathematical Sciences, 1998.

13 Gordon Plotkin and John Power. Notions of computation determine monads. In *International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, pages 342–356. Springer, Berlin, Heidelberg, 2002.

14 Jaco van de Pol. Two different strong normalization proofs? In *Selected Papers from the Second International Workshop on Higher Order Algebra, Logic, and Term Rewriting (HOA '95)*, volume 1074 of *LNCS*, pages 201–220, 1995. doi:10.1007/3-540-61254-8_27.

15 Jaco van de Pol and Helmut Schwichtenberg. Strict functionals for termination proofs. In *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications (TLCA '95)*, pages 350–364. Springer-Verlag, 1995.

16 John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7:453–468, 1997.

17 John Power and Hayo Thielecke. Closed Freyd- and κ-categories. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1644 of *LNCS*, pages 625–634. Springer, 1999.

18 Helmut Schwichtenberg. Complexity of normalization in the pure typed lambda-calculus. In Anne Sjerp Troelstra and Dirk van Dalen, editors, *The L.E.J. Brouwer Centenary Symposium*, volume 110 of *Studies in Logic and the Foundations of Mathematics*, pages 453–457. Elsevier, 1982.

19 Thomas Streicher and Bernhard Reus. Classical logic, continuation semantics and abstract machines. *Journal of Functional Programming*, 8(6):543–572, 1998.

20 Glynn Winskel. *The formal semantics of programming languages: An introduction*. MIT Press, Cambridge, Massachusetts, 1993.

Degree Spectra, and Relative Acceptability of Notations

Nikolay Bazhenov
Sobolev Institute of Mathematics, Novosibirsk, Russia

Dariusz Kaloci\'nski
Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract

We investigate the interplay between the degree spectrum of a computable relation R on the computable structure $(\omega, <)$, i.e., natural numbers with the standard order, and the computability of the successor, relativized to that relation, in all computable copies of the structure – the property we dub successor's recoverability. In computable structure theory, this property is used to show that of all possible Turing degrees that could belong to the spectrum of R (namely, of all Δ_2 degrees), in fact only the computably enumerable degrees are contained in the spectrum. Interestingly, successor's recoverability (in the unrelativized form) appears also in philosophy of computing where it is used to distinguish between acceptable and deviant encodings (notations) of natural numbers. Since Shapiro's notations are rarely seen through the lens of computable structure theory, we first lay the elementary conceptual groundwork to understand notations in terms of computable structures and show how results pertinent to the former can fundamentally inform our understanding of the latter. Secondly, we prove a new result which shows that for a large class of computable relations (satisfying a certain effectiveness condition), having all c.e. degrees as a spectrum implies that the successor is recoverable from the relation. The recoverability of successor may be otherwise seen as relativized acceptability of every notation for the structure. We end with remarks about connections of our result to relative computable categoricity and to a similar direction pursued by Matthew Harrison-Trainor in [18], and with an open question.

2012 ACM Subject Classification Theory of computation → Computability

Keywords and phrases Computable structure theory, Degree spectra, ω-type order, C.e. degrees, Δ_2 degrees, Acceptable notation, Successor, Learnability

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.11

Related Version *Previous Version*: `https://arxiv.org/abs/2205.00791`

Funding *Nikolay Bazhenov*: The work of Bazhenov is supported by the Mathematical Center in Akademgorodok under the agreement No. 075-15-2022-281 with the Ministry of Science and Higher Education of the Russian Federation.
Dariusz Kaloci\'nski: The work of Kaloci\'nski is supported by the National Science Centre Poland under the agreement No. 2018/31/B/HS1/04018.

Acknowledgements We would like to thank Matthew Harrison-Trainor and Michał Wrocławski for helpful discussions and anonymous reviewers for their comments.

1 Introduction

Shapiro made a point that computations are performed on syntactic objects, such as strings of symbols, rather than on numbers themselves [34]. Some models of computation are directly based on this premise [38, 25] but some are not [36, 6]. When showing equivalence between these models, one uses some form of notation for natural numbers, e.g., the unary notation. However, as Shapiro observed, not every notation is appropriate for showing the desired

equivalence. Therefore, he asked when a notation is appropriate in the above sense, or, in his words, when it should be deemed acceptable. He showed that a notation is acceptable if and only if the successor function is computable in it.

Shapiro's notations became influential in philosophy of computing (see, e.g., [30, 8, 29, 40, 35]). Meanwhile, we have seen rapid development of a separate, though similar in spirit, research program known as computable structure theory which explores the relationship between computability and algebraic structures [1, 26]. Interestingly, Shapiro's framework can be fully rephrased in terms of computable structures under the following slogan: a notation for a given computable structure corresponds to a computable (isomorphic) copy of the structure. To our best knowledge, this simple fact, and its consequences, has been consistently overlooked (though, see [20]). Bringing to light the connection between Shapiro's notations and computable structure theory is the first, methodological, contribution of the paper, contained in Section 2.

We start Section 2 by providing an appropriate translation between Shapiro's notations and computable structures. Our exposition, although restricted to the structure $(\omega, <)$, i.e., natural numbers with the standard order, generalizes to other computable structures. As a proof of concept, we recast some of the notions, such as notation's acceptability, and results of Shapiro (later, in Section 2.1), using tools developed in computable structure theory. After that, we recall an important notion of degree spectrum. The notion of *degree spectrum* is a classical one [31, 32, 17]. The degree spectrum of a computable relation R on $(\omega, <)$ is the set of all Turing degrees of the images of R in all computable isomorphic copies of $(\omega, <)$. We demonstrate the applicability of this notion to notations by rejecting a certain claim, stemming from a philosophical paper by Benacerraf, according to which, in every notation for $(\omega, <)$, the successor is computable [3]. This claim is equivalent to saying that the degree spectrum of the successor on $(\omega, <)$ is trivial, i.e., consists of only the computable degree. The claim is rejected based on a known result according to which this degree spectrum is actually equal to the set of all (and only) computably enumerable degrees [5]. We then set the ground for Section 3, containing the main technical result. Section 2 could appeal to logically inclined philosophers, interested in connections with the advanced framework of computable structure theory, but also to computable structure theorists who could find an independent source of motivation for their own research (otherwise, they could safely skim through Section 2).

The second and main contribution of the paper is a new result, contained in Section 3. The result explores the logical relationship between the following two statements about a computable relation R: (I) R's degree spectrum on $(\omega, <)$ is equal to the set of all (and only) computably enumerable (c.e.) degrees, and (II) (the image $Succ_{\mathcal{A}}$ of) the successor function $Succ$ is computable relative to (the image $R_{\mathcal{A}}$ of) R, in every computable copy $\mathcal{A} = (\omega, <_{\mathcal{A}})$ of $(\omega, <)$. If R satisfies (II), we say that the successor is recoverable from R on $(\omega, <)$.

The *recoverability of the successor* appears naturally in the investigation of degree spectra on $(\omega, <)$. Every degree spectrum of a computable relation on $(\omega, <)$ is contained in the set of all Δ_2 degrees (recall that a Δ_2 degree is characterized by containing a subset of ω that can be recursively approximated; such sets are also called algorithmically learnable or limiting recursive [28, 15]). It is known that the recoverability of the successor from R on $(\omega, <)$ fixes R's spectrum to that of all c.e. degrees (see, e.g., [18, 2]) which form a proper subset of all Δ_2 degrees [7]. This means that the implication (II) $\Rightarrow$ (I) holds in general. On the other hand, recoverability of the successor plays an important role in isolating the class of acceptable notations [34]: plain acceptability of a computable copy $\mathcal{A}$ of $(\omega, <)$ means that $Succ_{\mathcal{A}}$ is computable. After relativization, we obtain a more general notion: a computable copy $\mathcal{A}$

of $(\omega, <)$ is acceptable relative to R if $Succ_{\mathcal{A}}$ is computable in $R_{\mathcal{A}}$. Now, recoverability of the successor from R on $(\omega, <)$ means that every computable copy $\mathcal{A}$ of $(\omega, <)$ is acceptable relative to R.

The question tackled in Section 3 deals with the implication (I) $\Rightarrow$ (II). For any known example of a computable relation R, the implication holds. We ask whether it holds in general, namely whether for any computable relation R, if R's degree spectrum on $(\omega, <)$ is equal to all c.e. degrees, then the successor is recoverable from R on $(\omega, <)$. Following [2], we attack this problem for a restricted but nevertheless inclusive class of computable relations, namely the graphs of unary total computable functions. We expand on techniques developed in [2], in particular by reusing the concept of block functions. Our result – which answers the aforementioned question in the affirmative – is proved for computable block functions that satisfy certain intuitive effectiveness condition (Theorem 21).

In Section 4 we remark about related work in computable structure theory. In particular, we comment on key differences between our theorem and a theorem by Matthew Harrison-Trainor, where the technique of recovering the successor is used [18]. We also outline a connection between successor's recoverability and the concept of relative computable categoricity.

The last section concludes the paper. We ask whether the effectiveness condition that we use in proving Theorem 21 can be dropped.

2 Notations and Computable Structures

Originally, Shapiro considered notations for natural numbers (henceforth, ω) with no additional structure. A *notation* for ω is any bijection $\sigma\colon S \to \omega$, where S is a fixed countably infinite set of strings over a finite alphabet.[1] The idea is that $\alpha \in S$ is a numeral denoting $\sigma(\alpha)$ and we think of computations as being performed only on numerals. Computability of an n-ary relation R *on natural numbers* in σ is equated, by definition, with the computability of the σ-preimage of R, $\{(\sigma^{-1}(a_1), \sigma^{-1}(a_2), \ldots, \sigma^{-1}(a_n)) : (a_1, a_2, \ldots, a_n) \in R\}$, which is a relation *on numerals*.

▶ **Example 1.** Let T be the set of all nonempty words over the alphabet $\{a\}$ and let $\tau(a^n) = n$, where a^n is the word consisting of n consecutive occurrences of a. Clearly, τ is a notation for ω. The τ-preimage of $<$ is equal to $\{(x, y) \in T^2 : \tau(x) < \tau(y)\}$. This set is computable because comparing word lengths is computable, and for every $x, y \in T$, $\tau(x) < \tau(y) \Leftrightarrow |x| < |y|$. Therefore, the relation $<$ (on natural numbers) is computable in τ.

▶ **Example 2.** Let T be as in Example 1 and let $X \subset \omega$ be non-computable. Consider the sequence $a, aa, aaa, \ldots$ and obtain a new one in the following way: for each $k \geq 0$, swap the positions of a^{2k+1} and a^{2k+2} if $k \in X$. For example, if $0, 2 \in X$ and $1 \notin X$, the new sequence starts with $aa, a, aaa, aaaa, aaaaaa, aaaaa, \ldots$. Let ρ be the inverse of the sequence obtained in this way. ρ is a notation for ω. $<$ is not computable in ρ because if it were computable then X would be, as $k \in X$ iff (a^{2k+2}, a^{2k+1}) belongs to the ρ-preimage of $<$.

Without loss of generality, we may assume that a notation for ω is any bijection from ω to ω. This follows from a simple but important fact that for every infinite computable set S of words over a finite alphabet there exists a computable enumeration of S without

[1] Shapiro's notation should not be confused with Kleene's notation for ordinals. Note, however, that an acceptable notation (to be defined) can indeed be viewed as a notation for the ordinal ω.

repetitions, i.e., a total computable injection $s\colon \omega \to S$ such that $s(\omega) = S$. In other words, s gives us a computable nonrepetitive sequence $s_0, s_1, s_2, \ldots$ which consists of all (and only) elements of S. s may be seen as an encoding of S by ω. We take such identification for granted.

Shapiro considered only notations for plain ω, i.e., for the set of natural numbers without any additional structure (in Section 2.1, we reproduce some of his results in the setting of computable structure theory). However, one can easily extend his notion to cover additional structure. We shall focus on the additional structure in the form of the simplest ordering possible – the standard order on natural numbers, which we denote by $<$. Hence, the structure under investigation is $(\omega, <)$. By a notation for $(\omega, <)$ we shall mean any notation σ for ω in which $<$ is computable.

Now, let us turn our attention to computable structures. A structure $(A, R_1, \ldots, R_n)$ is said to be *computable* if its universe A and each relation R_i is computable. Without loss of generality, we can assume that $A = \omega$, the reason being similar to the one already discussed for S. Clearly, $(\omega, <)$ is a computable structure.

To pinpoint the connection between notations and computable structures, consider the following definition.

▶ **Definition 3.** *$(\omega, \prec)$ is a computable copy of $(\omega, <)$ if $\prec$ is a computable ordering on ω and structures $(\omega, <)$ and $(\omega, \prec)$ are isomorphic.*

Now, we can make the following observations. Let $\sigma\colon \omega \to \omega$ be a notation for $(\omega, <)$ and let $\prec$ be the σ-preimage of $<$, i.e., $\prec := \{(\sigma^{-1}(x), \sigma^{-1}(y)) : x < y\}$. By the definition of notation, $\prec$ is computable. Moreover, by the definition of $\prec$, the structures $(\omega, <)$ and $(\omega, \prec)$ are isomorphic. Therefore, by Definition 3, $(\omega, \prec)$ is a computable copy of $(\omega, <)$.

The next observation goes in the other direction. Let $(\omega, \prec)$ be a computable copy of $(\omega, <)$. Let $h\colon (\omega, <) \cong (\omega, \prec)$ be an isomorphism between the two structures and let $\sigma = h^{-1}$. Obviously, σ is a notation for ω. Also, the σ-preimage of $<$ is equal to $\prec$ and, by Definition 3, $\prec$ is computable. Therefore, σ is a notation for $(\omega, <)$.

Based on the above two observations, we can posit that a notation for $(\omega, <)$ is any isomorphism that maps a computable structure $(\omega, \prec)$ to $(\omega, <)$. Therefore, instead of notations for $(\omega, <)$ we may equivalently speak about computable copies of $(\omega, <)$. This generalizes straightforwardly to arbitrary computable structures.

Let us recall one of the concepts introduced by Shapiro, namely acceptability of notation, and see what does it mean in terms of computable structures. A notation for ω is said to be *acceptable* if the successor function (henceforth, *Succ*) is computable in it (this implies that all recursive functions are); otherwise the notation is called deviant. Acceptability of notation for plain ω can be extended, in an obvious way, to notations for ω with additional structure, in particular to the structure $(\omega, <)$.

What does the desideratum of notation's acceptability mean from the perspective of computable structures? Before we answer this, consider the following convention which is commonplace when referring to isomorphic copies of a given structure.

▶ **Definition 4.** *Let R be a relation on $(\omega, <)$, i.e., $R \subseteq \omega^k$, for some $k \in \omega$, and let $\mathcal{A}$ be a computable copy of $(\omega, <)$. If φ is an isomorphism from $(\omega, <)$ to $\mathcal{A}$, we write $R_{\mathcal{A}}$ for the image of R under φ.*

Now, in terms of computable structures, the desideratum of acceptability of a computable copy $\mathcal{A} = (\omega, <_{\mathcal{A}})$ – which uniquely identifies a notation for $(\omega, <)$ – says that $Succ_{\mathcal{A}}$ should be computable.

Shapiro showed, among others, that not every notation for ω is acceptable. The following question arises immediately: is $Succ_{\mathcal{A}}$ computable, if $\mathcal{A}$ is any computable copy $\mathcal{A}$ of $(\omega, <)$? In other words, is every computable copy of $(\omega, <)$ acceptable? In an influential philosophical paper, Benacerraf hinted in passing at the affirmative (cf. p. 276 in [3] and, also, [4]). Essentially, his claim was that, in any given notation, the computability of the successor is equivalent to the computability of the ordering (we take the liberty to identify Benacerraf's intuitive concept of notation with the formal one).

As we will see, Benacerraf's claim is false in view of Proposition 6 below. The proposition uses an important notion of degree spectrum, originating from Richter [31, 32] and Harizanov [17] (see, also, [19, 14]). The notion of degree spectrum – here defined for $(\omega, <)$ but, in general, applicable to any computable structure – is based on the concept of Turing degrees. As a brief reminder, Turing degrees are equivalence classes of the relation $\equiv_T$ on subsets of ω, defined by $A \equiv_T B \Leftrightarrow (A \leq_T B \wedge B \leq_T A)$, where $X \leq_T Y$ means that the characteristic function of X can be computed by a program which can ask questions of the form "$n \in Y$?" for different n's and use the answers to make decisions while the program is running. For more information, see, e.g., [9, 33, 37].

▶ **Definition 5** (degree spectrum of a relation). *The degree spectrum of a computable relation R on $(\omega, <)$, in symbols $DgSp_{(\omega,<)}(R)$, is the set of Turing degrees of $R_{\mathcal{A}}$ over all computable copies $\mathcal{A}$ of $(\omega, <)$.*

We write $DgSp(R)$ as we do not consider degree spectra on structures other than $(\omega, <)$.

As we can see, the notion of degree spectrum encompasses all possible complexities of a relation (formalized as Turing degrees) across all notations for the underlying structure.

Before formulating Proposition 6, let us remind that a Turing degree is computably enumerable (c.e.) if it contains a computably enumerable set, i.e., a set which, if nonempty, can be enumerated by an algorithm. Since there exist noncomputable c.e. sets (e.g., the halting problem), there are c.e. degrees which are different than $\mathbf{0}$ (the degree of computable sets).

▶ **Proposition 6** (see, e.g., Example 1.3 in [5]). *The degree spectrum of the successor on $(\omega, <)$ consist of all (and only) c.e. Turing degrees.*

Clearly, Benacerraf's claim must be false, for otherwise every computable copy of $(\omega, <)$ would be acceptable (i.e., the successor would be computable in it) and we would have $DgSp(Succ) = \{\mathbf{0}\}$ which contradicts Proposition 6.

The degree spectrum of the successor on $(\omega, <)$, i.e., the set consisting of all c.e. degrees, is just one example of degree spectrum on $(\omega, <)$, but other examples exist and it is still not known what are all possibilities. One kind of degree spectrum is the trivial one, i.e., $\{\mathbf{0}\}$. Relations having this spectrum are called *intrinsically computable* and were characterized by Moses [27]. There are also other kinds of spectra, including the set of all Δ_2 degrees [10, 39, 18] and other spectra, discovered quite recently [2].

Anyway, the degree spectrum of the successor is, in a sense, special. Essentially, $DgSp(Succ) \subseteq DgSp(R)$ for every computable R which is not intrinsically computable (see, Theorem 4.7 in [39]). To show that $DgSp(R) = DgSp(Succ)$, one typically uses a technique known as *recovering the successor* [2]. The idea is to prove that $R_{\mathcal{A}} \geq_T Succ_{\mathcal{A}}$ holds for every computable copy $\mathcal{A}$ of $(\omega, <)$; this is sufficient because $Succ_{\mathcal{A}} \geq_T R_{\mathcal{A}}$ always,

i.e., for every computable copy $\mathcal{A}$ of $(\omega, <)$.[2] In a sense, such a proof shows how the successor (inside $\mathcal{A}$) can be recovered from the relation R (again, inside $\mathcal{A}$). If this can be done, we say that the successor is recoverable from R on $(\omega, <)$.

In the second part of the paper, we are interested whether one can go in the other direction. Specifically, we ask whether $DgSp(R) = DgSp(Succ)$ implies that the successor is recoverable from R on $(\omega, <)$. In other words, given the following properties of a computable relation R:

$$\text{the degree spectrum of } R \text{ on } (\omega, <) \text{ is equal to all c.e. degrees, and} \tag{I}$$

$$\text{for every computable copy } \mathcal{A} \text{ of } (\omega, <),\ R_{\mathcal{A}} \geq_T Succ_{\mathcal{A}}, \tag{II}$$

we ask whether (I) implies (II) (the other direction follows from one of the paragraphs above). Note that (II) can be seen as a relativized variant of acceptability of notation. We may posit that a computable copy (notation) $\mathcal{A} = (\omega, <_{\mathcal{A}})$ is acceptable relative to R if $R_{\mathcal{A}}$ computes $Succ_{\mathcal{A}}$. Thus, the property in question encompasses computable relations such that every computable copy $(\omega, <_{\mathcal{A}})$ is acceptable relative to them.

Although we are not able to prove this implication in full generality, we show that it holds for a wide subclass of all total computable functions (seen as computable binary relations). For certain types of functions, the implication already follows from earlier results which will be discussed at the beginning of Section 3. Our extension concerns the class of block functions, isolated in this context by Bazhenov et al. [2], and satisfying an additional effectiveness condition.

2.1 Shapiro's Theorems Re-proved

In this subsection, some of the results on Shapiro's notations, here Theorems 11 and 12, are re-proved using tools from computable structure theory. This provides concrete evidence of the intrinsic connection between the two frameworks. The proofs are based on early results from computable structure theory, some of which even predate Shapiro's paper.

For Theorem 11, we need a classical notion of the degree spectrum of a structure. It is different from the notion of the degree spectrum of a relation (Definition 5) in that it focuses on the complexity of the copies of the structure itself. Let L be a computable language, i.e., L consists of a computable set of constants, function and relation symbols, and the map assigning arity to each symbol is also computable.

▶ **Definition 7** (degree spectrum of a structure). *For a countably infinite L-structure $\mathcal{A}$, its* degree spectrum *is the set of all Turing degrees* $\mathbf{d}$ *such that there is an L-structure $\mathcal{B}$ with the following properties:*

- **(i)** *$\mathcal{B}$ is isomorphic to $\mathcal{A}$,*
- **(ii)** *the domain of $\mathcal{B}$ equals ω,*
- **(iii)** *the Turing degree of $D(\mathcal{B})$ – the atomic diagram of $\mathcal{B}$ – is equal to* $\mathbf{d}$ *(here a formula ϕ from the atomic diagram is identified with its Gödel number $\ulcorner \phi \urcorner$).*

▶ **Definition 8.** *A structure $\mathcal{A}$ is called* automorphically trivial *if there is a finite set $X \subseteq \mathrm{dom}(\mathcal{A})$ such that every permutation of $\mathrm{dom}(\mathcal{A})$ that fixes X is an automorphism of $\mathcal{A}$.*

[2] To compute in $Succ_{\mathcal{A}}$ whether $(a_1, \dots a_n) \in R_{\mathcal{A}}$, use $Succ_{\mathcal{A}}$ and one parameter, e.g. the $<_{\mathcal{A}}$-minimal element, to identify the positions $i_1, \dots, i_n$ of $a_1, \dots, a_n$ in the ordering $<_{\mathcal{A}}$, and return $R(i_1, \dots, i_n)$.

▶ **Example 9.** Let E be an equivalence relation on ω with finitely many finite equivalence classes and just one infinite equivalence class. The structure (ω, E) is automorphically trivial, because every permutation of ω that fixes $X = \{x \in \omega : \text{the } E\text{-equivalence class of } x \text{ is finite}\}$ is an automorphism of (ω, E). Moreover, the degree spectrum of (ω, E) is $\{\mathbf{0}\}$. For let $\mathcal{B} = (\omega, E_{\mathcal{B}})$ be isomorphic to (ω, E) via h. $\ulcorner a = b \urcorner \in D(\mathcal{B})$ iff $a = b$. $\ulcorner E(a, b) \urcorner \in D(\mathcal{B})$ iff $a, b \notin h^{-1}(X)$ or $a, b \in h^{-1}(X) \wedge E(h(a), h(b))$. The latter is computable by the finiteness of $h^{-1}(X)$. Hence, $D(\mathcal{B})$ is computable, given that the underlying Gödel numbering of formulas is effective.

▶ **Theorem 10** (Theorem 4 of [21]). *If a countably infinite structure $\mathcal{A}$ is not automorphically trivial, then its degree spectrum is closed upwards in Turing degrees.*

▶ **Theorem 11** (T1 and C1 in [34]). *A function $f : \omega \to \omega$ is computable in every notation for ω if and only if either f is almost constant (i.e., there is c such that $f(x) = c$ holds for all but finitely many x), or f is almost identity (i.e., $f(x) = x$ for all but finitely many x). A relation $R \subseteq \omega$ is computable in every notation for ω if and only if either R is finite, or R is cofinite.*

Proof. We only consider the nontrivial direction ($\Rightarrow$). Suppose that f is computable in every notation for ω. Fix some standard notation σ_0 – e.g., the one induced by decimal numerals. Our function f must be computable in σ_0. Consider the language $L = \{F\}$, where F is a unary function symbol. We define an L-structure $\mathcal{S}_f$ as follows: the domain of $\mathcal{S}_f$ equals ω, and the function symbol F is interpreted as our function f. Since $f(x)$ is computable in σ_0, it is easy to show that the structure $\mathcal{S}_f$ is computable.

Now, if $f(x)$ is neither almost constant nor almost identity, then one can show that the corresponding structure $\mathcal{S}_f$ is not automorphically trivial. Thus, by Theorem 10, one can obtain an isomorphic copy $\mathcal{A}$ of $\mathcal{S}_f$ such that the atomic diagram $D(\mathcal{A})$ of the structure $\mathcal{A}$ is Turing equivalent to, say, the halting problem. Now we define a notation $\sigma : S \to \omega$:

(i) S equals (the decimal representation of) ω, and
(ii) σ is an arbitrary isomorphism from $\mathcal{A}$ onto $\mathcal{S}_f$.

Since $D(\mathcal{A})$ is not computable, one can show that the function $f(x)$ is not computable in σ. ◀

Finally, let us consider the following result by Shapiro, in which notation's acceptability (a) gets an intuitive equivalent (b).

▶ **Theorem 12** ([34]). *For any notation σ for ω, the following are equivalent:*

(a) *For any function f, f is computable in σ if and only if f is computable in the standard notation (e.g., the stroke notation).*
(b) *The successor function is computable in σ.*

To establish the above result, we need the notion of computable categoricity which goes back to the papers by Mal'tsev [23, 24].

▶ **Definition 13.** *A computable L-structure $\mathcal{A}$ is* computably categorical *if for every computable L-structure $\mathcal{B}$, which is isomorphic to $\mathcal{A}$, there is a computable isomorphism g from $\mathcal{B}$ onto $\mathcal{A}$ (i.e., g is an isomorphism which is also a computable function).*

No matter which computable copy of a computably categorical structure we take, it will have the same computability-theoretic properties. A simple example of a computably categorical structure will be considered in the proof of Theorem 12.

▶ **Example 14.** $(\omega, <)$ is not computably categorical. By Proposition 6, there exists a computable structure $(\omega, <_{\mathcal{A}})$ isomorphic to $(\omega, <)$ such that $Succ_{\mathcal{A}}$ is noncomputable. If $(\omega, <)$ were computably categorical, we would have a computable isomorphism g from $(\omega, <_{\mathcal{A}})$ to $(\omega, <)$. But then $Succ_{\mathcal{A}}$ would be computable because $Succ_{\mathcal{A}} = g^{-1} \circ Succ \circ g$.

Recall that a structure $\mathcal{A}$ is finitely generated if there exists a finite set $G \subset dom(\mathcal{A})$ such that the set generated by G in $\mathcal{A}$ – i.e., the least set including G together with all the distinguished elements of $\mathcal{A}$, and closed under the operations of $\mathcal{A}$ – is equal to $dom(\mathcal{A})$.

▶ **Theorem 15** (Theorem 4.1.2 in [23])**.** *Every finitely generated, computable algebraic structure is computably categorical.*

Proof of Theorem 12. We sketch the proof for the direction (b)⇒(a). Since the computable structure $(\omega, Succ)$ is one-generated, $(\omega, Succ)$ is computably categorical by Theorem 15. Now consider a notation $\sigma\colon S \to \omega$ such that the successor function is computable in σ. Let $Succ^{\sigma}$ be the image of the successor under σ^{-1}. We define a computable L-structure $\mathcal{T}_{S,\sigma}$ as follows.

- The domain of $\mathcal{T}_{S,\sigma}$ equals ω.
- Fix a computable bijection ψ from ω onto S. The unary function symbol F is interpreted as follows: for $k \in \omega$,

$$F(k) = \psi^{-1}(Succ^{\sigma}(\psi(k))).$$

It is not hard to show that the structure $\mathcal{T}_{S,\sigma}$ is a computable isomorphic copy of $(\omega, Succ)$. Then the computable categoricity of $(\omega, Succ)$ allows to choose (the unique) computable isomorphism g from $\mathcal{T}_{S,\sigma}$ onto $(\omega, Succ)$.

After that, one can use the computability of g to easily show that the notation σ satisfies the conditions from the item (a). ◀

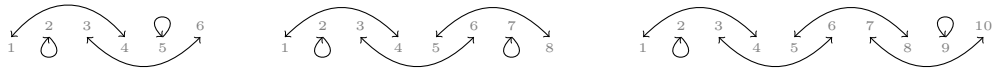
3 Spectrum of C.E. Degrees and Recovering the Successor

Following Bazhenov et al. [2], we restrict our attention to computable binary relations R of general interest – graphs of unary total computable functions. The results of Moses [27] imply that the degree spectrum of such an f is trivial if and only if f is almost constant or almost identity (see, also, the associated version of [2]). Clearly, the equivalence (I) ⇔ (II) holds for such functions as they do not satisfy neither (I) nor (II). Bazhenov et al. [2] isolated the class of quasi-block functions (which include almost constant and almost identity functions) and showed that all computable functions f outside this class satisfy (I) and (II) (Theorem 18 in [2]). We recall that f is a quasi-block function if there are arbitrarily long initial segments of $(\omega, <)$ which are closed under f. Further information about quasi-block functions, pertinent also to the problem at hand, can be found in [2].

Among quasi-block functions there is a narrower class of block functions which we define below. Given a linear ordering $\mathcal{L} = (L, \prec)$, by an interval in $\mathcal{L}$ we mean any set $I \subseteq L$ such that for every $x, y, z \in L$, if $x, z \in I$ and $x \prec y \prec z \in I$, then $y \in I$. For $a, b \in L$, $[a; b] = \{x \in L : a \preceq x \preceq b\}$, where $\preceq$ is the non-strict ordering corresponding to $\prec$.

▶ **Definition 16.** *Let $f\colon \omega \to \omega$ be a total function. An interval I in $(\omega, <)$ is* f-closed *if for all $x \in I$, $f(x) \in I$ and $f^{-1}(\{x\}) \subseteq I$. For a finite non-empty interval $I \subset \omega$, the structure $(I, <, f \upharpoonright I)$ is an* f-block *if it has the following properties:*

(i) *I is an f-closed interval and it cannot be written as a disjoint union of at least two f-closed intervals;*



$\mathcal{J}_0$. $\mathcal{J}_1$. $\mathcal{J}_2$.

Figure 1 Structures $\mathcal{J}_n = ([1; 6+2n], <, f)$, for $n = 0, 1, 2$, where f is the involution such that $f(k) = k$ iff $k = 2$ or $k = 6 + 2n - 1$, and $f(k) = k + 3$ for odd numbers $\leq 6 + 2n - 3$.

(ii) *$\{x \in \omega : x < \min(I)\}$ is f-closed.*
The function f is a block function *if for every $a \in \omega$, there is an f-block containing a. If $(I, <, f \restriction I)$ is an f-block, we refer to its isomorphism type as an* f-type *(or a type).*

A block function on $(\omega, <)$ can be visualized as a sequence of blocks, arranged one after another according to $<$. We develop this intution in the examples below.

▶ **Example 17** (borrowed from [2])**.** Consider finite structures $\mathcal{J}_n$ from Figure 1. Let g be the involution such that $(\omega, <, g) \cong \mathcal{J}_0 + \mathcal{J}_1 + \mathcal{J}_2 + \ldots$ Clearly, g is a block function. Condition (ii) in Definition 16 ensures that every element is contained in a unique g-block. Without (ii), each loop, i.e., a substructure of the form $(\{x\}, < \restriction \{x\}, g \restriction \{x\})$ with $g(x) = x$, would be a g-block itself.

▶ **Example 18.** Let f be a function such that each of its blocks is isomorphic to a structure $\mathcal{C}_k = ([0; k], <, f_k)$, for some $k \in \omega$, where $<$ orders $[0; k]$ in the standard way, $f_k(x) = x + 1$, for $0 \leq x < k$, and $f_k(k) = 0$. It is natural to view $\mathcal{C}_k$ as a *cycle*. This illustrative example is worth remembering because we will use it in the proof of the main result, Theorem 21.

In this paper, we consider functions f that satisfy the effectiveness condition $(\star)$ defined below. The problem whether $(\star)$ can be dropped without affecting Theorem 21 is left open (see, Section 5).

For a set X, by $\mathrm{card}(X)$ we denote the cardinality of X.

▶ **Definition 19.** *Let $f \colon \omega \to \omega$. We define $cp_f \colon \omega \to \omega \cup \{\infty\}$ as follows:*

$cp_f(x) = \mathrm{card}(f^{-1}(\{x\}))$, *for all $x \in \omega$.*

Observe that for the involution g from Example 17 we have $cp_g = 1$. The same holds for functions considered in Example 18.

The effectiveness condition is the following statement about f:

cp_f is computable. $(\star)$

Clearly, functions from Examples 17 and 18 satisfy $(\star)$. More generally, any injective computable block function satisfies it because such a function must be composed of finite cycles and thus the preimage of each single element has cardinality 1. Another less trivial example could be a computable block function having only finitely many pairwise nonembeddable types. In general, however, the behavior of cardinalities of the preimages can be quite complicated. For example, it is not hard to produce an example of a computable block function f such that cp_f is noncomputable (see, also, Theorem 27).

Let f be a block function and let $(\omega, <, f) = \mathcal{F}_1 + \mathcal{F}_2 + \ldots$, where each $\mathcal{F}_n$ is an f-block. By an initial segment of $(\omega, <, f)$ we mean any structure $\Sigma_{i=1}^{k} \mathcal{F}_k$, for $k \in \omega$. We say that $(\mathcal{B}_j)_{j \in \omega}$ is a sequence of growing initial segments of $(\omega, \prec, f)$ if each $\mathcal{B}_j$ is an initial segment of $(\omega, \prec, f)$ and for all j we have $dom(\mathcal{B}_j) \subset dom(\mathcal{B}_{j+1})$. Given structures $\mathcal{A}$ and $\mathcal{B}$ over the

same relational signature (treat each function as a relation), we say that $\mathcal{A}$ embeds precisely once in $\mathcal{B}$ if there is exactly one embedding from $\mathcal{A}$ to $\mathcal{B}$. Recall that $h\colon dom(\mathcal{A}) \to dom(\mathcal{B})$ is an embedding from $\mathcal{A}$ to $\mathcal{B}$ if h is an injection and for every relational symbol R from the signature, for every tuple $\overline{a} \in dom(\mathcal{A})$ of length equal to the arity of R, $\mathcal{A} \models R(\overline{a})$ if and only if $\mathcal{B} \models R(h(\overline{a}))$.

▶ **Lemma 20.** *Let f be a unary total computable block function. If there exists an infinite computable sequence $(\mathcal{B}_j)_{j\in\omega}$ of growing initial segments of $(\omega, <, f)$ such that each $\mathcal{B}_j$ embeds in $(\omega, <, f)$ precisely once, then the successor is recoverable from f on $(\omega, <)$.*

Proof. To compute $Succ_{\mathcal{A}}(x)$ for a given computable copy $\mathcal{A} = (\omega, <_{\mathcal{A}})$ of $(\omega, <)$, look for bigger and bigger substructures $\mathcal{A}_j$ of $(\omega, <_{\mathcal{A}}, f_{\mathcal{A}})$ that look precisely as $\mathcal{B}_j$. We do this by enumerating ω one by one, arranging enumerated elements according to $<_{\mathcal{A}}$ and asking $f_{\mathcal{A}}$ for the values of the enumerated elements. Once $\mathcal{A}_j$ isomorphic to $\mathcal{B}_j$ is discovered, we are sure that $\mathcal{A}_j$ is an initial segment of $(\omega, <_{\mathcal{A}}, f_{\mathcal{A}})$. We continue in this manner and wait for x and at least one $y >_{\mathcal{A}} x$ to enter some $\mathcal{A}_j$. At this point we know the position of x in $<_{\mathcal{A}}$ and $Succ_{\mathcal{A}}(x)$ can be read out from $\mathcal{A}_j$. ◀

We prove (I)⇒(II) by contrapositive. Notice that we can additionally assume that we work with computable block functions which are not almost identities because, as discussed earlier, such functions have the trivial degree spectrum.

▶ **Theorem 21.** *Let f be a computable block function such that it is not almost identity and it satisfies the effectiveness condition $(\star)$. Assume that the successor is not recoverable from f on $(\omega, <)$. Then the degree spectrum of f on $(\omega, <)$ contains a non-c.e. degree.*

▶ **Corollary 22.** *Suppose f is a computable block function satisfying the effectiveness condition $(\star)$. Then* (I) *implies* (II)*, i.e., if the degree spectrum of f on $(\omega, <)$ is equal to all c.e. degrees, then the successor is recoverable from f on $(\omega, <)$.*

Before we start the proof, let us remind a few conventions regarding Turing functionals. A Turing functional, also called a Turing operator or a computable operator, is a program that is allowed to query an oracle. We can enumerate them in an effective way $\Theta_0, \Theta_1, \ldots$. We write $\Theta_e^X(n)$ for the output of the eth Turing functional on input n when it uses $X \subseteq \omega$ as oracle. Θ_e corresponds to a fixed program that can be used with different oracles. We assume in s steps it is possible to read at most the first s entries of the oracle. For a finite string $\sigma \in 2^{<\omega}$, $\Theta_e^\sigma(n)$ means the same as $\Theta_{e,|\sigma|}^\sigma(n)$. Sometimes we explicitly write the number of steps t after the code of the program, $\Theta_{e,t}^\sigma$.

Proof of Theorem 21. Fix an effective enumeration $(\Phi_i, \Psi_i, W_i)_{i\in\omega}$ containing all triples such that Φ_i and Ψ_i are Turing functionals, and W_i is a c.e. set. For brevity, the graph of the function $f_{\mathcal{A}}$ (i.e., the isomorphic image of our function f inside $\mathcal{A}$) is denoted by Γ_A. We build a computable isomorphic copy $\mathcal{A} = (\omega, <_A)$ of the ordering $(\omega, <)$. Along the construction, we satisfy the following requirements:

> If $W_i = \Phi_i^{\Gamma_A}$ and $\Gamma_A = \Psi_i^{W_i}$, then there is a computable sequence $(\mathcal{B}_j)_{j\in\omega}$ of growing initial segments of $(\omega, <, f)$ such that each $\mathcal{B}_j$ embeds in $(\omega, <, f)$ precisely once. $(\mathcal{P}_i)$

Satisfying each $\mathcal{P}_i$ is sufficient by Lemma 20. Indeed, since the successor is not recoverable from f, Lemma 20 guarantees that the degree Γ_A is not c.e. Our strategy for satisfying $\mathcal{P}_i$ incorporates the classical construction of a properly d.c.e. Turing degree by Cooper [7] adapted to the setting of block functions [2]. Recall that a Turing degree is properly d.c.e. if

it contains a subset X of natural numbers such that X is equal to the difference of two c.e. sets and X is not Turing equivalent to any c.e. set. The eventual success of our strategy is secured by a series of threats which attempt to build $(\mathcal{B}_m)$. This will be a degenerate infinite injury construction, described using the framework of trees of strategies (see, e.g., [22]). Degeneracy here means that infinitary outcomes (to be defined) never turn up as the true outcomes of the construction.

For the sake of simplicity, first we give a construction for the case when each block is isomorphic to a cycle, as in Example 18. The modifications needed for the general case of the theorem will be discussed at the end of the proof.

Strategy for $\mathcal{P}_i$. The strategy attempts to build a computable sequence of initial segments $(\mathcal{B}_m)_{m\in\omega}$. Suppose that our strategy starts working at a stage s_0. Then the strategy proceeds as follows.

(1) Set $m = 0$.

(2) Choose two adjacent blocks $\tilde{\mathcal{C}}_m < \tilde{\mathcal{D}}_m$ in $(\omega, <, f)$ such that we have not copied them into the structure $\mathcal{A}_{s_m}$ yet (where s_m is the current stage), and $\tilde{\mathcal{D}}_m$ contains at least two elements. Such a choice is possible, since the function f is not almost identity (hence, it has infinitely many cycles of size ≥ 2).
We extend $\mathcal{A}_{s_m}$ to a finite structure $\mathcal{A}^1_m$ by copying all missing elements up to the end of the block $\tilde{\mathcal{D}}_m$ (these elements are appended to the end of $\mathcal{A}_{s_m}$). Then the structure $\mathcal{A}^1_m$ can be decomposed as follows:

$$\mathcal{A}^1_m = \mathcal{A}^{1,init}_m + \mathcal{C}_m + \mathcal{D}_m,$$

where $\mathcal{C}_m \cong \tilde{\mathcal{C}}_m$ and $\mathcal{D}_m \cong \tilde{\mathcal{D}}_m$.
Let x_m be the rightmost element of $\mathcal{C}_m$, and let y_m be the leftmost element of $\mathcal{D}_m$. It is clear that at the moment, $\langle x_m, y_m\rangle \notin \Gamma_A$. As usual, we restrict "$\langle x_m, y_m\rangle \notin \Gamma_A$" by forbidding lower priority actions to add new elements between the elements of $\mathcal{A}^1_m$.

(3) Wait for a stage $s' > s_m$ witnessing the following computations: for some $t' \leq s'$, we have (at the stage s')

$$W_i \upharpoonright t' = \Phi_i^{\Gamma_A} \upharpoonright t', \text{ and } 0 = \Gamma_A(\langle x_m, y_m\rangle) = \Psi_i^{W_i\upharpoonright t'}(\langle x_m, y_m\rangle).$$

Note that the current structure $\mathcal{A}_{s'}$ can be decomposed as follows:

$$\mathcal{A}_{s'} = \mathcal{A}^1_m + \mathcal{A}^{2,fin}_m = \mathcal{A}^{1,init}_m + \mathcal{C}_m + \mathcal{D}_m + \mathcal{A}^{2,fin}_m,$$

where $\mathcal{A}^{2,fin}_m$ contains the elements added after the end of Step (2).

(4) We define $\mathcal{B}_m := \mathcal{A}_{s'}$. We extend $\mathcal{A}_{s'}$ by adding *precisely one* fresh element between the rightmost element of $\mathcal{A}^{1,init}_m$ and the leftmost element of $\mathcal{C}_m$.
Inside the resulting structure $\mathcal{A}^3_m$, the number x_m becomes the leftmost element of a copy of the cycle $\mathcal{D}_m$. This implies that at the moment, we have $\langle x_m, y_m\rangle \in \Gamma_A$. Similarly to Step (2), we restrict "$\langle x_m, y_m\rangle \in \Gamma_A$".

(5) We wait for a stage $s'' > s'$ witnessing the following condition: for some $t'' \leq s''$, we have $t' \leq t''$ and

$$W_i \upharpoonright t'' = \Phi_i^{\Gamma_A} \upharpoonright t'', \text{ and } 1 = \Gamma_A(\langle x_m, y_m\rangle) = \Psi_i^{W_i\upharpoonright t''}(\langle x_m, y_m\rangle).$$

When the stage s'' is found, we go back to Step (2) with $m + 1$ (in place of m), while *simultaneously waiting* at Step (6) with m.

(6) We wait for a stage $s''' > s''$ witnessing the following condition: one can embed the elements of $\mathcal{A}_{s'''}$ into $(\omega, <, f) \upharpoonright (2 \cdot \mathrm{card}(\mathrm{dom}(\mathcal{A}_{s'''})) + 1)$ in the following special "semi-isomorphic" way. There exists a 1-1 function $\xi \colon \mathrm{dom}(\mathcal{A}_{s'''}) \to [0, 2 \cdot \mathrm{card}(\mathrm{dom}(\mathcal{A}_{s'''}))]$ such that:

- ξ respects the ordering, i.e., for *all* elements $x, y \in \mathcal{A}_{s'''}$, $x <_{\mathcal{A}_{s'''}} y \Leftrightarrow \xi(x) < \xi(y)$;
- if x (originally) was an element of some block I inside $\mathcal{A}_{s'} = \mathcal{B}_m$, then the whole block I should go into some copy of I inside $(\omega, <, f)$; in other words, the restriction of ξ to the domain of $\mathcal{A}_{s'}$ is an isomorphic embedding from $(\mathrm{dom}(\mathcal{A}_{s'}), <_{\mathcal{A}_{s'}}, f_{\mathcal{A}_{s'}})$ into $(\omega, <, f)$.

Note the following: if such a "semi-isomorphic" embedding ξ exists, then one may assume that this ξ satisfies an additional condition:

> If I is a block inside $\mathcal{A}_m^{1,init}$, then I is mapped to its counterpart inside $(\omega, <, f)$ $\qquad (**)$
> (i.e., if $x \in I$ and x is the i-th element from the left inside $\mathcal{A}_{s'''}$, then $\xi(x)$ equals i).

Indeed, since the structure $\mathcal{A}_m^3$ from Step (4) is obtained by adding only one element just before $\mathcal{C}_m$, the embedding ξ must move the contents of $\mathcal{C}_m$ to the right of the $(\omega, <, f)$-counterpart of $\mathcal{A}_m^{1,init}$. This allows us not to move $\mathcal{A}_m^{1,init}$, and just map it to its $(\omega, <, f)$-counterpart.

(7) Extend $\mathcal{A}_{s'''}$ to a finite structure by using the "semi-isomorphic" embedding ξ described above. More formally, we take the least unused numbers $y \notin \mathrm{dom}(\mathcal{A}_{s'''})$ to extend $\mathcal{A}_{s'''}$ to a finite structure $\mathcal{A}_m^4$ such that there exist a number N and an isomorphism $h \colon \mathcal{A}_m^4 \cong (\omega, <, f) \upharpoonright N$ with the property $\xi \subseteq h$. Stop the strategy, including the actions for all $m' \neq m$.

Outcomes of $\mathcal{P}_i$.

w_m : Waiting at Step (3) forever for this m. Then either $W_i \neq \Phi_i^{\Gamma_A}$ or $\Gamma_A \neq \Psi_i^{W_i}$.

w'_m : Waiting at Step (5) forever for this m. Then, again, $W_i \neq \Phi_i^{\Gamma_A}$ or $\Gamma_A \neq \Psi_i^{W_i}$

s : "Stop", i.e., some m reached Step (7). Then we have:

- $0 = \Gamma_A(\langle x_m, y_m \rangle) = \Gamma_{A,s'''+1}(\langle x_m, y_m \rangle) = \Psi_{i,s'}^{W_{i,s'} \upharpoonright t'}(\langle x_m, y_m \rangle)$.
- Let u be the use for the computation $\Psi_{i,s'}^{W_{i,s'} \upharpoonright t'}(\langle x_m, y_m \rangle) = 0$ at Step (3). Notice that $u \leq t'$. Since at Step (5) we see $\Psi_{i,s''}^{W_{i,s''} \upharpoonright t''}(\langle x_m, y_m \rangle) = 1$, this implies that there exists an element $a \leq u$ such that $a \in W_{i,s''} \setminus W_{i,s'}$.
- Since $a \notin W_{i,s'}$, at Step (3) we have $0 = \Phi_{i,s'}^{\Gamma_{A,s'}}(a)$. Let v be the use for the computation $\Phi_{i,s'}^{\Gamma_{A,s'}}(a) = 0$.
- The embedding ξ from Step (6) guarantees that $\Gamma_A \upharpoonright v = \Gamma_{A,s'} \upharpoonright v$. Hence, $W_i(a) = W_{i,s''}(a) = 1 \neq 0 = \Phi_i^{\Gamma_A}(a)$.

Therefore, the requirement $\mathcal{P}_i$ is satisfied.

∞ : Eventually waiting at Step (6) for each $m \in \omega$. Then for each m, every $\mathcal{A}_{s'''}$ lacks an appropriate "semi-isomorphic" embedding ξ. This means that each $\mathcal{B}_m$ can be isomorphically embedded only once into $(\omega, <, f)$. As discussed above, this contradicts Lemma 20.

The current outcome of a strategy is equal to:

- s, if the strategy is already stopped.
- Otherwise, let m be the current (maximal) value of our strategy parameter m. If for this m, we wait at Step (3), then the outcome is w_m. If we wait at Step (5), then the outcome is w'_m.

Construction. We use the following ordering of the finitary outcomes: $s < \cdots < w_2' < w_2 < w_1' < w_1 < w_0' < w_0$. The tree of strategies includes only the finitary outcomes. More formally, we set $\Lambda = \{s\} \cup \{w_m, w_m' : m \in \omega\}$, and the tree T is equal to $\Lambda^{<\omega}$. The i-th level of the tree contains strategies $\alpha \in T$ devoted to the requirement $\mathcal{P}_i$.

If σ and τ are two finite strings from T, then by $\sigma^\frown\tau$ we denote the concatenation of σ and τ. As usual, we say that σ is *to the left* of τ if there exist some $\rho \in T$ and $o_1, o_2 \in \Lambda$ such that $o_1 < o_2$, $\sigma \supseteq \rho^\frown o_1$, and $\tau \supseteq \rho^\frown o_2$.

As usual, at a stage s of the construction we visit the strategies $\alpha_0, \alpha_1, \ldots, \alpha_s$, where $\alpha_0 = \emptyset$, and for each $i < s$ we have $\alpha_{i+1} = \alpha_i^\frown o$, where o is the current outcome of the strategy α_i. By g_s we denote the current finite path, i.e., the sequence $(\alpha_0, \alpha_1, \ldots, \alpha_s)$.

Verification. It is clear that the constructed $\mathcal{A}$ is a computable linear order on ω: indeed, if $x < y$ inside $\mathcal{A}_s$ for some $s \in \omega$, then $x <_{\mathcal{A}_t} y$ for all $t \geq s$.

Let g be the true path of the construction, i.e., the limit $\lim_s g_s$. More formally, for $k \in \omega$ and $\alpha \in T$, here we have

$$g(k) = \alpha \;\Leftrightarrow\; \exists t(\forall s \geq t)(g_s(k) = \alpha).$$

Note that in general, g could be a finite sequence.

▶ **Lemma 23.** *The path g is infinite, and every requirement $\mathcal{P}_i$ is satisfied.*

Proof. Suppose that a strategy α belongs to the true path g, and its associated requirement is $\mathcal{P}_i$. Consider the following three cases.

Case 1. There is a number m such that starting from some stage s, the outcome of α is always the same $o \in \{w_m, w_m'\}$. Then it is clear that $\alpha^\frown o$ belongs to g. In addition, for the number m, the strategy is forever stuck either at Step (3) or at Step (5). This means that $W_i \neq \Phi_i^{\Gamma_A}$ or $\Gamma_A \neq \Psi_i^{W_i}$.

Case 2. At some step α has outcome s. Then $\alpha^\frown s$ lies on the path g. In addition, $W_i \neq \Phi_i^{\Gamma_A}$ as discussed in the description of the outcome s.

Case 3. Otherwise, for each $m \in \omega$, α eventually goes through the outcomes w_m and w_m'. Since we never reach Step (7), this means that each finite structure $\mathcal{B}_m$ can be isomorphically embedded into $(\omega, <, f)$ only once: indeed, if some $\mathcal{B}_m$ could be embedded twice, then we could eventually use the second such embedding to recover the "semi-isomorphic" map ξ and to reach Step (7) for this m. Then, as discussed in the beginning of the proof of the theorem, Lemma 20 guarantees that Case 3 is impossible.

We conclude that the path g is infinite, and each $\mathcal{P}_i$ is satisfied. ◀

In order to finish the proof of Theorem 21, now it is sufficient to show that the order $\mathcal{A} = (\omega, <_A)$ is isomorphic to $(\omega, <)$.

Recall that at Step (6), we always choose a map ξ satisfying Condition $(**)$. Consider a $\mathcal{P}_i$-strategy α. Condition $(**)$ implies that for every $m \in \omega$ and every $x \in \mathcal{A}_m^{1,init}$, α never adds new elements which are $<_A$-below x.

Suppose that an element x is added to $\mathcal{A}$ by some strategy σ.

Consider an arbitrary strategy α. If α is to the right of σ, then α never works after the starting stage of σ. If $\alpha \supset \sigma$ or α is to the left of σ, then x always belongs to $\mathcal{A}_m^{1,init}$ of this particular α. Hence, α never adds elements $<_A$-below x.

We deduce that new elements which are $<_A$-below x could be added only by $\alpha \subseteq \sigma$. The proof of Lemma 23 implies that each such α adds only finitely many elements to $\mathcal{A}$. Therefore, for an arbitrary element x, $\mathcal{A}$ contains only finitely many elements $<_A$-less than x. This implies that $(\omega, <_A)$ is isomorphic to $(\omega, <)$. This concludes the proof for the case when each f-block is a cycle.

The general case. Now we discuss the general case of the theorem. The proof essentially follows the outline provided above, but we have to address two important details of how to implement the strategy for $\mathcal{P}_i$.

The first detail concerns Step (2), where one needs to choose two adjacent blocks $\tilde{\mathcal{C}}_m$ and $\tilde{\mathcal{D}}_m$. The question is how to *algorithmically* choose them?

Here the computability of the function $cp_f(x)$ is important – this fact guarantees that for a given $x \in \omega$, one can effectively recover the f-block I containing x. This effective recovery allows us to computably find the needed blocks $\tilde{\mathcal{C}}_m$ and $\tilde{\mathcal{D}}_m$.

The recovery of the block I can be arranged as follows. Without loss of generality, we may assume that x is the leftmost element of I. Since we know the value $cp_f(x)$, we can find all elements from the preimage $f^{-1}(\{x\})$. By using the function cp_f several times, we eventually find the *finite* set

$$P_f(x) = \bigcup_{j \in \omega} (f^{-1})^{(j)}(\{x\}).$$

If $P_f(x)$ already forms an f-block, then we stop the algorithm. Otherwise, there is (the least) $y_0 \notin P_f(x)$ such that $x < y_0 < \max P_f(x)$. We find the finite set $P_f(y_0)$. If the set $P_f(x) \cup P_f(y_0)$ forms an f-block, then we stop. Otherwise, again, we find the least $y_1 \notin P_f(x) \cup P_f(y_0)$ such that $y_0 < y_1 < \max(P_f(x) \cup P_f(y_0))$. We consider $P_f(x) \cup P_f(y_0) \cup P_f(y_1)$, etc. Since every f-block is finite, eventually we will find the desired f-block I.

The second detail concerns Step (2) and its interaction with Step (4). Since the block function f is not almost identity, f satisfies at least one of the following two conditions:

(a) There are infinitely many elements u such that u is the leftmost element of its block and $f(u) > u$.

(b) There are infinitely many pairs (u, v) such that u and v belong to the same block, u is the leftmost element of the block, $u < v$, and $f(v) = u$.

Assume that f satisfies (b) (the case of (a) could be treated in a similar way). Then in Step (2) of the strategy, we can always choose $\tilde{\mathcal{C}}_m, \tilde{\mathcal{D}}_m$ with the following property: the block $\tilde{\mathcal{D}}_m$ contains a pair $(\tilde{u}, \tilde{v})$ satisfying the condition described in item (b).

After building $\mathcal{A}^1_m$ (as described in Step (2)), we will choose x_m and y_m as follows. The element y_m is the rightmost element of $\mathcal{C}_m$, and the element x_m is the copy (inside $\mathcal{D}_m$) of the element $\tilde{v} \in \tilde{\mathcal{D}}_m$.

The intention behind this particular choice of x_m, y_m is the following. At the end of Step (2), we have $\langle x_m, y_m \rangle \notin \Gamma_A$. *In addition*, if we add *precisely one* element at Step (4), then we will immediately obtain that the condition $\langle x_m, y_m \rangle \in \Gamma_A$ becomes satisfied (since $\tilde{u} = f(\tilde{v})$ is the leftmost element of $\tilde{\mathcal{D}}_m$).

Taking the discussed details into account, one can arrange the proof of the general case in a straightforward manner. Theorem 21 is proved. ◀

4 Further Observations

In this section, we discuss some additional observations that could be interesting for a reader familiar with computable structure theory. First, we observe a connection with the paper [18] (more specifically, its Proposition 4.13), where a technique of recovering the successor is used. The main takeaway of the first observation is that our Theorem 21 cannot be deduced from this result.

Let $n \geq 1$, and let $k = \lceil n/2 \rceil$. A set $A \subseteq \omega^m$ is *n-c.e.* if there exist c.e. sets $U_1, V_1, U_2, V_2, \ldots, U_k, V_k$ such that $A = (U_1 \setminus V_1) \cup (U_2 \setminus V_2) \cup \cdots \cup (U_k \setminus V_k)$ and if n is odd, then $V_k = \emptyset$. This notion was introduced by Putnam in [28] where he proved

that a set A is n-c.e. if and only if there exists a recursive approximation of A (i.e., a recursive family of computable sets (A_s) satisfying $\lim A_s = A$) such that $A_0 = \emptyset$ and $card(\{t : A_t(x) \neq A_{t+1}(x)\}) \leq n$, for all $x \in \omega$. An m-ary relation R on $(\omega, <)$ is *intrinsically n-c.e.* if for every computable copy $\mathcal{B}$ of $(\omega, <)$, the set $R_{\mathcal{B}}$ is n-c.e.

The notion of an intrinsically n-c.e. relation could be extended to the transfinite levels of the Ershov hierarchy [11, 12, 13]: for a computable non-zero ordinal α, one can introduce the definition of an *intrinsically α-c.e.* relation. In addition, one can talk about being intrinsically α-c.e. *on a cone* (see Chapter 1 in [18]). Nevertheless, here we can omit the formal technical details: Harrison-Trainor (Proposition 4.12 in [18]) proved that if a relation R on $(\omega, <)$ is intrinsically α-c.e. on a cone, then R is intrinsically n-c.e. for some n. In the same paper, Harrison-Trainor (Proposition 4.13 in [18]) proved that every intrinsically n-c.e. relation R (on the structure $(\omega, <)$) satisfies (II). If every function f satisfying the premise of our Theorem 21 were intrinsically n-c.e. on $(\omega, <)$, then our result would follow from the aforementioned Proposition 4.13 in [18]. However, according to the following proposition, this is not the case.

▶ **Proposition 24.** *There exists a function f which is not intrinsically n-c.e., for any n, with the following properties: f is a computable block function, f is not almost identity, and f satisfies the effectiveness condition $(\star)$.*[3]

Proof. Let f be any computable block function such that for every $x \in \omega$, the f-block containing x is a cycle (the notion of cycle is defined in Example 18). It suffices to show that for each $n > 0$, there exists a computable copy $\mathcal{A} = (\omega, <_{\mathcal{A}})$ of $(\omega, <)$ such that the graph of $f_{\mathcal{A}}$ is not n-c.e.

Fix a standard effective listing $(X_i)_{i \in \omega}$ of all n-c.e. sets. We fix $n > 0$ and construct a desired computable copy $\mathcal{A}$ by finite injury priority argument. At each stage s, the current approximation $(A_s, <_{\mathcal{A}_s}; f_{\mathcal{A}_s})$ of $(\omega, <_{\mathcal{A}}; f_{\mathcal{A}})$ is isomorphic to the initial segment of $(\omega, <; f)$. Here is the requirement that we need to satisfy, for each natural number i:

The graph of $f_{\mathcal{A}}$ is not equal to X_i. $(\mathcal{R}_i)$

Strategy in isolation. When $\mathcal{R}_i$ enters the construction, say at stage s, we use fresh numbers to append to the current $\mathcal{A}_s$ a copy of the cycle that is immediately to the right of $\mathcal{A}_s$ in the standard copy. We thus obtain $\mathcal{A}_{s+1}$. Let u, v be the new cycle's endpoints (in $\mathcal{A}_{s+1}$) such that $f_{\mathcal{A}_{s+1}}(u) = v$. From now on, we will follow the computable approximation $X_{i,t}(\langle u, v\rangle)$ of our n-c.e. set. When we see that $X_{i,t}(\langle u, v\rangle) = 1$, then (if needed) we push things to the right (in an appropriate way) so that, after pushing, we have $f_{\mathcal{A}_{t+1}}(u) \neq v$. When we see that $X_{i,t}(\langle u, v\rangle) = 0$, then we push to the right to obtain $f_{\mathcal{A}_{t+1}}(u) = v$.

Observe that we can always do the "pushing". If f has only finitely many types of blocks, then we can arrange the construction in a way that each $\mathcal{R}_i$ is attached to a cycle that appears infinitely often. To obtain $f_{\mathcal{A}}(u) \neq v$, it suffices to insert one fresh number just before the cycle containing u, v. To obtain $f_{\mathcal{A}}(u) = v$ again, it suffices to search for the next occurrence of the cycle on which $\mathcal{R}_i$ was initialized and insert, right before v, sufficiently many fresh numbers so that v lands again on the first position of such a cycle (and u on the last).

[3] Notice that these properties correspond to the premise of Theorem 21.

If f has infinitely many types, then f has cycles of arbitrary length. In that case, obtaining $f_{\mathcal{A}}(u) \neq v$ is as above. To obtain $f_{\mathcal{A}}(u) = v$ it suffices to find a long enough cycle (appearing to the right of the current position of u), push to the right (by inserting fresh numbers just before v) and put the right amount of fresh numbers just after v so that u and v become endpoints again. ◀

▶ Remark 25. Observe that there are functions that satisfy the premise of Theorem 21 and have all c.e. degrees as a spectrum. For example, this will be true for any function of the following form: f is as in the proof of Proposition 24, and there exists an infinite c.e. set $L \subset \omega$ such that if $l \in L$, then there is precisely one f-block of cardinality l.

Our second observation looks at our property (II) from the point of view of computable categoricity. A computable structure $\mathcal{M}$ is *relatively computably categorical* if for any countable structure $\mathcal{N}$ such that $\mathrm{dom}(\mathcal{N}) = \omega$, there is an isomorphism $f\colon \mathcal{M} \cong \mathcal{N}$ such that f is computable in the atomic diagram $D(\mathcal{N})$.

Goncharov [16] proved that a computable structure $\mathcal{M}$ is relatively computably categorical if and only if there exists a c.e. family Θ of (finitary) existential formulas (with a fixed tuple of parameters $\bar{c}$ from $\mathcal{M}$) with the following properties:

- every tuple $\bar{a}$ from $\mathcal{M}$ satisfies some formula $\theta \in \Theta$;
- if $\bar{a}$ and $\bar{b}$ are tuples from $\mathcal{M}$ satisfying the same formula $\theta \in \Theta$, then $(\mathcal{M}, \bar{a}) \cong (\mathcal{M}, \bar{b})$.

Such a family Θ is usually called a *formally c.e. Scott family* for the structure $\mathcal{M}$.

▶ **Proposition 26.** *Let R be a computable relation on $(\omega, <)$. If the structure $(\omega, <, R)$ is relatively computably categorical, then the relation R satisfies* (II)*, i.e., for every computable copy $\mathcal{A}$ of $(\omega, <)$, we have $R_{\mathcal{A}} \geq_T Succ_{\mathcal{A}}$.*

Proof. Using a formally c.e. Scott family Θ for the structure $(\omega, <, R)$, we can obtain a finite tuple $\bar{c}$ and a computable sequence $(\psi_i(x, \bar{c}))_{i\in\omega}$ of existential formulas (in the language $\{<, R\}$) such that for each $k \in \omega$, the number k is the unique element x such that $(\omega, <, R) \models \psi_k(x, \bar{c})$.

Now let $\mathcal{A}$ be an arbitrary computable copy of $(\omega, <)$. Let $\bar{c}_{\mathcal{A}}$ be the isomorphic image (inside $\mathcal{A}$) of the tuple $\bar{c}$. Then we have:

- $y = Succ_{\mathcal{A}}(x)$ if and only if

$$(\mathcal{A}, R_{\mathcal{A}}) \models \bigvee_{i\in\omega} [\psi_i(x, \bar{c}_{\mathcal{A}}) \,\&\, \psi_{i+1}(y, \bar{c}_{\mathcal{A}})];$$

- $y \neq Succ_{\mathcal{A}}(x)$ if and only if

$$(\mathcal{A}, R_{\mathcal{A}}) \models (y \leq_{\mathcal{A}} x) \vee \bigvee_{i\in\omega,\, k\geq i+2} [\psi_i(x, \bar{c}_{\mathcal{A}}) \,\&\, \psi_k(y, \bar{c}_{\mathcal{A}})].$$

This implies that the graph of $Succ_{\mathcal{A}}$ is both c.e. and co-c.e. with respect to $R_{\mathcal{A}}$. Thus, we have $R_{\mathcal{A}} \geq_T Succ_{\mathcal{A}}$. ◀

5 Conclusions

The philosophical framework of Shapiro [34] is based on the observation that, strictly speaking, computations are performed on syntactic objects, such as strings of symbols, rather than on natural numbers themselves. This leads to the formal notion of a *notation*. In this paper, we re-cast this framework within the setting of computable structure theory. It turns out that there is a straightforward translation between the two approaches: a notation for a

given computable structure corresponds to a computable (isomorphic) copy of the structure. Surprisingly, this fact has been consistently overlooked in the literature (though, see [20]). The first contribution of this paper amounts to bringing this connection to the light, and showing, on the example of the structure $(\omega, <)$, how the two perspectives can inform each other.

In the context of the structure $(\omega, <)$, notations and computable structures find their common point in the property of computing the successor. The successor function was used by Shapiro to isolate the class of acceptable notations – those notations have particularly nice computational properties as functions computable in them are exactly the same as standard computable functions. The successor function has also useful applications in the investigation of degree spectra of computable relations on $(\omega, <)$; a particularly handy property is the recoverability of the successor from R on $(\omega, <)$ (which means that, in all computable copies of $(\omega, <)$, the image of the successor is computable relative to the image of R). This naturally leads us to the notion of relatively acceptable notation, i.e., a notation in which the successor is computable relative to a given additional relation.

The second, and main, contribution of the paper amounts to exploring the relationship between two properties of a computable relation R: (I) the degree spectrum of R on $(\omega, <)$ contains precisely the c.e. degrees, and (II) the successor is recoverable from R on $(\omega, <)$ (equivalently, every notation for $(\omega, <)$ is acceptable relative to R). It is a known fact in computable structure theory that (II) implies (I). Our main result (Theorem 21) concerns the reverse implication with regard to relations R that are graphs of unary total computable functions (actually, a particular subclass of so-called block functions, first considered in [2]). Theorem 21 proves that for a large class K of functions f, the two conditions given above are equivalent. Informally speaking, our proof uses the fact that the functions from the class K have pretty tame combinatorial properties.

As a concluding remark, we note that in general, Theorem 21 does not cover the class of all computable functions – even for the case of block functions. The proof of the following theorem is sketched in Appendix A.

▶ **Theorem 27.** *There exists a computable block function f such that:*

1. *the corresponding function $cp_f(x)$ is not computable;*
2. *each f-block occurs infinitely often in $(\omega, <, f)$;*
3. *the degree spectrum of f on $(\omega, <)$ contains all Δ_2 degrees.*

Nevertheless, we conjecture that the equivalence of (I) and (II) could be established for the class encompassing all unary total computable functions.

References

1 Chris J. Ash and Julia Knight. *Computable structures and the hyperarithmetical hierarchy*, volume 144 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, Amsterdam, 2000.

2 Nikolay Bazhenov, Dariusz Kalociński, and Michał Wrocławski. Intrinsic complexity of recursive functions on natural numbers with standard order. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2022.8.

3 Paul Benacerraf. What numbers could not be. *The Philosophical Review*, 74(1):47–73, 1965. doi:10.2307/2183530.

4 Paul Benacerraf. Recantation or Any old ω-sequence would do after all. *Philosophia Mathematica*, 4(2):184–189, 1996. doi:10.1093/philmat/4.2.184.

5 Jennifer Chubb, Andrey Frolov, and Valentina S. Harizanov. Degree spectra of the successor relation of computable linear orderings. *Archive for Mathematical Logic*, 48(1):7–13, 2009. doi:10.1007/s00153-008-0110-6.

6 Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936. doi:10.2307/2371045.

7 S. Barry Cooper. *Degrees of unsolvability*. PhD thesis, University of Leicester, 1971.

8 Brian Jack Copeland and Diane Proudfoot. Deviant encodings and Turing's analysis of computability. *Studies in History and Philosophy of Science Part A*, 41(3):247–252, 2010.

9 Nigel Cutland. *Computability: An Introduction to Recursive Function Theory*. Cambridge University Press, Cambridge, MA, 1980.

10 Rod Downey, Bakhadyr Khoussainov, Joseph S. Miller, and Liang Yu. Degree spectra of unary relations on $(\omega, \leq)$. In *Logic, Methodology and Philosophy of Science: Proceedings of the Thirteenth International Congress*, pages 35–55. College Publications, 2009. URL: http://homepages.mcs.vuw.ac.nz/~downey/publications/LOJan24.pdf.

11 Yurii L. Ershov. A hierarchy of sets. I. *Algebra and Logic*, 7(1):25–43, 1968. doi:10.1007/BF02218750.

12 Yurii L. Ershov. On a hierarchy of sets, II. *Algebra and Logic*, 7(4):212–232, 1968. doi:10.1007/BF02218664.

13 Yurii L. Ershov. On a hierarchy of sets. III. *Algebra and Logic*, 9(1):20–31, 1970. doi:10.1007/BF02219847.

14 Ekaterina B. Fokina, Valentina S. Harizanov, and Alexander Melnikov. Computable model theory. In R. Downey, editor, *Turing's legacy: Developments from Turing's ideas in logic*, volume 42 of *Lecture Notes in Logic*, pages 124–194. Cambridge University Press, Cambridge, 2014. doi:10.1017/CBO9781107338579.006.

15 E. Mark Gold. Limiting Recursion. *Journal of Symbolic Logic*, 30(1):28–48, 1965. doi:10.2307/2270580.

16 Sergey S. Goncharov. Autostability and computable families of constructivizations. *Algebra and Logic*, 14(6):392–409, 1975. doi:10.1007/BF01668470.

17 Valentina S. Harizanov. *Degree spectrum of a recursive relation on a recursive structure*. PhD thesis, University of Wisconsin-Madinson, 1987.

18 Matthew Harrison-Trainor. Degree spectra of relations on a cone. *Memoirs of the American Mathematical Society*, 253(1208):1–120, 2018. doi:10.1090/memo/1208.

19 Denis R. Hirschfeldt. Degree spectra of relations on computable structures. *Bulletin of Symbolic Logic*, 6(2):197–212, 2000. doi:10.2307/421207.

20 Dariusz Kalociński and Michał Wrocławski. Generalization of Shapiro's theorem to higher arities and noninjective notations. *Archive for Mathematical Logic*, September 2022. doi:10.1007/s00153-022-00836-4.

21 Julia F. Knight. Degrees coded in jumps of orderings. *Journal of Symbolic Logic*, 51(4):1034–1042, 1986. doi:10.2307/2273915.

22 Steffen Lempp. Priority arguments in computability theory, model theory, and complexity theory, 2012. preprint available on the author's webpage. URL: https://people.math.wisc.edu/~lempp/papers/prio.pdf.

23 Anatolii I. Mal'tsev. Constructive algebras. I. *Russian Mathematical Surveys*, 16(3):77–129, 1961. doi:10.1070/RM1961v016n03ABEH001120.

24 Anatolii I. Mal'tsev. On recursive abelian groups. *Soviet Mathematics. Doklady*, 3:1431–1434, 1962.

25 Andrei A. Markov. The theory of algorithms. *Trudy Matematicheskogo Instituta imeni V.A. Steklova*, 38:176–189, 1951. in Russian.

26 Antonio Montalbán. *Computable structure theory: Within the arithmetic*. Cambridge University Press, 2021.

27 Michael Moses. Relations intrinsically recursive in linear orders. *Mathematical Logic Quarterly*, 32(25-30):467–472, 1986. doi:10.1002/malq.19860322514.

28 Hilary Putnam. Trial and Error Predicates and the Solution to a Problem of Mostowski. *Journal of Symbolic Logic*, 30(1):49–57, 1965. doi:10.2307/2270581.

29 Paula Quinon. A taxonomy of deviant encodings. In Florin Manea, Russell G. Miller, and Dirk Nowotka, editors, *Sailing Routes in the World of Computation*, volume 10936 of *LNCS*, pages 338–348, Cham, 2018. Springer International Publishing.

30 Michael Rescorla. Church's Thesis and the conceptual analysis of computability. *Notre Dame Journal of Formal Logic*, 48(2):253–280, 2007. doi:10.1305/ndjfl/1179323267.

31 Linda J. C. Richter. *Degrees of Unsolvability of Models*. PhD thesis, University of Illinois at Urbana-Champaign, 1977.

32 Linda J. C. Richter. Degrees of structures. *Journal of Symbolic Logic*, 46(4):723–731, 1981. doi:10.2307/2273222.

33 Hartley Rogers. *Theory of recursive functions and effective computability*. MIT Press, Cambridge, MA, 1987.

34 Stewart Shapiro. Acceptable notation. *Notre Dame Journal of Formal Logic*, 23(1):14–20, 1982. doi:10.1305/ndjfl/1093883561.

35 Stewart Shapiro, Eric Snyder, and Richard Samuels. Computability, notation, and de re knowledge of numbers. *Philosophies*, 7(1), 2022. doi:10.3390/philosophies7010020.

36 John C. Shepherdson and Howard E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10(2):217–255, 1963. doi:10.1145/321160.321170.

37 Robert I. Soare. *Recursively Enumerable Sets and Degrees*. Springer-Verlag, New York, 1987.

38 Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Ser. 2*, 42:230–265, 1936. doi:10.1112/plms/s2-42.1.230.

39 Matthew Wright. Degrees of relations on ordinals. *Computability*, 7(4):349–365, 2018. doi:10.3233/COM-180086.

40 Michał Wrocławski. *Representations of numbers and their computational properties*. PhD thesis, University of Warsaw, Warsaw, 2019.

A Proof of Theorem 27

▶ **Theorem 27.** *There exists a computable block function f such that:*

1. *the corresponding function $cp_f(x)$ is not computable;*
2. *each f-block occurs infinitely often in $(\omega, <, f)$;*
3. *the degree spectrum of f on $(\omega, <)$ contains all Δ_2 degrees.*

Proof Sketch. As usual, for $i \in \omega$, φ_i denotes the unary partial computable function which has Gödel number i.

We build a computable block function f satisfying the following series of requirements:

$$\text{The function } cp_f \text{ is not equal to } \varphi_i. \qquad (\mathcal{P}_i)$$

$$\text{Each } f\text{-block occurs infinitely often inside } (\omega, <, f). \qquad (\mathcal{R})$$

At a stage s, we define the finite function $f \upharpoonright N_s$ (where $N_s \in \omega$ and $N_s < N_{s+1}$) in such a way that $(\{0, 1, \ldots, N_s - 1\}, <, f \upharpoonright N_s)$ consists of blocks.

Beforehand, we put $f(0) = 0$, $f(1) = 2$, and $f(2) = 1$.

The $\mathcal{R}$-strategy is a global one. The $\mathcal{R}$-requirement is satisfied in a simple way, as follows. At the end of each construction stage s, let B_s be the current set of all (isomorphism types of) blocks. Then (before starting the stage $s + 1$) we use fresh numbers x (i.e., the least numbers such that $f(x)$ is not defined yet) to extend f in the following way: for each block I from B_s, we add *precisely two* adjacent copies of I.

Note that in addition to the $\mathcal{R}$-requirement, this procedure will guarantee that in the final structure $(\omega, <, f)$, there will be infinitely many pairs $(x, x+1)$ such that $f(x) = x$ and $f(x+1) = x+1$. This preliminary observation will help us in the verification.

Strategy for $\mathcal{P}_i$. Suppose that the strategy starts working at a stage $s_0 + 1$.

(1) Choose a large fresh size l_i such that the cycle of size l_i cannot be isomorphically embedded into the current structure $(\{0, 1, \ldots, N_{s_0} - 1\}, <, f \upharpoonright N_{s_0})$. (See Example 18 for the definition of a cycle).
Extend f by adding a copy of the cycle of size l_i. Let w_i be the leftmost element of this copy.

(2) Wait for a stage $s' > s_0 + 1$ such that $\varphi_{i,s'}(w_i)\downarrow = 1$.

(3) Extend f by taking the least number x such that $f(x)$ is still undefined, and setting $f(x) := w_i$.
This concludes the description of the strategy. It is clear that it satisfies the requirement $\mathcal{R}_i$. Indeed, assume that the function φ_i is total. If $\varphi_i(w_i) \neq 1$, then the element w_i is a part of a cycle, and $cp_f(w_i) = 1 \neq \varphi_i(w_i)$. If $\varphi_i(w_i) = 1$, then we have $cp_f(w_i) = 2$.

Construction. The construction is arranged as a standard finite injury argument. The requirements are ordered: $\mathcal{P}_0 < \mathcal{P}_1 < \mathcal{P}_2 < \ldots$. When a higher priority strategy $\mathcal{P}_i$ acts (i.e., it extends f in its Step (3)), it initializes all lower priority strategies $\mathcal{P}_j$, $j > i$.

Verification. The non-computability of the function $cp_f(x)$ can be proved by a standard argument for finite injury constructions.

A more hard part is how to show that every Δ_2 degree belongs to the degree spectrum of the constructed f. This can be achieved by arranging an argument similar to the argument of Case (a) of Theorem 14 in [2]. Roughly speaking, given an arbitrary Δ_2 set X, we should encode it via a computable copy $\mathcal{A} = (\omega, <_A)$ as follows. For each $e \in \omega$, the numbers $4e$ and $4e+2$ will be $<_A$-adjacent, and more importantly:

- if $e \in X$, then $f_{\mathcal{A}}(4e) = 4e+2$ and $f_{\mathcal{A}}(4e+2) = 4e$;
- if $e \notin X$, then $f_{\mathcal{A}}(4e) = 4e$ and $f_{\mathcal{A}}(4e+2) = 4e+2$.

Our preliminary observation helps us to arrange this encoding. This concludes the proof sketch of Theorem 27. ◀

Hennessy-Milner Theorems via Galois Connections

Harsh Beohar
University of Sheffield, UK

Sebastian Gurke
Universität Duisburg-Essen, Germany

Barbara König
Universität Duisburg-Essen, Germany

Karla Messing
Universität Duisburg-Essen, Germany

Abstract

We introduce a general and compositional, yet simple, framework that allows to derive soundness and expressiveness results for modal logics characterizing behavioural equivalences or metrics (also known as Hennessy-Milner theorems). It is based on Galois connections between sets of (real-valued) predicates on the one hand and equivalence relations/metrics on the other hand and covers a part of the linear-time-branching-time spectrum, both for the qualitative case (behavioural equivalences) and the quantitative case (behavioural metrics). We derive behaviour functions from a given logic and give a condition, called compatibility, that characterizes under which conditions a logically induced equivalence/metric is induced by a fixpoint equation. In particular, this framework allows to derive a new fixpoint characterization of directed trace metrics.

2012 ACM Subject Classification Theory of computation → Concurrency; Theory of computation → Modal and temporal logics

Keywords and phrases behavioural equivalences and metrics, modal logics, Galois connections

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.12

Related Version *Full Version*: `https://arxiv.org/pdf/2207.05407.pdf`

Funding The first author was partially supported by the EPSRC NIA Grant EP/X019373/1, while the remaining authors were partially supported by the DFG project SpeQt.

Acknowledgements We want to thank Jonas Forster, Lutz Schröder and Paul Wild for several interesting discussions on the topics of this paper.

1 Introduction

In the verification of state-based transition systems, modal logics play a central role: they can be used to specify the properties that a system must satisfy and model-checking techniques allow to verify whether this is in fact the case. Modal logics also play a fundamental role in characterizing behavioural equivalences: van Glabbeek in his seminal paper [25] showed how a whole spectrum of behavioural equivalences and preorders can be characterized via modal logics. This characterization is also known as the Hennessy-Milner theorem [9], which says that two states x, y are equivalent (wrt. to some notion of behavioural equivalence) iff they satisfy the same formulas ϕ (of a given modal logic). Formally, $x \sim y \iff \forall \phi\colon (x \models \phi \iff y \models \phi)$.

For quantitative systems, the notion of behavioural equivalence is often too strict and small deviations in quantitative information, such as probabilities, can cause two states that intuitively behave very much alike to be inequivalent in a formal sense. Hence it is natural to consider various metrics for determining at what behavioural distance two states lie [7, 24]. This yields an extension of classical notions of behavioural equivalence which knows only

distance 0 (two states behave the same) and distance 1 (two states behave differently). Such metrics have often been studied in probabilistic settings [7], but they can be studied in other quantitative contexts, for instance metric transition systems [6, 8].

In the quantitative case, equivalences are replaced by pseudo-metrics and evaluation of a formula ϕ results in a real-valued (as opposed to a boolean-valued) function $[\![\phi]\!]$, intuitively indicating to which degree a state satisfies a formula. Stated in this context the Hennessy-Milner property says that $d(x,y) = \bigvee_\phi |[\![\phi]\!](x) - [\![\phi]\!](y)|$, where d is the behavioural metric.

We present a general framework that allows to easily deduce the Hennessy-Milner property for a variety of equivalences, preorders and (directed) metrics in the qualitative and quantitative setting. We rely on a well-known property [2, 4, 5] for Galois connections that says under which conditions left adjoints preserve least fixpoints. Such Galois connections relate the logical with the behavioural universe and translate sets of (real-valued) predicates to equivalences (metrics) and vice versa. Our *first* contribution is the identification of adjunctions both in quantitative/qualitative settings, which are crucial in capturing bisimilarity and (decorated) trace versions of equivalences/preorders/metrics.

While most contributions to this area start with a behavioural equivalence (resp. metric) and define a corresponding characteristic logic, our approach goes in the other direction, with the slogan: *"Derive behaviour functions from a modal logic"*. The recipe, which is our *second* contribution, is as follows: we define a logic function living in the logical universe and check that it is compatible with the closure induced by the Galois connection. Compatibility ensures that the Hennessy-Milner property is satisfied when we transfer the logic function into a behaviour function living in the behavioural universe. More concretely, we can guarantee that the least fixpoint of the logic function (the set of all formulas) induces an equivalence (resp. metric) which is the least fixpoint of the behaviour function. Note that in the qualitative case, the Galois connection is contravariant, resulting in behavioural equivalence being the greatest fixpoint, as usual.

Related ideas have been considered in more categorical settings [13, 18], here we demonstrate that this can be done in a purely lattice-theoretical setup and in particular for behavioural metrics. To our knowledge, the adjunctions that we are considering here, have not yet been used to derive Hennessy-Milner theorems and behaviour functions. Our *third* contribution is the novel connection to up-to functions and compatibility and we show how closure properties for up-to functions can be employed to combine logics, leading to a modular framework. Furthermore, the behaviour function that we obtain for the trace metric case is, as far as we know, not yet known in the literature. Our *final* contribution is the characterisation of these behaviour functions in more concrete terms both in the qualitative (Theorem 4.12 and Corollary 4.14) and quantitative (Theorem 5.17 and Corollary 5.22) cases. In turn, these general results effortlessly instantiate into many of the equivalences in the van Glabbeek spectrum and immediately yield: logical characterizations, the hierarchy between them and also recursive characterizations, which are often hard to obtain (at least in the metric case).

The full version of this paper, including all proofs, is available from [3].

2 Preliminaries

Functions and Relations

Given a function $f\colon X \to Y$ and $Z \subseteq X$ we write $f[Z]$ for $\{f(z) \mid z \in Z\}$. Similarly, for a relation $R \subseteq X \times X$ and $X' \subseteq X$, we define $R[X'] = \{y \in X \mid \exists x \in X'\colon (x,y) \in R\}$. Furthermore, Y^X denotes the set of all functions from X to Y and, for a given set $\mathcal{F} \subseteq Y^X$ of functions, by $\langle\mathcal{F}\rangle$ we denote a function of type $X \to Y^{\mathcal{F}}$ defined as $\langle\mathcal{F}\rangle(x)(f) = f(x)$. For $S \subseteq X$, $\chi_S\colon X \to \{0,1\}$ stands for the characteristic function of S.

A *congruence* is an equivalence relation $R \subseteq \mathcal{P}(X) \times \mathcal{P}(X)$ satisfying: $\bigcup_{i\in I} X_i \, R \bigcup_{i\in I} Y_i$ whenever $X_i \, R \, Y_i$ for all $i \in I$. Given any relation $R \subseteq \mathcal{P}(X) \times \mathcal{P}(X)$, by $cong(R)$ we denote its *congruence closure*, i.e., the smallest congruence such that $R \subseteq cong(R)$.

The directed relation lifting $R_{\overrightarrow{H}} \subseteq \mathcal{P}(X) \times \mathcal{P}(X)$ for a relation $R \subseteq X \times X$ is defined as $X_1 \, R_{\overrightarrow{H}} \, X_2 \iff \forall x_1 \in X_1 \exists x_2 \in X_2 \colon x_1 \, R \, x_2$. Furthermore, we write $R_H = R_{\overrightarrow{H}} \cap (R_{\overrightarrow{H}})^{-1}$, which can be seen as a special case of the Hausdorff distance (see below).

Pseudo-metrics

We use truncated addition and subtraction on the interval $[0,1]$, i.e., for $r, s \in [0,1]$ we have $r \oplus s = \min\{r+s, 1\}$, $r \ominus s = \max\{0, r-s\}$.

A *directed pseudo-metric* or *hemimetric* on a set X is a function $d \colon X \times X \to [0,1]$ such that for all $x, y, z \in X$ (i) $d(x,x) = 0$, (ii) $d(x,z) \leq d(x,y) \oplus d(y,z)$. It is called a *pseudo-metric* if in addition (iii) $d(x,y) = d(y,x)$ for all $x, y \in X$. Whenever $d(x,y) = 0$ implies $x = y$ we drop the prefix "pseudo-" and call d a *metric*. Given a directed pseudo-metric d on X, $\overline{d}$ refers to the *symmetrization* of d, i.e., $\overline{d}(x,y) = \max\{d(x,y), d(y,x)\}$, for every $x, y \in X$. Some examples of metrics used in this paper are the following:

- The discrete metric d_{disc} on a set A is $d_{\text{disc}}(a,b) = 1$ if $a \neq b$ and 0 otherwise.
- The *Euclidean* distance d on the interval $[0,1]$ given by $d(r,r') = |r - r'|$.
- The *sup-metric* d on $[0,1]^I$ is given by $d(p,p') = \sup_{i\in I} |p(i) - p'(i)|$.
- The product of two (pseudo)metric spaces (X, d_X) and (Y, d_Y) is a (pseudo)metric space $(X \times Y, d_X \otimes d_Y)$, where $(d_X \otimes d_Y)((x,y),(x',y')) = \max\{d_X(x,x'), d_Y(y,y')\}$.
- The directed Hausdorff lifting $d_{\overrightarrow{H}}$ of a pseudo-metric space (X,d) is a directed pseudo-metric on the power set $\mathcal{P}(X)$ given by $d_{\overrightarrow{H}}(U,V) = \sup_{x\in U} \inf_{y\in V} d(x,y)$. Intuitively, the Hausdorff distance between two sets is the farthest that any element of one set has to "travel" to reach the other set.

  It can equivalently be characterized as the infimum $d_{\overrightarrow{H}}(U,V) = \bigwedge\{\varepsilon \in [0,1] \mid U \subseteq V_\varepsilon\}$, where $V_\varepsilon = \{x \in X \mid \bigwedge_{v\in V} d(x,v) \leq \varepsilon\}$. This means that we are looking for the least ε such that U is included in the union of all ε-balls around elements of V.

  Moreover, the Hausdorff lifting d_H of a pseudo-metric d is the symmetrization of $d_{\overrightarrow{H}}$.

Given a directed pseudo-metric $d \colon X \times X \to [0,1]$, a function $f \colon X \to [0,1]$ is called *non-expansive wrt. d* whenever for all $x, y \in X$: $f(x) \ominus f(y) \leq d(x,y)$.

Lattices, Fixpoints and Galois Connections

A *complete lattice* $(\mathbb{L}, \sqsubseteq)$ consists of a set $\mathbb{L}$ with a partial order $\sqsubseteq$ such that each $Y \subseteq \mathbb{L}$ has a least upper bound $\bigsqcup Y$ (also called supremum, join) and a greatest lower bound $\bigsqcap Y$ (also called infimum, meet). In particular, $\mathbb{L}$ has a bottom element $\bot = \bigsqcap \mathbb{L}$ and a top element $\top = \bigsqcup \mathbb{L}$. Whenever the order is clear from the context, we simply write $\mathbb{L}$ for a complete lattice. For example:

- $([0,1], \leq)$ has a lattice structure with infimum $\bigwedge$ and supremum $\bigvee$.
- The set $Eq(X)$ ($Pre(X)$) of equivalences (preorders) on X is a lattice with $\bigsqcup = \bigcap$ and the join $\bigsqcup \mathcal{R}$ is the least equivalence (resp. preorder) generated by $\bigcup \mathcal{R}$.
- The set $PMet(X)$ ($DPMet(X)$) of (directed) pseudo-metrics is lattice-ordered by $\leq$.

Via the Knaster-Tarski theorem it is well-known that any monotone function $f \colon \mathbb{L} \to \mathbb{L}$ on a complete lattice $\mathbb{L}$ has a *least fixpoint* μf and a *greatest fixpoint* νf.

Let $\mathbb{L}$, $\mathbb{B}$ be two lattices. A *Galois connection* from $\mathbb{L}$ to $\mathbb{B}$ is a pair $\alpha \dashv \gamma$ of monotone functions $\alpha \colon \mathbb{L} \to \mathbb{B}$, $\gamma \colon \mathbb{B} \to \mathbb{L}$ such that for all $\ell \in L$: $\ell \sqsubseteq \gamma(\alpha(\ell))$ and for all $m \in \mathbb{B}$: $\alpha(\gamma(m)) \sqsubseteq m$. Equivalently, $\alpha(\ell) \sqsubseteq m \iff \ell \sqsubseteq \gamma(m)$, for all $\ell \in \mathbb{L}, m \in \mathbb{B}$. The function α (resp. γ) is also called the *left* (resp. *right*) *adjoint* and it preserves arbitrary joins (meets).

For an arbitrary function f, we define f^ω as $f^\omega(x) = \bigsqcup_{i\in\mathbb{N}} f^i(x)$. Given a function $f\colon X \to [0,1]$, the function $\tilde{f}\colon \mathcal{P}(X) \to [0,1]$ denotes the join-preserving function generated by f and is defined as $\tilde{f}(X') = \bigvee_{x\in X'} f(x)$ (for $X' \subseteq X$).

Closures

A *closure* c is a monotone, idempotent and extensive (i.e. $x \sqsubseteq c(x)$ for all x) function on a lattice. Given a Galois connection $\alpha \dashv \gamma$, the map $\gamma \circ \alpha$ is always a closure.

Given a set Z, a family $\mathcal{O}$ of operators on Z (of arbitrary, possibly infinite, arity) and a subset $Z' \subseteq Z$, we denote by $\mathrm{cl}^{\mathcal{O}}(Z')$ the least superset of Z' that is closed under all the operators from $\mathcal{O}$. The set $\mathcal{O}$ will sometimes be left implicit in favour of a more suggestive notation. For instance, given a set $\mathcal{S} \subseteq \mathcal{P}(X)$, $\mathrm{cl}^{\cup}(\mathcal{S})$ closes $\mathcal{S}$ under arbitrary unions and $\mathrm{cl}^{\cup,\cap}(\mathcal{S})$ under arbitrary unions and intersections. On the other hand $\mathrm{cl}_f^{\mathcal{O}}$ closes only under operators in $\mathcal{O}$ of finite arity (such as finite unions or intersections). Clearly, $\mathrm{cl}^{\mathcal{O}}$ and $\mathrm{cl}_f^{\mathcal{O}}$ are closures in the above sense.

A special case is the shift, where, given a set $\mathcal{F} \subseteq [0,1]^X$, $\mathrm{cl}^{\mathrm{sh}}(\mathcal{F})$ is the closure under constant shifts, i.e., operations $f \mapsto f \ominus c$, $f \mapsto f \oplus c$ for $c \in [0,1]$.

We end this subsection by a technical result which is needed to show that our "logic" function (cf. Section 3) is continuous.

▶ **Lemma 2.1.** *Let $(F_i \subseteq Z)_{i\in\mathbb{N}}$ be an increasing family of sets, i.e., $F_i \subseteq F_{i+1}$ for every $i \in \mathbb{N}$. If the set $\mathcal{O}$ (of operators on Z) contains operators of only finite arity, then $\mathrm{cl}^{\mathcal{O}}(\bigcup_{i\in\mathbb{N}} F_i) = \bigcup_{i\in\mathbb{N}} \mathrm{cl}^{\mathcal{O}}(F_i)$.*

Transition Systems

We will restrict to systems of the following kind in this paper.

▶ **Definition 2.2** ((Metric) Transition Systems). *A* transition system *over an alphabet A is a pair $(X, \to)$ consisting of a state space X and a transition relation $\to \subseteq X \times A \times X$. We write $x \xrightarrow{a} x'$ for $(x,a,x') \in \to$. For $x \in X$, $\delta(x) = \{(a,x') \mid x \xrightarrow{a} x'\}$ and $\delta_a(x)$ denotes the a-successors of x. A transition system is* finitely branching *if $\delta(x)$ is finite for every x.*

For a set $\Delta \subseteq A \times X$ we denote by $\mathrm{lab}(\Delta)$ *the set of labels of Δ, in other words the projection to the first argument, i.e.* $\mathrm{lab}(\Delta) = \{a \mid \exists x \in X\colon (a,x) \in \Delta\}$. *Similarly* $\mathrm{tgt}(\Delta)$ *is the set of targets and projects to the second argument.*

A metric transition system *over A is a triple $(X, \to, d_A)$ with a metric $d_A\colon A \times A \to [0,1]$.*

▶ **Definition 2.3** (Traces). *For $x \in X, \sigma = a_1 \cdots a_n \in A^*$, we write $x \xrightarrow{\sigma} x'$ if $x \xrightarrow{a_1} \cdots \xrightarrow{a_n} x'$ and define $\mathrm{Tr}(x) = \{\sigma \mid \exists x'\colon x \xrightarrow{\sigma} x'\}$. We extend δ, δ_a to sequences $\hat{\delta}, \hat{\delta}_\sigma$ in the obvious way.*

Given a metric transition system, the distance of two traces is defined as $d_{\mathrm{Tr}}\colon A^* \times A^* \to [0,1]$ where $d_{\mathrm{Tr}}(\sigma_1,\sigma_2) = 1$ if $|\sigma_1| \neq |\sigma_2|$, $d_{\mathrm{Tr}}(\varepsilon,\varepsilon) = 0$ and $d_{\mathrm{Tr}}(a_1\sigma_1', a_2\sigma_2') = \max\{d_A(a_1,a_2), d_{\mathrm{Tr}}(\sigma_1',\sigma_2')\}$ (sup-metric).

3 General Framework

Our results are based on the following theorem that shows how fixpoints are preserved by Galois connections, a well-known property, see for instance [2, 4, 5].

We first introduce the notion of compatibility that has been studied in connection with up-to techniques, enhancing coinductive proofs [23].

▶ **Definition 3.1.** *Let* $\log, c\colon \mathbb{L} \to \mathbb{L}$ *be two monotone endo-functions on a lattice* $\mathbb{L}$*. We call* $\log$ *c-compatible whenever* $\log \circ c \sqsubseteq c \circ \log$.

▶ **Theorem 3.2.** *Let* $\mathbb{L}, \mathbb{B}$ *be two complete lattices with a Galois connection* $\alpha\colon \mathbb{L} \to \mathbb{B}$, $\gamma\colon \mathbb{B} \to \mathbb{L}$ *and two monotone endo-functions* $\log\colon \mathbb{L} \to \mathbb{L}$, $\mathrm{beh}\colon \mathbb{B} \to \mathbb{B}$.

1. *Then* $\alpha \circ \log = \mathrm{beh} \circ \alpha$ *implies* $\alpha(\mu\, \log) = \mu\, \mathrm{beh}$.
2. *Let* $c = \gamma \circ \alpha$ *be the closure operator corresponding to the Galois connection and assume that* $\mathrm{beh} = \alpha \circ \log \circ \gamma$*. Then c-compatibility of* $\log$ *implies* $\alpha(\mu\, \log) = \mu\, \mathrm{beh}$.
3. *Whenever* $\alpha \circ \log = \mathrm{beh} \circ \alpha$ *and* $\log$ *reaches its fixpoint in* ω *steps, i.e.,* $\mu \log = \log^{\omega}(\bot)$*, so does* beh.

Here $\mathbb{L}$ is the universe in which the logic lives and $\mathbb{B}$ is the universe in which equivalences respectively metrics live. Furthermore log is the "logic function", constructing modal logic formulas, and μ log will be the set of all formulas. On the other hand, beh is the "behaviour function" whose least (respectively greatest) fixpoint is the behavioural metric (equivalence).

▶ Remark 3.3. Note that the above theorem is true even in more general situations, for example if $\mathbb{L}$ and $\mathbb{B}$ are only assumed to be complete partial orders. We however stick to complete lattices since they are more widely known. Also, on a complete lattice many notions of continuity, such as Scott-continuity or chain-continuity, coincide [19]. In the following we will therefore simply say that a monotone function on $\mathbb{L}$ or $\mathbb{B}$ is continuous if it preserves suprema of all (well-ordered) chains.

The recipe used in this paper is the following: first, define a logical universe $\mathbb{L}$ and a logic function $\log\colon \mathbb{L} \to \mathbb{L}$. Then choose a suitable Galois connection $\alpha \dashv \gamma$ to a behaviour universe $\mathbb{B}$ and show that log is c-compatible, where $c = \gamma \circ \alpha$ is the closure associated to the Galois connection. Then derive the behaviour function $\mathrm{beh} = \alpha \circ \log \circ \gamma\colon \mathbb{B} \to \mathbb{B}$ and from the results above, we automatically obtain the equality $\alpha(\mu\, \log) = \mu\, \mathrm{beh}$, which tells us that logical and behavioural equivalence respectively distance coincide (Hennessy-Milner theorem). This will be worked out in the following examples.

Combining logic functions results in the combination of the corresponding behaviour functions, which is essential in establishing Hennessy-Milner theorems compositionally.

▶ **Proposition 3.4.** *Let* $i \in \{1, 2\}$ *and* $\log_i, c\colon \mathbb{L} \to \mathbb{L}$ *be monotone functions on a complete lattice* $\mathbb{L}$ *such that* $\log_i$ *are c-compatible. Then* $\log_1 \sqcup \log_2$ *and* $\log_1 \circ \log_2$ *are also c-compatible.*

Let $\mathrm{beh}_i = \alpha \circ \log_i \circ \gamma$ *be the behaviour functions corresponding to* $\log_i$*. Then the behaviour functions of* $\log_1 \sqcup \log_2$ *and* $\log_1 \circ \log_2$ *are, respectively,* $\mathrm{beh}_1 \sqcup \mathrm{beh}_2$ *and* $\mathrm{beh}_1 \circ \mathrm{beh}_2$.

Furthermore every constant function k and the identity are c-compatible. Their corresponding behaviour functions are the constant function $b \mapsto \alpha(\ell)$ (where ℓ is the constant value of k) respectively the co-closure $\alpha \circ \gamma$.

We are using techniques for the construction of up-to functions studied in [23], but we are using them in a non-standard way. The point is subtle since the closure is usually supposed to be the up-to function, while in our notion of compatibility the logic function plays this role. Furthermore we are interested in least fixpoints, while the results of [23] consider post-fixpoints up-to in order to show that a lattice element is below the greatest fixpoint.

We end this section by characterising the compatibility property when the closure c is induced by an adjoint situation $\alpha \dashv \gamma$ (as in Theorem 3.2). This result is in turn used to relate with the notion of approximating family of predicates [14] in Section 6.

▶ **Lemma 3.5.** *Let* $\alpha \dashv \gamma$ *be a Galois connection between lattices* $\mathbb{L}$, $\mathbb{B}$ *(with* $c = \gamma \circ \alpha$*) and let* $\log\colon \mathbb{L} \to \mathbb{L}$ *be a monotone function. Furthermore let* $\ell \in \mathbb{L}$*. Then* $\log(c(\ell)) \sqsubseteq c(\log(\ell))$ *iff*

$$\forall \ell' \in \mathbb{L}\colon \big(\alpha(\ell') \sqsubseteq \alpha(\ell) \implies \alpha(\log(\ell')) \sqsubseteq \alpha(\log(\ell))\big).$$

4 Qualitative Case

We will start with the classical, qualitative case with behavioural equivalences on the one side and boolean-valued modal logics on the other side. In this way we will recreate parts of the theory of [25], incorporating it into the setting of adjunctions as described earlier. Throughout this section we fix a transition system $(X, \to)$ over A.

4.1 Bisimilarity

For bisimilarity we work with the lattices $\mathbb{L} = (\mathcal{P}(\mathcal{P}(X)), \subseteq)$ and $\mathbb{B} = (Eq(X), \supseteq)$. The Galois connection is given as follows, where $[x]_R$ is the equivalence class of x wrt. R:

$$\begin{aligned} \alpha_{\mathrm{b}}(\mathcal{S}) &= \{(x,x') \in X \times X \mid \forall S \in \mathcal{S} \colon (x \in S \iff x' \in S)\} \\ \gamma_{\mathrm{b}}(R) &= \{S \subseteq X \mid \forall (x,x') \in R \colon (x \in S \iff x' \in S)\} = \left\{ \bigcup \{[x]_R \mid x \in S\} \mid S \subseteq X \right\}. \end{aligned}$$

Intuitively α_{b} generates an equivalence on X from a set of subsets of X and γ_{b} maps an equivalence to all subsets of X that are closed under this equivalence. Both functions are monotone and it is easy to see from the definition that it is indeed a Galois connection (see also Proposition 4.2 below). As logic function we consider $\log_{\mathrm{b}} \colon \mathcal{P}(\mathcal{P}(X)) \to \mathcal{P}(\mathcal{P}(X))$ with $\log_{\mathrm{b}}(\mathcal{S}) = \bigcup_{a \in A} \Diamond_a[\mathrm{cl}_f^{\cup,\neg}(\mathcal{S})]$, where $\mathrm{cl}_f^{\cup,\neg}$ closes $\mathcal{S}$ under finite unions and complement (hence also finite intersections). Moreover, $\Diamond_a(S) = \{x \in X \mid \exists x' \in S : \ x \xrightarrow{a} x'\}$ for $a \in A$.

The set $\mu \log_{\mathrm{b}}$ of subsets of X is obtained by evaluating modal logic formulas consisting of constants *true*, *false* (empty conjunction/disjunction), binary conjunctions/disjunctions, negation and diamond modality, where the outermost operator is always the modality. Note that $\mu \log_{\mathrm{b}}$ is a strict subset of the usual modal logic formulas, but sufficient for expressivity.

▶ Remark 4.1. The continuity of $\log_{\mathrm{b}}$ deserves some attention. Note that the size of A (be it finite or infinite) has no effect on the continuity of $\log_{\mathrm{b}}$. Rather it follows from Lemma 2.1 and the fact the direct image of a function preserves arbitrary unions. As this argument remains unchanged in other contexts (e.g. simulation preorders and (bi)simulation metrics), we will henceforth tacitly state that our logic functions in the sequel are continuous.

We first study the closure associated to the Galois connection, which is important for showing compatibility later on, and the corresponding co-closure.

▶ **Proposition 4.2.** *The closure $\mathrm{c}_{\mathrm{b}} = \gamma_{\mathrm{b}} \circ \alpha_{\mathrm{b}}$ closes a set $\mathcal{S} \subseteq \mathcal{P}(X)$ under arbitrary boolean operations (union, intersection, complement), while the co-closure $\alpha_{\mathrm{b}} \circ \gamma_{\mathrm{b}}$ is the identity.*

The next step is to show that the logic function is indeed c_{b}-compatible, so that we can invoke Theorem 3.2. Not being compatible basically means that the closure c_{b} introduces operators that clash with logical equivalence. For the proof of Proposition 4.4 we require the fact that the transition system is finitely branching. We first need the following lemma:

▶ **Lemma 4.3.** *Let $(X, \to)$ be a finitely branching transition system and $(X_i \subseteq X)_{i \in \mathcal{I}}$ be a sequence of sets of states. Then, for $a \in A$, we have $\Diamond_a\Big(\bigcap_{i \in \mathcal{I}} X_i\Big) = \bigcap_{\mathcal{I}_0 \subseteq \mathcal{I} \text{ finite}} \Diamond_a\Big(\bigcap_{i \in \mathcal{I}_0} X_i\Big)$.*

▶ **Proposition 4.4.** *For finitely branching transition systems, $\log_{\mathrm{b}}$ is c_{b}-compatible.*

This theorem would straightforwardly generalize to the case where the set of a-successors is finite for each a in the qualitative case, but not directly in the quantitative case which we treat later. Hence, in this paper, we require the transition system to be finitely branching for branching equivalences/metrics, a requirement that is unnecessary in the trace case.

As a result we can derive the behaviour function from the logic function via the Galois connection. Not surprisingly, this behaviour function is in fact the well-known function whose greatest fixpoint (remember the contravariance) is bisimilarity.

▶ **Proposition 4.5.** *The behaviour function* $\mathrm{beh_b}$ *can be characterized as:* $x_1\ \mathrm{beh_b}(R)\ x_2$ *iff*

$$\forall a \in A, y_1 \in \delta_a(x_1)\ \exists y_2 \in \delta_a(x_2)\colon y_1\ R\ y_2 \wedge \forall a \in A, y_2 \in \delta_a(x_2)\ \exists y_1 \in \delta_a(x_1)\colon y_1\ R\ y_2.$$

In particular this means that $(x_1, x_2) \in \alpha_\mathrm{b}(\mu \log_\mathrm{b}) = \mu\ \mathrm{beh_b}$ *iff* x_1, x_2 *are bisimilar.*

It is well known that the behaviour function $\mathrm{beh_b}$ for bisimilarity is continuous if the underlying transition system is finitely branching.

4.2 Simulation Preorders

In this section we show that not only equivalences, but also behavioural preorders can be integrated into our framework. Our logical and behavioural universes are given by the lattices $\mathbb{L} = (\mathcal{P}(\mathcal{P}(X)), \subseteq)$ and $\mathbb{B} = (Pre(X), \supseteq)$. The Galois connection is given as follows:

$$\begin{aligned} \alpha_\mathrm{s}(\mathcal{S}) &= \{(x_1, x_2) \mid \forall S \in \mathcal{S}\colon (x_1 \in S \Rightarrow x_2 \in S)\} \\ \gamma_\mathrm{s}(R) &= \{S \subseteq X \mid \forall s \in S\colon R[\{s\}] \subseteq S\}. \end{aligned}$$

In other words, $\alpha_\mathrm{s}(\mathcal{S})[x] = \bigcap\{S \in \mathcal{S} \mid x \in S\}$. As logic function we consider $\log_\mathrm{s}\colon \mathcal{P}(\mathcal{P}(X)) \to \mathcal{P}(\mathcal{P}(X))$ with $\log_\mathrm{s}(\mathcal{S}) = \bigcup_{a\in A} \Diamond_a[\mathrm{cl}_f^\cap(\mathcal{S})]$, where $\mathrm{cl}_f^\cap$ closes a family of sets $\mathcal{S}$ under finite intersections. Hence the corresponding logic may use $\Diamond_a$ $(a \in A)$, conjunction and *true* (the empty intersection), where we again consider only formulas where the outermost operator is a modality. The logic function $\log_\mathrm{s}$ is continuous and $\mu \log_\mathrm{s}$ contains all sets that are obtained from evaluating such formulas.

As desired, the closure induced by the Galois connection closes under union and intersection, but *not* under negation, an operation that should be disallowed in a logic characterizing simulation. The co-closure is instead the identity on preorders, as in Section 4.1.

▶ **Proposition 4.6.** *The closure* $\mathrm{c_s} = \gamma_\mathrm{s} \circ \alpha_\mathrm{s}$ *closes a family of subsets of* X *under arbitrary unions and intersections. Moreover, the co-closure* $\alpha_\mathrm{s} \circ \gamma_\mathrm{s}$ *is the identity on* $Pre(X)$*.*

We show that $\log_\mathrm{s}$ is $\mathrm{c_s}$-compatible and subsequently state the main result of this section.

▶ **Proposition 4.7.** *For finitely branching transition systems,* $\log_\mathrm{s}$ *is* $\mathrm{c_s}$*-compatible.*

▶ **Theorem 4.8.** *The behaviour function* $\mathrm{beh_s}$ *can be characterized as follows:* $x_1\ \mathrm{beh_s}(R)\ x_2$ *iff* $\forall a \in A, y_1 \in \delta_a(x_1)\ \exists y_2 \in \delta_a(x_2)\colon y_1\ R\ y_2$*, i.e.,* $(x_1, x_2) \in \alpha_\mathrm{s}(\mu \log_\mathrm{s}) = \mu\ \mathrm{beh_s}$ *iff* x_2 *simulates* x_1*. Moreover, for finitely branching transition systems,* $\mathrm{beh_s}$ *is continuous.*

4.3 Trace Equivalence

We now follow the same storyline to set up a Galois connection and framework for trace equivalence, which will later be enriched to decorated traces like complete/failure/ready traces [25]. Note that we cannot use the Galois connections from the previous sections, since in particular c-compatibility would fail, due to the fact that negation and conjunction have to be disallowed in a logic using the diamond modality to characterize trace equivalence, while instead disjunction is permitted. On the logic side we use the same lattice $\mathbb{L} = (\mathcal{P}(\mathcal{P}(X)), \subseteq)$, however, the behaviour lattice $\mathbb{B} = (Eq(\mathcal{P}(X)), \supseteq)$ is the set of all equivalences over $\mathcal{P}(X)$ (instead of equivalences over X). Choosing powerset as a semantic domain seems natural due to determinization. The corresponding Galois connection is given as follows:

$$\alpha_{\mathrm{t}}(\mathcal{S}) = \{(X_1, X_2) \in \mathcal{P}(X) \times \mathcal{P}(X) \mid \forall S \in \mathcal{S}\colon (X_1 \cap S \neq \emptyset \iff X_2 \cap S \neq \emptyset)\}$$
$$\gamma_{\mathrm{t}}(R) = \{S \subseteq X \mid \forall (X_1, X_2) \in R\colon (X_1 \cap S \neq \emptyset \iff X_2 \cap S \neq \emptyset)\}.$$

Now we consider $\log_{\mathrm{t}}\colon \mathcal{P}(\mathcal{P}(X)) \to \mathcal{P}(\mathcal{P}(X))$ with $\log_{\mathrm{t}}(\mathcal{S}) = \bigcup_{a \in A} \Diamond_a[\mathcal{S}] \cup \{X\}$, which is again continuous. Then $\mu \log_{\mathrm{t}}$ represents a set of subsets of X obtained by evaluating modal logic formulas consisting of the constant *true* (which evaluates to $\{X\}$) and iterated application of the diamond modalities.

▶ **Proposition 4.9.** *The closure $\mathrm{c}_{\mathrm{t}} = \gamma_{\mathrm{t}} \circ \alpha_{\mathrm{t}}$ closes a set of subsets of X under arbitrary unions, while the co-closure $\alpha_{\mathrm{t}} \circ \gamma_{\mathrm{t}}$ maps an equivalence on $\mathcal{P}(X)$ to its congruence closure.*

As indicated in the general "recipe", the next step is to show that the logic function is compatible with the closure. Intuitively this is true since diamond distributes over union.

▶ **Proposition 4.10.** *The logic function $\log_{\mathrm{t}}$ is c_{t}-compatible.*

Finally the induced behaviour function is the one expected for trace equivalence: the bisimilarity function on the determinized transition system. This is true only for congruences, since $\mathrm{beh}_{\mathrm{t}}$ automatically returns a congruence.

▶ **Proposition 4.11.** *On a congruence relation $R \subseteq \mathcal{P}(X) \times \mathcal{P}(X)$, we have $X_1\ \mathrm{beh}_{\mathrm{t}}(R)\ X_2$ iff $(X_1 = \emptyset \iff X_2 = \emptyset) \wedge \forall a \in A\colon \delta_a[X_1]\ R\ \delta_a[X_2]$. The restriction of $\mathrm{beh}_{\mathrm{t}}$ to congruences is continuous, independent of the branching type of the transition system.*

Since on congruences $\mathrm{beh}_{\mathrm{t}}$ agrees with the usual fixpoint function for trace equivalence and $\mathrm{beh}_{\mathrm{t}}$ preserves congruences, in the corresponding Kleene iteration we obtain only congruences and hence it agrees with the usual one, where one computes bisimilarity on the determinized transition system. Hence it is easy to see that $\mu\,\mathrm{beh}_{\mathrm{t}}$ is indeed trace equivalence (cf. Theorem 4.12).

Decorated Trace Equivalences

We now consider completed trace/ready/failure/possible futures equivalence from the van Glabbeek spectrum [25] and explain how these equivalences can be obtained by adding fixed predicates. We parameterize over a family $\mathcal{S}$ of predicates over the state space (see Figure 1). We first characterize the fixpoint of the behaviour function, modified with an extra preorder as follows. The advantage of this characterisation is that it allows to state various decorated trace equivalences in terms of transfer properties as in the definition of bisimulation relations.

▶ **Theorem 4.12.** *Let $R_0 \in Pre(X)$ and consider the map $\mathrm{beh}_{R_0} = \mathrm{beh}_{\mathrm{t}} \cap (R_0)_H$. Then $\mu\ \mathrm{beh}_{R_0}$ is equal to the set $\Omega(R_0)$ of those pairs (X_1, X_2), such that if $x_1 \in X_1$ admits a trace $x_1 \xrightarrow{\sigma} x_1'$, then there exists $x_2 \in X_2$, such that $x_2 \xrightarrow{\sigma} x_2'$ and $x_1'\ R_0\ x_2'$ (and vice versa).*

In order to infer that $\mu\,\mathrm{beh}_{\mathrm{t}}$ characterizes trace equivalence simply set $R_0 = X \times X$.

The idea is to fix a set $\mathcal{S}$ of predicates and add these to our trace logic, using $R_0 = \alpha_{\mathrm{s}}(\mathcal{S})$ as the preorder required in the above theorem. In order to ensure that logical and behavioural equivalence coincide, we require that $\mathcal{S}$ has certain "good" properties.

▶ **Lemma 4.13.** *Let $\mathcal{S} \subseteq \mathcal{P}(X)$ such that $\forall x \exists S \in \mathcal{S}\colon (x \in S \wedge \forall y\colon x\ \alpha_{\mathrm{s}}(\mathcal{S})\ y \iff y \in S)$. Then, $\alpha_{\mathrm{t}}(\mathcal{S})$ coincides with the relation lifting $(\alpha_{\mathrm{s}}(\mathcal{S}))_H$.*

$\mathcal{S}$	$x\ R_0\ y$	Behavioural equivalence
$\{T_X\}$	$\mathrm{lab}(\delta(x)) = \emptyset \implies \mathrm{lab}(\delta(y)) = \emptyset$	completed trace
$\{\mathrm{Ref}(B) \mid B \subseteq A\}$	$\mathrm{lab}(\delta(y)) \subseteq \mathrm{lab}(\delta(x))$	failure
$\{\mathrm{Ready}(B) \mid B \subseteq A\}$	$\mathrm{lab}(\delta(x)) = \mathrm{lab}(\delta(y))$	ready
$\mathrm{cl}^{\cap}(\mu\ \log_t \cup \neg(\mu\ \log_t))$	$\mathrm{Tr}(x) = \mathrm{Tr}(y)$	possible futures

Figure 1 Behavioural equivalences obtained from a logic of the form $\log_0(\mathcal{F}) = \log_t(\mathcal{F}) \cup \mathcal{S}$, respectively a behaviour function of the form $\mathrm{beh}_0 = \mathrm{beh}_t \cap (R_0)_H$.

The condition of Lemma 4.13 is for instance satisfied if $\mathcal{S}$ is closed under intersections. We obtain the following characterization of decorated trace logics.

▶ **Corollary 4.14.** *Assume that $\mathcal{S}$ satisfies the requirements of Lemma 4.13 and let $R_0 = \alpha_s(\mathcal{S})$. Consider the logic function $\log_{\mathcal{S}} = \log_t \cup \mathcal{S}$. Then $\alpha_t(\mu\ \log_{\mathcal{S}}) = \Omega(R_0) = \mu\ \mathrm{beh}_{R_0}$.*

Hence if we instantiate $\mathcal{S}$ as in Figure 1, where

$$T_X = \delta^{-1}(\emptyset) \qquad \mathrm{Ref}(B) = \{x \mid \mathrm{lab}(\delta(x)) \cap B = \emptyset\} \qquad \mathrm{Ready}(B) = \{x \mid \mathrm{lab}(\delta(x)) = B\},$$

we obtain complete trace/failure/readiness equivalences as the least fixpoint of beh_{R_0}. In all these cases $\mathcal{S} \cup \{X\}$ satisfies the requirements of Lemma 4.13.

Note that $\{X\}$ is already generated by $\log_t$. The predicate T_X semantically corresponds to the predicate denoted 0 in [25] (satisfied by those states that have no outgoing transitions). Similarly, the predicate $\mathrm{Ref}(B)$ (resp. $\mathrm{Ready}(B)$) corresponds to the predicate $\tilde{B}$ (resp. B) in [25], which is satisfied by those states that refuse (resp. enable) all the actions from B.

5 Quantitative Case

After discussing the classical case of behavioural equivalences, we will now follow an analogous path to obtain behavioural distances in a quantitative setting. We will begin by first considering the bisimulation pseudo-metric, then directed simulation pseudo-metric, and lastly conclude with the directed (decorated) trace pseudo-metric, from which one can obtain the undirected version by symmetrization. In each case, we will again start out by defining the logics and derive the fixpoint equations for the corresponding behaviour function.

In addition, our decorated trace distance can be seen as the quantitative generalization of a decorated trace preorder, which when instantiated corresponds to (complete) trace/failure/ready inclusions. So, in this sense, our decorated trace distance is going to be parametric. Lastly, though the concrete trace distance is studied elsewhere (cf. [6, 8]), we are not aware of this fixpoint characterization of (decorated) trace distance in the literature. There is a recursive characterization in [8], but based on an auxiliary lattice that serves as memory.

In the rest of this section we fix a metric transition system $(X, \to, d_A)$ over A.

5.1 Bisimulation Pseudo-metrics

Recall the adjunction from Section 4.1, which we will enrich by replacing a predicate $S \subseteq X$ with a function $f\colon X \to [0,1]$, while pseudo-metrics now play the role of equivalences. In particular, our logical and behavioural universes are given by the lattices $\mathbb{L} = (\mathcal{P}([0,1]^X), \subseteq)$ and $\mathbb{B} = (PMet(X), \leq)$, respectively. Moreover, the Galois connection is given as follows:

$$\alpha_{\mathrm{B}}(\mathcal{F})(x_1,x_2) = \bigvee_{f\in\mathcal{F}} |f(x_1)-f(x_2)| \qquad (\text{for } \mathcal{F}\subseteq[0,1]^X)$$
$$\gamma_{\mathrm{B}}(d) = \{f\in[0,1]^X \mid \forall x_1,x_2\in X : |f(x_1)-f(x_2)|\le d(x_1,x_2)\} \quad (\text{for } d\in PMet(X)).$$

That is, $\alpha_{\mathrm{B}}(\mathcal{F})$ is the least metric on X such that all functions in $\mathcal{F}$ are non-expansive wrt. the Euclidean metric on $[0,1]$, while γ_{B} returns all the non-expansive functions wrt. $d\in PMet(X)$.

Next we introduce a family of modalities $(\bigcirc_a f)_{a\in A}$ in the style of [6]:

$$\bigcirc_a f(x) = \bigvee\{\overline{D_a}(b)\wedge f(x') \mid x \xrightarrow{b} x'\}, \quad \text{where } D_a(b)=d_A(b,a) \text{ and } \overline{D_a}(b)=1-D_a(b).$$

We consider the (continuous) logic function $\log_{\mathrm{B}}\colon \mathcal{P}([0,1]^X)\to\mathcal{P}([0,1]^X)$ that maps a set $\mathcal{F}\subseteq[0,1]^X$ of functions to the set $\bigcup_{a\in A}\bigcirc_a[\mathrm{cl}_f^{\wedge,\neg,\mathrm{sh}}(\mathcal{F})]$, where $\mathrm{cl}_f^{\wedge,\neg,\mathrm{sh}}$ closes $\mathcal{F}$ under finite meets, complements ($f\mapsto 1-f$), and constant shifts (and hence also under finite joins), which are all non-expansive operators (cf. Proposition 5.3). It should be noted that $\bigcirc_a$ is a quantitative generalization of the qualitative diamond modality in the following sense.

▶ **Proposition 5.1.** *If d_A is a discrete metric then $\bigcirc_a f(x)=1 \iff x\in\Diamond_a f^{-1}(\{1\})$.*

Following the development of Section 4.1, we establish the metric version of Lemma 4.3:

▶ **Lemma 5.2.** *Let $(X,\to,d_A)$ be a finitely branching metric transition system and $\mathcal{F}\subseteq[0,1]^X$ be a family of functions. Then for $c\in A$ we have $\bigcirc_c\left(\bigwedge_{f\in\mathcal{F}} f\right) = \bigwedge_{\mathcal{F}_0\subseteq\mathcal{F} \text{ finite}} \bigcirc_c\left(\bigwedge_{f\in\mathcal{F}_0} f\right)$.*

In the quantitative case, the closures induced by the Galois connections had appealing characterizations via boolean operators. Here the closure c_{B} is obtained by post-composing the functions in $\mathcal{F}$ with *all* non-expansive operators. This is in fact a corollary of the McShane-Whitney extension theorem [20, 26].

▶ **Proposition 5.3.** *The closure $\mathrm{c}_{\mathrm{B}}=\gamma_{\mathrm{B}}\circ\alpha_{\mathrm{B}}$ on $\mathcal{F}\subseteq[0,1]^X$ can be characterized as follows:*

$$\mathrm{c}_{\mathrm{B}}(\mathcal{F}) = \{\mathrm{op}\circ\langle\mathcal{F}\rangle \mid \mathrm{op}\colon[0,1]^{\mathcal{F}}\to[0,1] \text{ is non-expansive wrt. the sup-metric}\}.$$

Moreover, the co-closure $\alpha_{\mathrm{B}}\circ\gamma_{\mathrm{B}}$ is the identity.

The proof of the above proposition and the next two results are analogous to the corresponding results in the next section on simulation.

▶ **Proposition 5.4.** *For finitely branching transition systems, $\log_{\mathrm{B}}$ is c_{B}-compatible.*

▶ **Theorem 5.5.** *The behaviour function $\mathrm{beh}_{\mathrm{B}}$ on any $d\in PMet(X)$ is $\mathrm{beh}_{\mathrm{B}}(d)=(d_A\otimes d)_H\circ(\delta\times\delta)$, which results exactly in bisimulation metrics as considered in [6]. Moreover, $\mathrm{beh}_{\mathrm{B}}$ is continuous for finitely branching transition systems.*

It is well-known that the kernel of the bisimulation metric, i.e., the pairs of states with distance 0, is exactly bisimilarity [8].

5.2 Directed Simulation Metrics

In this section, we will treat simulation distance. Our logical and behavioural universes are $\mathbb{L}=(\mathcal{P}([0,1]^X),\subseteq)$ and $\mathbb{B}=(DPMet(X),\le)$ with

$$\alpha_{\mathrm{S}}(\mathcal{F})(x_1,x_2) = \bigvee_{f\in\mathcal{F}} (f(x_1)\ominus f(x_2)) \qquad (\text{for } \mathcal{F}\subseteq[0,1]^X)$$
$$\gamma_{\mathrm{S}}(d) = \{f\in[0,1]^X \mid \forall x_1,x_2\in X : f(x_1)\ominus f(x_2)\le d(x_1,x_2)\} \quad (\text{for } d\in DPMet(X)).$$

Now our (continuous) logic function $\log_S \colon \mathcal{P}([0,1]^X) \to \mathcal{P}([0,1]^X)$ is the mapping $\mathcal{F} \mapsto \bigcup_{c \in A} \bigcirc_c [\mathrm{cl}_f^{\wedge,\mathrm{sh}}(\mathcal{F})]$, where $\mathrm{cl}_f^{\wedge,\mathrm{sh}}$ closes $\mathcal{F}$ under finite meets and constant shifts (whose necessity is argued in Example 5.9). To characterize the closure $\mathrm{c_S}$ we use a directed version of the McShane-Whitney extension theorem [20, 26] (a special case of enriched Kan extensions).

▶ **Proposition 5.6.** *The closure $\mathrm{c_S} = \gamma_S \circ \alpha_S$ on $\mathcal{F}$ is the set given in Proposition 5.3 except that* op *is non-expansive wrt. the directed sup-metric. The co-closure $\alpha_S \circ \gamma_S$ is the identity.*

In order to show $\mathrm{c_S}$-compatibility of $\log_S$, we first derive an alternative characterization of the closure in terms of certain normal form given below. Note that a similar statement holds in the context of bisimulation pseudo-metric when we replace the closure $\mathrm{c_S}$ by $\mathrm{c_B}$.

▶ **Proposition 5.7** (Normal Form). *Let $\mathcal{F} \subseteq [0,1]^X$ with $1 \in \mathcal{F}$ and $f \in \mathrm{c_S}(\mathcal{F})$. Then there is a family of functions $\{f^{\varepsilon}_{(x,y)} \mid \varepsilon > 0 \text{ and } x, y \in X\}$, where each function $f^{\varepsilon}_{(x,y)}$ is a constant shift of a function in $\mathcal{F}$, such that $f = \bigvee_{\varepsilon>0} \bigwedge_{x \in X} \bigvee_{y \in X} f^{\varepsilon}_{(x,y)}$.*

These results enable us to show that the logic function is indeed compatible with closure, a prerequisite for being able to derive the corresponding behaviour function.

▶ **Proposition 5.8.** *For finitely branching transition systems, $\log_S$ is $\mathrm{c_S}$-compatible.*

▶ **Example 5.9.** *We show that adding shifts to the logic function is necessary to obtain compatibility. Consider the metric transition system $(\{x, y, x_1, y_1\}, \{x \xrightarrow{1} x_1, y \xrightarrow{0} y_1\}, d_A)$ with d_A is an Euclidean metric over the alphabet $A = [0,1]$.*

Assume that $\mathcal{F} = \{f\}$ with $f(x) = f(y) = f(x_1) = 1/2$, $f(y_1) = 0$, where the pseudo-metric $d = \alpha_S(\mathcal{F})$ has distance 0 for the states x, y, x_1 and it yields distance $1/2$ between y_1 and all other states. Then it is easy to see that g with $g(x) = g(y) = g(x_1) = 1$, $g(y_1) = 1/2$ is contained in $\mathrm{c_S}(\mathcal{F})$, since it is non-expansive wrt. d. Then $\bigcirc_1 g \in \log_S(\mathrm{c_S}(\mathcal{F}))$ and

$$\bigcirc_1 g(x) \ominus \bigcirc_1 g(y) = (\overline{D_1}(1) \wedge g(x_1)) \ominus (\overline{D_1}(0) \wedge g(y_1)) = (1 \wedge 1) \ominus (0 \wedge 1/2) = 1.$$

In order for compatibility to hold, $\bigcirc_1 g$ must be contained in $\mathrm{c_S}(\log_S(\mathcal{F}))$, i.e., it has to be non-expansive wrt. $\alpha_S(\log_S(\mathcal{F}))$. If the logic function does not use shifts, it only closes $\mathcal{F}$ under finite meets and joins, which results in f, 0 (empty join), 1 (empty meet). For all $r \in [0,1], \bar{f} \in \mathrm{c_S}(\mathcal{F})$, we have $\bigcirc_r \bar{f}(x) \ominus \bigcirc_r \bar{f}(y) < \bigcirc_1 g(x) \ominus \bigcirc_1 g(y)$, which means $\bigcirc_1 g \notin \mathrm{c_S}(\log_S(\mathcal{F}))$. In particular,

$$\bigcirc_r f(x) \ominus \bigcirc_r f(y) = (\overline{D_r}(1) \wedge f(x_1)) \ominus (\overline{D_r}(0) \wedge f(y_1)) = (\overline{D_r}(1) \wedge 1/2) \ominus (\overline{D_r}(0) \wedge 0) \leq 1/2.$$

▶ **Theorem 5.10.** *The behaviour function $\mathrm{beh_S}$ can be characterized as $\mathrm{beh_S}(d) = (d_A \otimes d)^{\rightarrow}_H \circ (\delta \times \delta)$ for any $d \in DPMet(X)$. In particular, $\alpha_S(\mu \log_S) = \mu\ \mathrm{beh_S}$ is the directed similarity metric of [6]. Moreover, $\mathrm{beh_S}$ is continuous for finitely branching transition systems.*

Every metric transition system can be viewed as a classical one by forgetting the metric on the labels. In addition, we can first compute the simulation metric of the quantitative system and then discretize the values to obtain qualitative simulation equivalence.

▶ **Proposition 5.11.** *Consider the Galois connection $\alpha \dashv \gamma$ with $\alpha \colon DPMet(X) \to Pre(X)$ and $\gamma \colon Pre(X) \to DPMet(X)$ given by the maps $\alpha(d) = \{(x,y) \mid d(x,y) = 0\}$, $\gamma(R) = 1 - \chi_R$. If the transition system is finitely branching, then $\alpha \circ \mathrm{beh_S}(d) = \mathrm{beh_s} \circ \alpha(d)$ for every $d \in DPMet(X)$. In particular $\mu\ \mathrm{beh_s} = \alpha(\mu\ \mathrm{beh_S})$ due to Theorem 3.2.*

We conclude this section by the observation that the characterization in Theorem 5.10 allows us to eliminate constant shifts from the simulation logic.

▶ **Corollary 5.12.** *Let* $\log' \colon \mathcal{P}([0,1]^X) \to \mathcal{P}([0,1]^X)$ *be the variant of* $\log_{\mathrm{S}}$*, where we do not close under constant shifts. If the transition system is finitely branching, then* $\log'$ *is still sound and expressive for simulation, that means* $\alpha_{\mathrm{S}}(\mu \log') = \mu \,\mathrm{beh}_{\mathrm{S}}$*.*

5.3 Directed Trace Metrics

In this section, we treat the directed version of the (decorated) trace distance whose fixpoint characterization is novel and, at the same time, the most complex scenario considered in this paper. We will sometimes omit the adjective "directed".

Based on the qualitative case of trace equivalence (Section 4.3), we fix the logical and behavioural universes to be the lattices $\mathbb{L} = (\mathcal{P}([0,1]^X), \subseteq)$ and $\mathbb{B} = (DPMet(\mathcal{P}(X)), \leq)$ with

$$\alpha_{\mathrm{T}}(\mathcal{F})(X_1, X_2) = \bigvee_{f \in \mathcal{F}} (\tilde{f}(X_1) \ominus \tilde{f}(X_2)) \qquad (\text{for } \mathcal{F} \subseteq [0,1]^X)$$

$$\gamma_{\mathrm{T}}(d) = \{f \in [0,1]^X \mid \tilde{f} \text{ is non-expansive wrt. } d\} \qquad (\text{for } d \in DPMet(\mathcal{P}X)).$$

It is easy to see that $\alpha_{\mathrm{T}}(\mathcal{F})$ is always join-preserving in its first argument. Notice that we could have defined $\mathbb{L}$ as those functions in $[0,1]^{\mathcal{P}(X)}$ that are join-preserving. As a result, one expects that the closure c_{T} may close a set $\mathcal{F}$ under all non-expansive and join-preserving operators. However, this is unfortunately not true as witnessed by the following counterexample. This points to the more fundamental problem that there is no McShane-Whitney type result for non-expansive, join-preserving operators: a non-expansive, join-preserving operator defined on some subset does not necessarily have an extension to the whole space which is both non-expansive and join-preserving.

▶ **Example 5.13.** *Let* $X = \{x, y, z\}$ *and* $\mathcal{F} = \{f_1, f_2\} \subseteq [0,1]^X$*, where* f_1 *and* f_2 *are the mappings* $x, y \mapsto 1, z \mapsto 0$ *and* $x, z \mapsto 0, y \mapsto 1/2$*, respectively. Now consider a map* $g \colon X \to [0,1]$ *with* $g(x) = 1/2$*,* $g(y) = 1$ *and* $g(z) = 0$*. Then it is easily seen that* $g \in \mathrm{c}_{\mathrm{T}}(\mathcal{F})$*. However, we claim that there is no join-preserving and non-expansive operator* $\mathrm{op} \colon [0,1]^2 \to [0,1]$ *such that* $g = \mathrm{op} \circ \langle f_1, f_2 \rangle$*. Assume otherwise that* $\mathrm{op}(f_1(u), f_2(u)) = g(x)$ *(for* $u \in X$*), which implies* $\mathrm{op}(1,0) = 1/2$*,* $\mathrm{op}(1, 1/2) = 1$*, and* $\mathrm{op}(0,0) = 0$*. Due to non-expansivity of* op *we conclude that* $\mathrm{op}(0, 1/2) \leq 1/2$*, which leads to the following contradiction:*

$$1 = \mathrm{op}(1, 1/2) = \mathrm{op}((1,0) \vee (0, 1/2)) = \mathrm{op}(1,0) \vee \mathrm{op}(0, 1/2) = 1/2.$$

As (continuous) logic function $\log_{\mathrm{T}} \colon \mathcal{P}([0,1]^X) \to \mathcal{P}([0,1]^X)$ we define $\log_{\mathrm{T}}(\mathcal{F}) = \bigcup_{a \in A} \bigcirc_a [\mathrm{cl}^{\mathrm{sh}}(\mathcal{F})] \cup \{1\}$, where $\mathrm{cl}^{\mathrm{sh}}$ closes a set of functions under constant shifts, as in Section 5.2, and 1 is the constant 1-function. Typically, operators of a "metric" logic ought to preserve non-expansiveness, which is not the case for the shift $f \mapsto f \oplus r$; since it might increase the distance of a non-empty set to the empty set. This is not problematic in our case, since the distance of $\emptyset$ to any other set is 1 anyway, induced by the constant operator 1. (Note that the empty join is 0.) We will show in Theorem 5.20 that our logic characterizes trace distance, i.e., $\alpha_{\mathrm{T}}(\mu \log_{\mathrm{T}}) = d_T$, where $d_T := (d_{\mathrm{Tr}})_{\overrightarrow{H}} \circ (\mathrm{Tr} \times \mathrm{Tr})$.

The co-closure, on the other hand, is straightforward to characterize.

▶ **Proposition 5.14.** *The co-closure* $\alpha_{\mathrm{T}} \circ \gamma_{\mathrm{T}}$ *maps a directed pseudo-metric* d *to the greatest directed pseudo-metric* d' *such that* $d' \leq d$ *and* d' *is join-preserving in its first argument.*

Next we turn our attention to the compatibility of our logic function. Here we have to work around the fact that the closure can not be easily characterized, neither in terms of operators nor in terms of a suitable normal form (cf. Proposition 5.7). Still, compatibility holds, even for transition systems of arbitrary branching type.

▶ **Lemma 5.15.** *Let $h\colon X \to [0,1]$, $c \in A$ and $Y \subseteq X$. Then it holds that*

$$\widetilde{\bigcirc_c} h(Y) = \bigwedge_{\Delta \subseteq \delta[Y]} \left(\widetilde{D_c}(\mathrm{lab}(\delta[Y] \setminus \Delta)) \vee \tilde{h}(\mathrm{tgt}(\Delta))\right)$$

▶ **Proposition 5.16.** *The logic function $\log_{\mathrm{T}}$ is c_{T}-compatible.*

Now we can characterize the behaviour function as follows. To the best of our knowledge, this is the first time that a fixpoint function for trace metrics on the powerset of the state space has been established. There is also a fixpoint characterization given in [8] although on an auxiliary lattice which serves as a memory.

▶ **Theorem 5.17.** *Let $d \in DPMet(\mathcal{P}(X))$ such that d is join preserving in its first argument and $d(X_1, \emptyset) = 1$ for every non-empty set $X_1 \subseteq X$. Then the behaviour function $\mathrm{beh}_{\mathrm{T}}$ can be characterized by the conditional equation: $\mathrm{beh}_{\mathrm{T}}(d)(X_1, \emptyset) = 1$ if $X_1 \neq \emptyset$ and otherwise*

$$\mathrm{beh}_{\mathrm{T}}(d)(X_1, X_2) = \bigvee_{(a,x') \in \delta[X_1]} \bigvee_{\Delta \subseteq \delta[X_2]} \Big(\bigwedge_{b \in \mathrm{lab}(\Delta)} d_A(a,b) \wedge d(\{x'\}, \mathrm{tgt}(\delta[X_2] \setminus \Delta)) \Big).$$

Moreover $\mathrm{beh}_{\mathrm{T}}$ is continuous, independent of the branching type of the transition system.

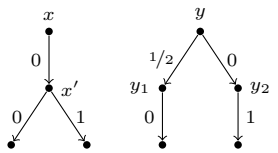
The special case of $X_1 \neq \emptyset$, $X_2 = \emptyset$ is an effect of the term $\{1\}$ in the logic function $\log_{\mathrm{T}}$.

Note that to our knowledge there is no straightforward way to compute the bisimilarity distance on the determinization (see Theorem 17 in [6]). Next, we explain the above fixpoint equation as a two-player game.

▶ Remark 5.18. Given two sets $X_1, X_2 \subseteq X$ and a threshold $\varepsilon \in [0,1]$, we want to check, whether $d_{\mathrm{T}}(X_1, X_2) \leq \varepsilon$ with a game played by two players: Death D and Maiden M. First, D chooses a transition (a, x') of $\delta[X_1]$ and also stipulates a set $\Delta \subseteq \delta[X_2]$ of allowed transitions for M. Now M has two possibilities: she can either accept the set Δ presented by D, or she can reject it. If she rejects it, she can only reach states in $Y' := \mathrm{tgt}(\delta[X_2] \setminus \Delta)$ and whatever way D chooses to continue his trace from the state x', M must continue her trace from one of the states in Y'. The game therefore continues with the sets $\{x'\}$ and Y'. If, on the other hand, M chooses to accept the set presented by D, then, in trying to duplicate the trace begun by D with one of the transitions in Δ, she makes an error of at least $\bigwedge_{(b,y') \in \Delta} d_A(a,b)$. It is clear that M should only make this decision if she thinks that D can otherwise force a larger error in a later stage of the game. The game therefore ends and M wins iff $\bigwedge_{(b,y') \in \Delta} d_A(a,b) \leq \varepsilon$.

▶ **Example 5.19.** *We compute the directed trace distance of the state x to the state y in the metric transition system over $A = [0,1]$ depicted in Figure 2 with $\mathrm{Tr}(x) = \{(0,0),(0,1)\}$, $\mathrm{Tr}(y) = \{(1/2,0),(0,1)\}$. There is only one outgoing transition from x and there are four possible choices for $\Delta \subseteq \delta(y)$. The corresponding terms are calculated in Figure 2, where we used that we already computed $d_{\mathrm{T}}(\{x'\},\{y_1\}) = d_{\mathrm{T}}(\{x'\},\{y_2\}) = d_{\mathrm{T}}(\{x'\},\emptyset) = 1$ and $d_{\mathrm{T}}(\{x'\},\{y_1,y_2\}) = 0$. Taking the maximum of the minima we see that $d_{\mathrm{T}}(\{x\},\{y\}) = 1/2$, which is indeed the Hausdorff distance between the two trace sets.*

In the case of the trace metric we can eliminate constant shifts from the logic without losing expressiveness. This is a consequence of Corollary 5.22, which we will prove later.



Δ	$\bigwedge_{(b,y')\in\Delta} d_A(a,b)$	$d_{\mathrm{T}}(\{x'\}, \mathrm{tgt}(\delta(y)\setminus\Delta))$
$\emptyset$	1	0
$\{(1/2, y_1)\}$	1/2	1
$\{(0, y_2)\}$	0	1
$\delta(y)$	0	1

Figure 2 Computation of trace distance.

Decorated Trace Distances

Now we consider the quantitative generalization of decorated trace preorders. We follow a presentation similar to Theorem 4.12, wherein we characterize the least fixpoint of a behaviour function parameterized by a distance $d_0 \in \mathit{DPMet}(X)$, which is in turn induced by a set $\mathcal{G} \subseteq [0,1]^X$, corresponding to completed/failure/readiness traces.

▶ **Theorem 5.20.** *Let $d_0 \in \mathit{DPMet}(X)$ and consider the map $\mathrm{beh}_{d_0} \colon \mathit{DPMet}(\mathcal{P}(X)) \to \mathit{DPMet}(\mathcal{P}(X))$ defined as $\mathrm{beh}_{d_0}(d) = \mathrm{beh}_{\mathrm{T}}(d) \vee (d_0)^{\rightarrow}_{H}$, for any $d \in \mathit{DPMet}(\mathcal{P}(X))$. Then $\mu\,\mathrm{beh}_{d_0}(X_1, X_2)$ is characterized as the infimum of those $\varepsilon \in [0,1]$ that satisfy:*

$$\forall x_1 \in X_1, x_1' \in X, \sigma \in A^* \colon x_1 \xrightarrow{\sigma} x_1' \\ \implies \exists x_2 \in X_2, x_2' \in X, \tau \in A^* \colon x_2 \xrightarrow{\tau} x_2' \wedge d_{\mathrm{Tr}}(\sigma,\tau) \le \varepsilon \wedge d_0(x_1', x_2') \le \varepsilon.$$

When d_0 is the constant 0-metric, this results in the behaviour function $\mathrm{beh}_{\mathrm{T}}$ that characterizes trace distance. Next, we reformulate the result in terms of a set $\mathcal{G} \subseteq [0,1]^X$; this in turn helps in deriving the characterization of various decorated trace distances. We start by imposing a condition on such a set $\mathcal{G}$ that guarantees that $\alpha_{\mathrm{T}}(\mathcal{G})$ is the directed Hausdorff lifting of $d_0 = \alpha_{\mathrm{S}}(\mathcal{G})$ (cf. Section 5.2), which ensures that by Proposition 3.4 the enriched logic function induces a behaviour function as in the previous theorem.

▶ **Lemma 5.21.** *Let $\mathcal{G} \subseteq [0,1]^X$ such that $d_0 = \alpha_{\mathrm{S}}(\mathcal{G})$. Then $\alpha_{\mathrm{T}}(\mathcal{G}) = (d_0)^{\rightarrow}_{H}$ whenever*

$$\forall \varepsilon > 0, x \in X\ \exists g \in \mathcal{G} :\ g(x) = 1 \wedge \forall x' :\ g(x) \ominus g(x') \ge d_0(x,x') - \varepsilon.$$

▶ **Corollary 5.22.** *Assume that $\mathcal{G} \subseteq [0,1]^X$ satisfies the requirements of Lemma 5.21. Then $\alpha_{\mathrm{T}}(\mu(\log_{\mathrm{T}} \cup \mathcal{G})) = (d_{\mathrm{Tr}} \otimes d_0)^{\rightarrow}_{H} \circ (\hat{\delta} \times \hat{\delta})$. The same holds if the logic function $\log_{\mathrm{T}}$ is replaced by $\log'$ with $\log'(\mathcal{F}) = \bigcup_{a\in A} \bigcirc_a[\mathcal{F}] \cup \{1\}$ (without shifts).*

$\mathcal{G}$	$d_0(x,y)$	Behavioural distance
$\{f_A^{\mathrm{Ref}}\}$	$f_A^{\mathrm{Ref}}(x) \ominus f_A^{\mathrm{Ref}}(y)$	completed trace
$\{f_B^{\mathrm{Ref}}, \mid B \subseteq A\}$	$(d_{\mathrm{disc}})^{\rightarrow}_{H}(\mathrm{lab}(\delta(y)), \mathrm{lab}(\delta(x)))$	(discrete) failures
$\mathrm{cl}^{\wedge,\mathrm{sh}}(\{g_a \mid a \in A\})$	$(d_A)^{\rightarrow}_{H}(\mathrm{lab}(\delta(y)), \mathrm{lab}(\delta(x)))$	(Hausdorff) failures
$\{f_B^{\mathrm{Ready}} \mid B \subseteq A\}$	$d_{\mathrm{disc}}(\mathrm{lab}(\delta(x)), \mathrm{lab}(\delta(y)))$	(discrete) readiness
$\mathrm{cl}^{\wedge,\mathrm{sh}}(\{g_a, 1-g_a \mid a \in A\})$	$(d_A)_H(\mathrm{lab}(\delta(x)), \mathrm{lab}(\delta(y)))$	(Hausdorff) readiness
$\mathrm{cl}^{\wedge,\mathrm{sh}}(\mu\ \log_{\mathrm{T}} \cup \neg(\mu\ \log_{\mathrm{T}}))$	$\overline{d_{\mathrm{T}}}(\{x\}, \{y\})$	possible futures

Figure 3 Behavioural distances obtained from a logic of the form $\log_0(\mathcal{F}) = \log_{\mathrm{T}}(\mathcal{F}) \cup \mathcal{G}$, respectively a behaviour function of the form $\mathrm{beh}_0 = \mathrm{beh}_{\mathrm{T}} \vee (d_0)^{\rightarrow}_{H}$, where $d_0 = \alpha_{\mathrm{S}}(\mathcal{G})$.

These results supply fixpoint characterizations of several meaningful behavioural distances. In Figure 3 we summarize which primitive set of functions $\mathcal{G} \subseteq [0,1]^X$ has to be added to the trace logic in order to get (directed) metric versions of some decorated trace semantics considered in [25]: completed/failure/ready trace semantics.

▶ **Corollary 5.23.** *Consider the following functions* $f_B^{\mathrm{Ref}}, f_B^{\mathrm{Ready}}, g_a \in [0,1]^X$*:*

$$f_B^{\mathrm{Ref}}(x) = \begin{cases} 1 & x \in \mathrm{Ref}(B) \\ 0 & \textit{otherwise} \end{cases} \quad f_B^{\mathrm{Ready}}(x) = \begin{cases} 1 & x \in \mathrm{Ready}(B) \\ 0 & \textit{otherwise} \end{cases} \quad g_a(x) = \bigwedge_{b \in \mathrm{lab}(\delta(x))} d_A(a,b).$$

Then by adding $\mathcal{G}$ *to* $\log_{\mathrm{T}}$ *results in the behaviour functions and distances as given in Figure 3.*

Note that the different versions of failures and readiness metrics correspond to different ways to measure the distance between the refuse/ready sets of two states. In the first version we take the discrete metric on $\mathcal{P}(A)$, and in the second version we take the Hausdorff lifting of d_A. In the qualitative setting, the two notions collapse. The Hausdorff versions are the ones to use if we want to recover the hierarchy of [25].

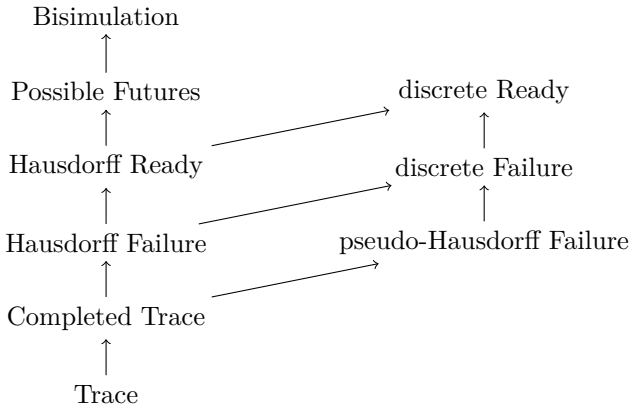


Figure 4 A spectrum of behavioural distances.

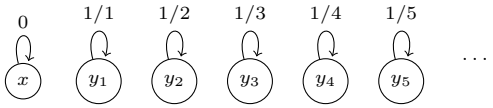
Consider also the pseudo-Hausdorff failure semantics arising from choosing a set $\mathcal{G}$ of predicates with $\alpha_S(\mathcal{G})(x,y) = d_0(x,y) = (d_A)^{\rightarrow}_{H}(A \setminus \mathrm{lab}(\delta(x)), A \setminus \mathrm{lab}(\delta(y)))$. In the qualitative setting this notion collapses with Hausdorff failure and discrete failure semantics, but in the metric setting the pseudo-Hausdorff failure distance is not even bounded by the bisimulation distance (see Figure 4). The inclusions shown in Figure 4 are obvious from comparing the corresponding metrics d_0 in Figure 3.

Again we conclude by comparing the qualitative and quantitative case.

▶ **Proposition 5.24.** *Consider the map* $\alpha\colon DPMet(\mathcal{P}(X)) \to Pre(\mathcal{P}(X))$ *given by* $\alpha(d) = \{(X_1, X_2) \mid d(X_1, X_2) = 0\}$*. If the set* A *of actions is finite, then* $\mu\ \mathrm{beh}_{\mathrm{t}} = \alpha(\overline{\mu\ \mathrm{beh}_{\mathrm{T}}})$*.*

The necessity of requiring finiteness of A is illustrated by Example 5.25.

▶ **Example 5.25.** *Consider the transition system depicted below:*



The trace distance of $X_1 = \{y_{i+1} \mid i \in \mathbb{N}\}$ *and* $X_2 = X_1 \cup \{x\}$ *is* $\overline{\mu\ \mathrm{beh}_{\mathrm{T}}}(X_1, X_2) = 0$*. However we do not have full trace inclusion, hence* $(X_1, X_2) \notin \mu\ \mathrm{beh}_{\mathrm{t}}$*.*

6 Concluding Remarks, Related and Future Work

We presented a recipe to construct (bi)simulation equivalence/distance and trace equivalence/distance (together with various forms of their decorated trace counterparts) as the least fixpoint of behaviour functions on the underlying lattice $\mathbb{B}$ of equivalences/distances. Furthermore, upon realising the relevant Galois connection $\alpha \dashv \gamma$ between the lattices $\mathbb{L}$ (modelling sets of predicates) and $\mathbb{B}$, we showed in each case that these behaviour functions arise naturally (i.e., $\mathrm{beh} = \alpha \circ \log \circ \gamma$) when the logic function log is compatible with the closure $\gamma \circ \alpha$. By doing so, we not only recover the fixpoint characterizations of the branching-time spectrum, but we also gave novel ones in the linear-time spectrum (like the trace distances and their variations: completed trace/failure/ready/possible futures).

Related work

Our work is related to [6, 8], where the former establishes a logical characterization (using the syntax of LTL and μ-calculus) of bisimulation and trace distances, while the latter recasts a part of the classical linear-branching time spectrum to a quantitative one involving metrics, based on games. The fixpoint and logical characterizations of (decorated) trace distances were not present in both [6, 8]. In [8] the authors parameterize over various trace distances, which we are not, although this is an interesting direction for future work. By restricting to pointwise trace distance with discount one, we obtain corresponding notions for bisimilarity, trace and (Hausdorff) readiness. Note that [8] does not treat failures. Also, our game in Remark 5.18 is different from the games played in [8], since it is played locally on the powerset domain.

Coalgebraists familiar with fibrations/indexed categories [10] will recognize the Galois connection between the fibres of two indexed categories: one modelling the logical universe, the other behavioural universe on the state space of a coalgebra. Indeed, Klin in his PhD thesis [13] has explored this adjoint situation $\alpha_{\mathrm{b}} \dashv \gamma_{\mathrm{b}}$ (cf. Section 4.1); note that behavioural metrics were not treated in [13]. The two approaches diverge in the treatment of closures especially in the context of decorated traces. In this paper, closures are always induced as monad from the adjoint situation and to handle (decorated) trace equivalences we consider the adjoint situation $\alpha_{\mathrm{t}} \dashv \gamma_{\mathrm{t}}$ since the closure c_{b} is not sound w.r.t. (decorated) trace equivalence. In Klin's approach, on the contrary, the adjunction $\alpha_{\mathrm{b}} \dashv \gamma_{\mathrm{b}}$ used to characterize bisimilarity is fixed (even for decorated trace equivalences), but the notion of closure is left parametric [13, Definition 3.31]. Our new insight in the qualitative case is that the closure is naturally induced by the Galois connection and the characterization of fixpoint preservation is a fundamental ingredient.

We also point out the differences to the dual adjunction approach [12, 17, 18, 22] to coalgebraic modal logic. There the functor on the "logic universe" characterizes the *syntax* of the logics, while the semantics is given by a natural transformation. In [18] the approach is lifted to fibrations (in which the equivalence lives). Generalizing our approach however would lead to a situation where we obtain a fibred adjunction between two fibrations (for logic and behaviour) on the same category.

In [14] the approach of [18] is instantiated to a quantitative setting, without treating trace metrics. A central notion there is that of an approximating family, which, translated into our language, says that $\mathcal{F} \subseteq [0,1]^X$ is an approximating family iff $\forall f \in [0,1]^X : \alpha(\mathcal{F}) \geq \alpha(\{f\})$ implies $\alpha(\log(\mathcal{F})) \geq \alpha(\log(\{f\}))$, with log being restricted to applying modalities. If log is join-preserving, this is equivalent to $\log(c(\mathcal{F})) \subseteq c(\log(\mathcal{F}))$ (this is a direct consequence of Lemma 3.5), i.e., it is strongly related to compatibility.

Future work

Taking inspiration from the above, we want to generalize our work to the level of coalgebras with an approach based on fibrations, enabling us to treat other branching types, such as probabilistic branching. Note that the coalgebraic treatment of establishing Hennessy-Milner theorems in [11, 13] does not subsume the behavioural distances covered in this paper, while the qualitative spectrum has been generalized using graded monads [21]. We plan to develop fixpoint and logical characterizations of coalgebraic behavioural metrics [1, 15], which are generalizations of both bisimulation pseudo-metric and trace distance.

We are also interested in exploring connections with [16], a paper studying the question which formulas of Hennessy-Milner logic are preserved by quotienting through a behavioural equivalence.

Another direction is to consider behavioural equivalences (such as failure trace/ready trace equivalences and variations) that cannot be captured by our modular approach (i.e., by extending the logic functions $\log_t/\log_T$ with a *constant* function). We also want to characterize undirected trace distance directly without the symmetrization of directed trace distance.

Another line of research is to determine under which circumstances we can restrict to finitary operations, from which we deviate occasionally by closing under arbitrary meets or intersections. This should be feasible by restricting to finitely branching transition systems. Also, in the metric case, we plan to optimize the syntax by restricting shifts and modalities to rational numbers. Last, but not least, it will be interesting to work out the compatibility of $\log_B$ for a weaker class of metric transition systems than those which are finitely branching.

References

1 Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König. Coalgebraic behavioral metrics. *Logical Methods in Computer Science*, 14(3), 2018. Selected Papers of the 6th Conference on Algebra and Coalgebra in Computer Science (CALCO 2015).

2 Paolo Baldan, Barbara König, and Tommaso Padoan. Abstraction, up-to techniques and games for systems of fixpoint equations. In *Proc. of CONCUR '20*, volume 171 of *LIPIcs*, pages 25:1–25:20. Schloss Dagstuhl – Leibniz Center for Informatics, 2020.

3 Harsh Beohar, Sebastian Gurke, Barbara König, and Karla Messing. Hennessy-Milner theorems via Galois connections, 2022. `arXiv:2207.05407`.

4 Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proc. of POPL '79 (San Antonio, Texas)*, pages 269–282. ACM Press, 1979.

5 Patrick Cousot and Radhia Cousot. Temporal abstract interpretation. In Mark N. Wegman and Thomas W. Reps, editors, *Proc. of POPL '00*, pages 12–25. ACM, 2000.

6 Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching system metrics. *IEEE Transactions on Software Engineering*, 35(2):258–273, 2009.

7 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318:323–354, 2004.

8 Uli Fahrenberg and Axel Legay. The quantitative linear-time-branching-time spectrum. *Theoretical Computer Science*, 538:54–69, 2014.

9 Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.

10 Bart Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1st edition, January 1999.

11 Bartek Klin. The least fibred lifting and the expressivity of coalgebraic modal logic. In *Proc. of CALCO '05*, pages 247–262. Springer, 2005. LNCS 3629.

12 Bartek Klin. Coalgebraic modal logic beyond sets. In *Proc. of MFPS '07*, volume 173 of *ENTCS*, pages 177–201, 2007.

13 Bartosz Klin. *An Abstract Coalgebraic Approach to Process Equivalence for Well-Behaved Operational Semantics*. PhD thesis, University of Aarhus, 2004.

14 Yuichi Komorida, Shin-ya Katsumata, Clemens Kupke, Jurriaan Rot, and Ichiro Hasuo. Expressivity of quantitative modal logics: Categorical foundations via codensity and approximation. In *Proc. LICS '21*, pages 1–14. IEEE, 2021.

15 Barbara König and Christina Mika-Michalski. (Metric) bisimulation games and real-valued modal logics for coalgebras. In *Proc. of CONCUR '18*, volume 118 of *LIPIcs*, pages 37:1–37:17. Schloss Dagstuhl – Leibniz Center for Informatics, 2018.

16 Antonín Kucera and Javier Esparza. A logical viewpoint on process-algebraic quotients. *Journal of Logic and Computation*, 13(6):863–880, 2003.

17 Clemens Kupke and Dirk Pattinson. Coalgebraic semantics of modal logics: An overview. *Theoretical Computer Science*, 412:5070–5094, 2011.

18 Clemens Kupke and Jurriaan Rot. Expressive logics for coinductive predicates. In *Proc. of CSL '20*, volume 152 of *LIPIcs*, pages 26:1–26:18. Schloss Dagstuhl – Leibniz Center for Informatics, 2020.

19 George Markowsky. Chain-complete posets and directed sets with applications. *Algebra Universalis*, 6(1):53–68, 1976.

20 E. J. McShane. Extension of range of functions. *Bull. Amer. Math. Soc.*, 40(12):837–842, 1934.

21 Stefan Milius, Dirk Pattinson, and Lutz Schröder. Generic trace semantics and graded monads. In *Proc. of CALCO '15*, volume 35 of *LIPIcs*, pages 253–269. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015.

22 Dusko Pavlovic, Michael Mislove, and James Worrell. Testing semantics: Connecting processes and process logics. In *Proc. of AMAST '06*, pages 308–322. Springer, 2006. LNCS 4019.

23 Damien Pous. Complete lattices and up-to techniques. In *Proc. of APLAS '07*, pages 351–366. Springer, 2007. LNCS 4807.

24 Franck van Breugel and James Worrell. A behavioural pseudometric for probabilistic transition systems. *Theoretical Computer Science*, 331:115–142, 2005.

25 Rob van Glabbeek. The linear time – branching time spectrum I. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001.

26 Hassler Whitney. Analytic extensions of differentiable functions defined in closed sets. *Transactions of the American Mathematical Society*, 36(1):63–89, 1934.

A Curry-Howard Correspondence for Linear, Reversible Computation

Kostia Chardonnet
Université Paris-Saclay, Inria, CNRS, ENS Paris-Saclay, LMF, 91190, Gif-sur-Yvette, France
Équipe Quacs, Inria, Palaiseau, France
Université Paris Cité, CNRS, IRIF, 75013, Paris, France

Alexis Saurin
Université Paris Cité, CNRS, IRIF, 75013, Paris, France
Inria πr^2, Paris, France

Benoît Valiron
Université Paris-Saclay, Inria, CentraleSupélec, CNRS, ENS Paris-Saclay, LMF, 91190, Gif-sur-Yvette, France
Équipe Quacs, Inria, Palaiseau, France

Abstract

In this paper, we present a linear and reversible programming language with inductives types and recursion. The semantics of the languages is based on pattern-matching; we show how ensuring syntactical exhaustivity and non-overlapping of clauses is enough to ensure reversibility. The language allows to represent any Primitive Recursive Function. We then give a Curry-Howard correspondence with the logic μMALL: linear logic extended with least fixed points allowing inductive statements. The critical part of our work is to show how primitive recursion yields circular proofs that satisfy μMALL validity criterion and how the language simulates the cut-elimination procedure of μMALL.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Equational logic and rewriting

Keywords and phrases Reversible Computation, Linear Logic, Curry-Howard

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.13

Related Version *Full Version*: https://hal.archives-ouvertes.fr/hal-03747425

Funding This work has been partially funded by the French National Research Agency (ANR) under the research projects SoftQPRO ANR17-CE25-0009-02 and RECIPROG ANR-21-CE48-019-01 and within the framework of "Plan France 2030", under the research project ANR-22-PETQ-0007.

1 Introduction

$$\dfrac{\begin{array}{c}\vdots\,s\\ A\to B\end{array}\qquad\begin{array}{c}\vdots\,t\\ A\end{array}}{B}$$

Figure 1 Modus Ponens.

Computation and logic are two faces of the same coin. For instance, consider a proof s of $A \to B$ and a proof t of A. With the logical rule *Modus Ponens* one can construct a proof of B: Figure 1 features a graphical presentation of the corresponding proof. Horizontal lines stand for deduction steps – they separate conclusions (below) and hypotheses (above). These deduction steps can be stacked vertically up to axioms in order to describe complete proofs. In Figure 1 the proofs of A and $A \to B$ are symbolized with vertical ellipses. The ellipsis annotated with s indicates that s is a complete proof of $A \to B$ while t stands for a complete proof of A.

This connection is known as the *Curry-Howard correspondence* [7, 10]. In this general framework, types correspond to formulas and programs to proofs, while program evaluation is mirrored with proof simplification (the so-called cut-elimination). The Curry-Howard correspondence formalizes the fact that the proof s of $A \to B$ can be regarded as a *function* – parametrized by an argument of type A – that produces a proof of B whenever it is fed with a proof of A. Therefore, the computational interpretation of Modus Ponens corresponds to the *application* of an argument (i.e. t) of type A to a function (i.e. s) of type $A \to B$. When computing the corresponding program, one substitutes the parameter of the function with t and get a result of type B. On the logical side, this corresponds to substituting every axiom introducing A in the proof s with the full proof t of A. This yields a direct proof of B without any invocation of the "lemma" $A \to B$.

Paving the way toward the verification of critical softwares, the Curry-Howard correspondence provides a versatile framework. It has been used to mirror first and second-order logics with dependent-type systems [5, 14], separation logics with memory-aware type systems [18, 12], resource-sensitive logics with differential privacy [9], logics with monads with reasoning on side-effects [21, 15], etc.

Reversible computation is a paradigm of computation which emerged as an energy-preserving model of computation in which data is never erased [8] that makes sure that, given some process f, there always exists an inverse process f^{-1} such that $f \circ f^{-1} = \mathrm{Id} = f^{-1} \circ f$. Many aspects of reversible computation have been considered, such as the development of reversible Turing Machines [16], reversible programming languages [11] and their semantics [6, 13]. However, the formal relationship between a logical system and a computational model have not been developed yet.

This paper aims at proposing a type system featuring inductive types for a purely linear and reversible language. We base our study on the approach presented in [20]. In this model, reversible computation is restricted to two main types: the tensor, written $A \otimes B$ and the co-product, written $A \oplus B$. The former corresponds to the type of all pairs of elements of type A and elements of type B, while the latter represents the disjoint union of all elements of type A and elements of type B. For instance, a bit can be typed with $\mathbb{1} \oplus \mathbb{1}$, where $\mathbb{1}$ is a type with only one element. The language in [20] offers the possibility to code isos – reversible maps – with pattern matching. An iso is for instance the swap operation, typed with $A \otimes B \leftrightarrow B \otimes A$. However, if [20] hints at an extension towards pure quantum computation, the type system is not formally connected to any logical system.

The problem of reversibility between finite type of same cardinality simply requires to check that the function is injective. That is no longer the case when we work with types of infinite cardinality such as natural numbers.

The main contribution of this work is a Curry-Howard correspondence for a purely reversible typed language in the style of [20], with added generalised inductive types and terminating recursion, enforced by the fact that recursive functions must be structurally recursive: each recursive call must be applied to a decreasing argument. We show how ensuring exhaustivity and non-overlapping of the clauses of the pattern-matching are enough to ensure reversibility and that the obtained language can encode any Primitive Recursive function [19]. For the Curry-Howard part, we capitalize on the logic $\mu\mathsf{MALL}$ [1, 3]: an extension of the additive and multiplicative fragment of linear logic with least and greatest fixed points allowing inductive and coinductive statements. This logic contains both a tensor and a co-product, and its strict linearity makes it a good fit for a reversible type system. In the literature, multiple proofs systems have been considered for $\mu\mathsf{MALL}$, some finitary proof system with explicit induction inferences à la Park [1] as well as non-well-founded proof

systems which allow to build infinite derivation [3, 2]. The present paper focuses on the latter. In general, an infinite derivation is called a *pre-proof* and is not necessarily consistent. To solve this problem μMALL comes equipped with a *validity criterion*, telling us when an infinite derivation can be considered as a logical proof. We show how the syntactical constraints of being structurally recursive imply the validity of pre-proofs.

Organisation of the paper. The paper is organised as follows: in Section 2 we present the language, its syntax, typing rules and semantics and show that any function that can be encoded in our language represents an isomorphism. In Section 3 we show that our language can encode any Primitive Recursive Function [19], this is shown by encoding the set of Recursive Primitive Permutations [17] functions. Then in Section 4, we develop on the Curry-Howard Correspondence part: we show, given a well-typed term from our language, how to translate it into a circular derivation of the logic μMALL and show that the given derivation respects the validity condition and how our evaluation strategy simulates the cut-elimination procedure of the logic.

2 First-order Isos

Our language is based on the one introduced by Sabry et al [20] which define isomorphisms between various types, included the type of lists. We build on the reversible part of the paper by extending the language to support both a more general rewriting system and more general inductive types: while they only allow the inductive type of lists, we consider arbitrary inductive types. The language is defined by layers. Terms and types are presented in Table 1, while typing derivations, inspired from μMALL, can be found in Tables 2 and 3. The language consists of the following pieces.

Basic type. They allow us to construct first-order terms. The constructors $\texttt{inj}_l$ and $\texttt{inj}_r$ represent the choice between either the left or right-hand side of a type of the form $A \oplus B$; the constructor $\langle , \rangle$ builds pairs of elements (with the corresponding type constructor $\otimes$); `fold` represents inductive structure of the types $\mu X.A$. A value can serve both as a result and as a pattern in the defining clause of an iso. We write $(x_1, \dots, x_n)$ for $\langle x_1, \langle \dots, x_n \rangle \rangle$ or $\overrightarrow{x}$ when n is non-ambiguous and $A_1 \otimes \cdots \otimes A_n$ for $A_1 \otimes (\cdots \otimes A_n)$ and A^n for $\underbrace{A \otimes \cdots \otimes A}_{n \text{ times}}$.

First-order isos. An iso of type $A \leftrightarrow B$ acts on terms of base types. An iso is a function of type $A \leftrightarrow B$, defined as a set of clauses of the form $\{v_1 \leftrightarrow e_1 \mid \dots \mid v_n \leftrightarrow e_n\}$. In the clauses, the tokens v_i are open values and e_i are expressions. In order to apply an iso to a term, the iso must be of type $A \leftrightarrow B$ and the term of type A. In the typing rules of isos, the $\texttt{OD}_A(\{v_1, \dots, v_n\})$ predicate (corrected from [20], as their definition makes it impossible to type Toffoli) syntactically enforces the exhaustivity and non-overlapping conditions on a set of well-typed values $v_1, \dots, v_n$ of type A. The typing conditions make sure that both the left-hand-side and right-hand-side of clauses satisfy this condition. Its formal definition can be found in Table 4 where $Val(e)$ is defined as $Val(\texttt{let}\, p = \omega\ p'\ \texttt{in}\ e) = Val(e)$, and $Val(v) = v$ otherwise. These checks are crucial to make sure that our isos are indeed reversible. In the last rule on Table 4, we define $\pi_1(S)$ and $\pi_2(S)$ as respectively $\{v \mid \langle v, w \rangle \in S\}$ and $\{w \mid \langle v, w \rangle \in S\}$ and S^1_v and S^2_v respectively as $\{w \mid \langle v, w \rangle \in S\}$ and $\{w \mid \langle w, v \rangle \in S\}$ Exhaustivity for an iso $\{v_1 \leftrightarrow e_1 \mid \dots \mid v_n \leftrightarrow e_n\}$ of type $A \leftrightarrow B$ means that the expressions on the left (resp. on the right) of the clauses describe all possible values for the type A (resp. the type B). Non-overlapping means that two expressions cannot match the same value. For instance, the left and right injections $\texttt{inj}_l\ v$ and $\texttt{inj}_r\ v'$ are non-overlapping

Table 1 Terms and types.

$$
\begin{array}{lll}
\text{(Base types)} & A, B ::= \mathbb{1} \mid A \oplus B \mid A \otimes B \mid \mu X.A & \\
\text{(Isos, first-order)} & \alpha ::= A \leftrightarrow B & \\
\text{(Values)} & v ::= () \mid x \mid \mathtt{inj}_l\ v \mid \mathtt{inj}_r\ v \mid \langle v_1, v_2 \rangle \mid \mathtt{fold}\ v & \\
\text{(Pattern)} & p ::= x \mid \langle p_1, p_2 \rangle & \\
\text{(Expressions)} & e ::= v \mid \mathtt{let}\ p_1 = \omega\ p_2\ \mathtt{in}\ e & \\
\text{(Isos)} & \omega ::= \{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\} \mid \mathbf{fix}\ f.\omega \mid f & \\
\text{(Terms)} & t ::= () \mid x \mid \mathtt{inj}_l\ t \mid \mathtt{inj}_r\ t \mid \langle t_1, t_2 \rangle \mid & \\
 & \quad \mathtt{fold}\ t \mid \omega\ t \mid \mathtt{let}\ p = t_1\ \mathtt{in}\ t_2 &
\end{array}
$$

while a variable x is always exhaustive. The construction $\mathbf{fix}\ g.\omega$ represents the creation of a recursive function, rewritten as $\omega[g := \mathbf{fix}\ g.\omega]$ by the operational semantics. Each recursive function needs to satisfy a structural recursion criteria: making sure that one of the input arguments strictly decreases on each recursive call. Indeed, since isos can be non-terminating (due to recursion), we need a criterion that implies termination to ensure that we work with total functions. If ω is of type $A \leftrightarrow B$, we can build its inverse $\omega^\perp : B \leftrightarrow A$ and show that their composition is the identity. In order to avoid conflicts between variables we will always work up to α-conversion and use Barendregt's convention [4, p.26] which consists in keeping all bound and free variables names distinct, even when this remains implicit.

The type system is split in two parts: one for terms (noted $\Delta; \Psi \vdash_e t : A$) and one for isos (noted $\Psi \vdash_\omega \omega : A \leftrightarrow B$). In the typing rules, the contexts Δ are sets of pairs that consist of a term-variable and a base type, where each variable can only occur once and Ψ is a singleton set of a pair of an iso-variable and an iso-type association.

▶ **Definition 1** (Structurally Recursive). *Given an iso* $\mathbf{fix}\ f.\{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\} : A_1 \otimes \cdots \otimes A_m \leftrightarrow C$*, it is structurally recursive if there is* $1 \leq j \leq m$ *such that* $A_j = \mu X.B$ *and for all* $i \in \{1, \ldots, n\}$ *we have that* v_i *is of the form* $(v_i^1, \ldots, v_i^m)$ *such that* v_i^j *is either:*

- *A closed value, in which case* e_i *does not contain the subterm* $f\ p$
- *Open, in which case for all subterm of the form* $f\ p$ *in* e_i *we have* $p = (x_1, \ldots, x_m)$ *and* $x_j : \mu X.B$ *is a strict subterm of* v_i^j*.*

Given a clause $v \leftrightarrow e$*, we call the value* v_i^j *(resp. the variable* x_j*) the* decreasing argument *(resp. the* focus*) of the structurally recursive criterion.*

▶ Remark 2. As we are focused on a very basic notion of structurally recursive function, the typing rules of isos allow to have at most one iso-variable in the context, meaning that we cannot have intertwined recursive call.

Finally, our language is equipped with a rewriting system $\to$ on terms, defined in Definition 4, that follows a deterministic call-by-value strategy: each argument of a function is fully evaluated before applying the substitution. This is done through the use of an evaluation context $C[]$, which consists of a term with a hole (where $C[t]$ is C where the hole has been filled with t). Due to the deterministic nature of the strategy we directly obtain the unicity of the normal form. The evaluation of an iso applied to a value relies on pattern-matching : the argument is matched against the left-hand-side of each clause until

Table 2 Typing of terms and expressions.

$$\frac{}{\emptyset;\Psi\vdash_e ():\mathbb{1}} \quad \frac{}{x:A;\Psi\vdash_e x:A} \quad \frac{\Delta;\Psi\vdash_e t:A}{\Delta;\Psi\vdash_e \mathtt{inj}_l\ t:A\oplus B} \quad \frac{\Delta;\Psi\vdash_e t:B}{\Delta;\Psi\vdash_e \mathtt{inj}_r\ t:A\oplus B}$$

$$\frac{\Delta_1;\Psi\vdash_e t_1:A \quad \Delta_2;\Psi\vdash_e t_2:B}{\Delta_1,\Delta_2;\Psi\vdash_e \langle t_1,t_2\rangle:A\otimes B} \quad \frac{\Delta;\Psi\vdash_e t:A[X\leftarrow \mu X.A]}{\Delta;\Psi\vdash_e \mathtt{fold}\ t:\mu X.A}$$

$$\frac{\Psi\vdash_\omega f:A\leftrightarrow B \quad \Delta;\Psi\vdash_e t:A}{\Delta;\Psi\vdash_e f\ t:B} \quad \frac{\vdash_\omega \omega:A\leftrightarrow B \quad \Delta;\Psi\vdash_e t:A}{\Delta;\Psi\vdash_e \omega\ t:B}$$

$$\frac{\Delta_1;\Psi\vdash_e t_1:A_1\otimes\cdots\otimes A_n \qquad \Delta_2,x_1:A_1,\ldots,x_n:A_n;\Psi\vdash_e t_2:B}{\Delta_1,\Delta_2;\Psi\vdash_e \mathtt{let}\,(x_1,\ldots,x_n)=t_1\ \mathtt{in}\ t_2:B}$$

Table 3 Typing of isos.

$$\frac{\begin{array}{cccc}\Delta_1\vdash_e v_1:A & \ldots & \Delta_n\vdash_e v_n:A & \mathtt{OD}_A(\{v_1,\ldots,v_n\})\\ \Delta_1;\Psi\vdash_e e_1:B & \ldots & \Delta_n;\Psi\vdash_e e_n:B & \mathtt{OD}_B(\{Val(e_1),\ldots,Val(e_n)\})\end{array}}{\Psi\vdash_\omega \{v_1\ \leftrightarrow\ e_1\ \mid\ \ldots\ \mid\ v_n\ \leftrightarrow\ e_n\}:A\leftrightarrow B.}$$

$$\frac{}{f:\alpha\vdash_\omega f:\alpha} \qquad \frac{f:\alpha\vdash_\omega \omega:\alpha \qquad \textbf{fix}\ f.\omega \text{ is structurally recursive}}{\Psi\vdash_\omega \textbf{fix}\ f.\omega:\alpha}$$

one of them matches (written $\sigma[v]=v'$), in which case the pattern-matching, as defined in Table 5, returns a substitution σ that sends variables to values. Because we ensure exhaustivity and non-overlapping (Lemma 5), the pattern-matching can always occur on well-typed terms. The *support* of a substitution σ is defined as $\mathrm{supp}(\sigma)=\{x\mid (x\mapsto v)\in\sigma\}$.

▶ **Definition 3** (Substitution). *Applying substitution σ on an expression t, written $\sigma(t)$, is defined as : $\sigma(()) = (), \sigma(x) = v$ if $\{x\mapsto v\}\subseteq\sigma, \sigma(\mathtt{inj}_r\ t) = \mathtt{inj}_r\ \sigma(t), \sigma(\mathtt{inj}_l\ t) = \mathtt{inj}_l\ \sigma(t), \sigma(\langle t,t'\rangle) = \langle\sigma(t),\sigma(t')\rangle, \sigma(\omega\ t) = \omega\ \sigma(t)$ and $\sigma(\mathtt{let}\,p = t_1\ \mathtt{in}\ t_2) = (\mathtt{let}\,p = \sigma(t_1)\ \mathtt{in}\ \sigma(t_2))$.*

▶ **Definition 4** (Evaluation relation $\to$). *We define $\to$ the rewriting system of our language as follows:*

$$\frac{t_1\to t_2}{C[t_1]\to C[t_2]}\ \text{Cong} \qquad \frac{\sigma[p]=v}{\mathtt{let}\,p=v\ \mathtt{in}\ t\to\sigma(t)}\ \text{LetE} \qquad \frac{}{(\textbf{fix}\ f.\omega)\to\omega[f:=(\textbf{fix}\ f.\omega)]}\ \text{IsoRec}$$

$$\frac{\sigma[v_i]=v'}{\{v_1\ \leftrightarrow\ e_1\ \mid\ \ldots\ \mid\ v_n\ \leftrightarrow\ e_n\}\ v'\to\sigma(e_i)}\ \text{IsoApp}$$

with $C ::= [\,] \mid \mathtt{inj}_l\ C \mid \mathtt{inj}_r\ C \mid \omega\ C \mid \mathtt{let}\,p = C\ \mathtt{in}\ t \mid \langle C,v\rangle \mid \langle v,C\rangle$

As usual we note $\to^*$ for the reflexive transitive closure of $\to$.

▶ **Lemma 5** ($\mathtt{OD}_A(A)$ ensures exhaustivity and non-overlapping.). *Let $\mathtt{OD}_A(S)$ and $\vdash_e v:A$, then there exists a unique $v'\in S$ such that v' matches v under substitution σ, i.e. $\sigma[v']=v$.*

As mentioned above, from any iso $\omega:A\leftrightarrow B$ we can build its inverse $\omega^\perp:B\leftrightarrow A$, the inverse operation is defined inductively on ω and is given in Definition 6.

Table 4 Exhaustivity and Non-Overlapping.

$$\frac{}{\mathtt{OD}_A(\{x\})} \qquad \frac{}{\mathtt{OD}_{\mathbb{1}}(\{()\})} \qquad \frac{\mathtt{OD}_A(S) \qquad \mathtt{OD}_B(T)}{\mathtt{OD}_{A\oplus B}(\{\mathtt{inj}_l\ v \mid v \in S\} \cup \{\mathtt{inj}_r\ v \mid v \in T\})}$$

$$\frac{\mathtt{OD}_{A[X\leftarrow \mu X.A]}(S)}{\mathtt{OD}_{\mu X.A}(\{\mathtt{fold}\ v \mid v \in S\})} \qquad \frac{\begin{array}{c}\mathtt{OD}_A(\pi_1(S)), \forall v \in \pi_1(S), \mathtt{OD}_B(S^1_v) \\ \text{or } \mathtt{OD}_B(\pi_2(S)), \forall v \in \pi_2(S), \mathtt{OD}_A(S^2_v)\end{array}}{\mathtt{OD}_{A\otimes B}(S = \{\langle v_1, v'_1\rangle, \ldots, \langle v_n, v'_n\rangle\})}$$

Table 5 Pattern-matching.

$$\frac{\sigma[e] = e'}{\sigma[\mathtt{inj}_l\ e] = \mathtt{inj}_l\ e'} \qquad \frac{\sigma[e] = e'}{\sigma[\mathtt{inj}_r\ e] = \mathtt{inj}_r\ e'} \qquad \frac{\sigma = \{x \mapsto e\}}{\sigma[x] = e} \qquad \frac{\sigma[e] = e'}{\sigma[\mathtt{fold}\ e] = \mathtt{fold}\ e'}$$

$$\frac{\sigma_2[e_1] = e'_1 \quad \sigma_1[e_2] = e'_2 \quad \mathrm{supp}(\sigma_1) \cap \mathrm{supp}(\sigma_2) = \emptyset \quad \sigma = \sigma_1 \cup \sigma_2}{\sigma[\langle e_1, e_2\rangle] = \langle e'_1, e'_2\rangle} \qquad \frac{}{\sigma[()] = ()}$$

▶ **Definition 6** (Inversion). *Given an iso ω, we define its dual $\omega^\perp$ as : $f^\perp = f, (\mathbf{fix}\ f.\omega)^\perp = \mathbf{fix}\ f.\omega^\perp, \{(v_i \leftrightarrow e_i)_{i\in I}\}^\perp = \{((v_i \leftrightarrow e_i)^\perp)_{i\in I}\}$ And the inverse of a clause as:*

$$\left(\begin{array}{lll} v_1 & \leftrightarrow & \mathtt{let}\ p_1 = \omega_1\ p'_1\ \mathtt{in} \\ & \cdots & \\ & & \mathtt{let}\ p_n = \omega_n\ p'_n\ \mathtt{in}\ v'_1 \end{array}\right)^\perp := \left(\begin{array}{lll} v'_1 & \leftrightarrow & \mathtt{let}\ p'_n = \omega_n^\perp\ p_n\ \mathtt{in} \\ & \cdots & \\ & & \mathtt{let}\ p'_1 = \omega_1^\perp\ p_1\ \mathtt{in}\ v_1 \end{array}\right).$$

We can show that the inverse is well-typed and behaves as expected:

▶ **Lemma 7** (Inversion is well-typed). *Given $\Psi \vdash_\omega \omega : A \leftrightarrow B$, then $\Psi \vdash_\omega \omega^\perp : B \leftrightarrow A$.*

▶ **Theorem 8** (Isos are isomorphisms). *For all well-typed isos $\vdash_\omega \omega : A \leftrightarrow B$, and for all well-typed values $\vdash_e v : A$, if $(\omega\ (\omega^\perp\ v)) \to^* v'$ then $v = v'$.*

▶ **Example 9.** We can define the iso of type : $A \oplus (B \oplus C) \leftrightarrow C \oplus (A \oplus B)$ as

$$\left\{\begin{array}{l} \mathtt{inj}_l\ (a) \leftrightarrow \mathtt{inj}_r\ (\mathtt{inj}_l\ (a)) \\ \mathtt{inj}_r\ (\mathtt{inj}_l\ (b)) \leftrightarrow \mathtt{inj}_r\ (\mathtt{inj}_r\ (b)) \\ \mathtt{inj}_r\ (\mathtt{inj}_r\ (c)) \leftrightarrow \mathtt{inj}_l\ (c) \end{array}\right\}$$

▶ **Example 10.** We give the encoding of the isomorphism $map(\omega)$ and its inverse: for any given iso $\vdash \omega : A \leftrightarrow B$ in our language, we can define $map(\omega) : [A] \leftrightarrow [B]$ where $[A] = \mu X.\mathbb{1} \oplus (A \otimes X)$ is the type of lists of type A and $[\,]$ is the empty list $(\mathtt{fold}\ (\mathtt{inj}_l\ ()))$ and $h :: t$ is the list construction $(\mathtt{fold}\ (\mathtt{inj}_r\ \langle h, t\rangle))$. We also give its dual $map(\omega)^\perp$ below, as given by Definition 6.

$$map(\omega) : [A] \leftrightarrow [B]$$
$$= \mathbf{fix}\ f. \left\{\begin{array}{ll} [\,] & \leftrightarrow [\,] \\ h :: t & \leftrightarrow \mathtt{let}\ h' = \omega\ h\ \mathtt{in} \\ & \quad \mathtt{let}\ t' = f\ t\ \mathtt{in} \\ & \quad h' :: t' \end{array}\right\}$$

$$map(\omega)^\perp : [B] \leftrightarrow [A]$$
$$= \mathbf{fix}\ f. \left\{\begin{array}{ll} [\,] & \leftrightarrow [\,] \\ h' :: t' & \leftrightarrow \mathtt{let}\ t = f\ t'\ \mathtt{in} \\ & \quad \mathtt{let}\ h = \omega^\perp h'\ \mathtt{in} \\ & \quad h :: t \end{array}\right\}$$

▶ Remark 11. In our two examples, the left and right-hand side of the $\leftrightarrow$ on each function respect both the criteria of exhaustivity – every-value of each type is being covered by at least one expression – and non-overlapping – no two expressions cover the same value. Both isos are therefore bijections.

The language enjoys the standard properties of typed languages of progress and subject reduction:

▶ **Lemma 12** (Subject Reduction). *If* $\Delta; \Psi \vdash_e t : A$ *and* $t \to t'$ *then* $\Delta; \Psi \vdash_e t' : A$.

▶ **Lemma 13** (Progress). *If* $\vdash_e t : A$ *then, either* t *is a value, or* $t \to t'$.

3 Computational Content

In this section, we study the computational content of our language. In the case of only finite types made up of the tensor, plus and unit, one can represent any bijection by case analysis. Homewer, with infinite types, the expressivity becomes less clear. We show that we can encode Recursive Primitive Permutations [17] (RPP), which shows us that we can encode at least all Primitive Recursive Functions [19].

We give a few reminders on the language RPP and its main results, then show how to encode it.

3.1 Reminder on RPP

RPP is a set of integer-valued functions of variable arity; we define it by arity as follows: we note RPP^k for the set of functions in RPP from $\mathbb{Z}^k$ to $\mathbb{Z}^k$, it is built inductively on $k \in \mathbb{N}$ by: the successor (S), the predecessor (P), the identity (ID) and the sign-change that are part of RPP^1. The swap function ($\mathcal{X}$) and the binary permutation $\mathcal{X}$ which sends the pair (x, y) to (y, x) are part of RPP^2 and then, for any function $f, g, h \in \mathrm{RPP}^k$ and $j \in \mathrm{RPP}^l$, we can build (i) the sequential composition $f; g \in \mathrm{RPP}^k$, (ii) the parallel composition $f \mid\mid j \in \mathrm{RPP}^{k+l}$ (iii) the iterator $It[f] \in \mathrm{RPP}^{k+1}$ and (iv) the selection $If[f, g, h] \in \mathrm{RPP}^{k+1}$.

Finally, the set of all functions that form RPP is taken as the union for all k of the RPP^k:

$$\mathrm{RPP} = \cup_{k \in \mathbb{N}} \mathrm{RPP}^k$$

We present the semantics of each constructors of RPP under a graphical form, as in [17], where the left-hand-side variables of the diagram represent the input of the function and the right-hand-side is the output of the function. The semantics of all those operators are given in Figure 2.

▶ Remark 14. In their paper [17], the authors make use of two other constructors: generalised permutations over $\mathbb{Z}^k$ and *weakenings* of functions, but those can actually be defined from the other constructors so that in the following section we do not give their encoding.

Then, if $f \in \mathrm{RPP}^k$ we can define an inverse f^{-1}:

▶ **Definition 15** (Inversion). *The inversion is defined as follow :*

$$\begin{array}{ccc} \mathrm{Id}^{-1} = \mathrm{Id} & \mathrm{S}^{-1} = \mathrm{P} & \mathrm{P}^{-1} = \mathrm{S} \\ \mathrm{Sign}^{-1} = \mathrm{Sign} & \mathcal{X}^{-1} = \mathcal{X} & (g; f)^{-1} = f^{-1}; g^{-1} \\ (f \mid\mid g)^{-1} = f^{-1} \mid\mid g^{-1} & (\mathbf{It}[f])^{-1} = \mathbf{It}[f^{-1}] & (\mathbf{If}[f, g, h])^{-1} = \mathbf{If}[f^{-1}, g^{-1}, h^{-1}] \end{array}$$

▶ **Proposition 16** (Inversion defines an inverse [17]). *Given* $f \in \mathrm{RPP}^k$ *then* $f; f^{-1} = \mathrm{Id} = f^{-1}; f$

$$x \left[S\right] x+1 \qquad x \left[P\right] x-1 \qquad x \left[\text{Sign}\right] -x \qquad x \left[\text{Id}\right] x \qquad \begin{matrix} x \\ y \end{matrix}\left[\mathcal{X}\right]\begin{matrix} y \\ x \end{matrix}$$

$$\begin{matrix} x_1 \\ \vdots \\ x_n \end{matrix}\left[\, f;g \,\right]\begin{matrix} y_1 \\ \vdots \\ y_n \end{matrix} = \begin{matrix} x_1 \\ \vdots \\ x_n \end{matrix}\left[\, f \,\right]\left[\, g \,\right]\begin{matrix} y_1 \\ \vdots \\ y_n \end{matrix} \qquad \begin{matrix} x_1 \\ \vdots \\ x_k \\ x'_1 \\ \vdots \\ x'_l \end{matrix}\left[\, f \parallel g \,\right]\begin{matrix} y_1 \\ \vdots \\ y_k \\ y'_1 \\ \vdots \\ y'_l \end{matrix} = \begin{matrix} x_1 \\ \vdots \\ x_k \\ x'_1 \\ \vdots \\ x_l \end{matrix}\left[\begin{matrix} f \\ \\ g \end{matrix}\right]\begin{matrix} y_1 \\ \vdots \\ y_n \\ y'_1 \\ \vdots \\ y'_l \end{matrix}$$

$$\begin{matrix} x_1 \\ \vdots \\ x_n \\ x \end{matrix}\left[\, If[f,g,h] \,\right]\begin{matrix} y_1 \\ \vdots \\ y_n \\ x \end{matrix} = \begin{cases} f\ (x_1,\ldots,x_n) \text{ if } x>0 \\ g\ (x_1,\ldots,x_n) \text{ if } x=0 \\ h\ (x_1,\ldots,x_n) \text{ if } x<0 \end{cases} \qquad \begin{matrix} x_1 \\ \vdots \\ x_n \\ x \end{matrix}\left[\, It[f] \,\right]\begin{matrix} y_1 \\ \vdots \\ y_n \\ x \end{matrix} = \underbrace{(f;\ldots;f)}_{|\ x\ |}(x_1,\ldots,x_n)$$

Figure 2 Generators of RPP.

▶ **Theorem 17** (Soudness & Completeness [17]). *RPP is PRF-Complete and PRF-Sound: it can represent any Primitive Recursive Function and every function in RPP can be represented in PRF.*

3.2 From RPP to Isos

We start by defining the type of stricly positive natural numbers, npos, as $\text{npos} = \mu X.\mathbb{1} \oplus X$. We define $\underline{n}$, the encoding of a positive natural number into a value of type npos as $\underline{1} = \texttt{fold}\ \texttt{inj}_l\ ()$ and $\underline{n+1} = \texttt{fold}\ \texttt{inj}_r\ \underline{n}$. Finally, we define the type of integers as $Z = \mathbb{1} \oplus (\text{npos} \oplus \text{npos})$ along with $\overline{z}$ the encoding of any $z \in \mathbb{Z}$ into a value of type Z defined as: $\overline{0} = \texttt{inj}_l\ ()$, $\overline{z} = \texttt{inj}_r\ \texttt{inj}_l\ \underline{z}$ for z positive, and $\overline{z} = \texttt{inj}_r\ \texttt{inj}_r\ \underline{-z}$ for z negative. Given some function $f \in \text{RPP}^k$, we will build an iso $\text{isos}(f) : Z^k \leftrightarrow Z^k$ which simulates f. $\text{isos}(f)$ is defined by the size of the proof that f is in RPP^k.

▶ **Definition 18** (Encoding of the primitives).

- *The Sign-change of type* $Z \leftrightarrow Z$ *is* $\left\{ \begin{matrix} \texttt{inj}_r\ (\texttt{inj}_l\ x) & \leftrightarrow & \texttt{inj}_r\ (\texttt{inj}_r\ x) \\ \texttt{inj}_r\ (\texttt{inj}_r\ (x)) & \leftrightarrow & \texttt{inj}_r\ (\texttt{inj}_l\ x) \\ \texttt{inj}_l\ () & \leftrightarrow & \texttt{inj}_l\ () \end{matrix} \right\}$
- *The identity is* $\{x \leftrightarrow x\} : Z \leftrightarrow Z$
- *The Swap is* $\{(x,y) \leftrightarrow (y,x)\} : Z^2 \leftrightarrow Z^2$
- *The Predecessor is the inverse of the Successor*
- The Successor is

$$\left\{ \begin{matrix} \texttt{inj}_l\ () & \leftrightarrow & \texttt{inj}_r\ (\texttt{inj}_l\ (\texttt{fold}\ (\texttt{inj}_l\ ()))) \\ \texttt{inj}_r\ (\texttt{inj}_l\ x) & \leftrightarrow & \texttt{inj}_r\ (\texttt{inj}_l\ (\texttt{fold}\ (\texttt{inj}_r\ x))) \\ \texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{fold}\ (\texttt{inj}_l\ ()))) & \leftrightarrow & \texttt{inj}_l\ () \\ \texttt{inj}_r\ (\texttt{inj}_r\ (\texttt{fold}\ (\texttt{inj}_r\ x))) & \leftrightarrow & \texttt{inj}_r\ (\texttt{inj}_r\ x) \end{matrix} \right\} : Z \leftrightarrow Z$$

▶ **Definition 19** (Encoding of Composition). *Let* $f, g \in \mathrm{RPP}^j$, $\omega_f = \mathrm{isos}(f)$ *and* $\omega_g = \mathrm{isos}(g)$ *the isos encoding* f *and* g*, we build* $\mathrm{isos}(f;g)$ *of type* $Z^j \leftrightarrow Z^j$ *as:*

$$\mathrm{isos}(f;g) = \left\{ \begin{array}{ccc} & & \texttt{let}\ (y_1,\ldots,y_j) = \omega_f\ (x_1,\ldots,x_j)\ \texttt{in} \\ (x_1,\ldots,x_j) & \leftrightarrow & \texttt{let}\ (z_1,\ldots,z_j) = \omega_g\ (y_1,\ldots,y_j)\ \texttt{in} \\ & & (z_1,\ldots,z_j) \end{array} \right\}$$

▶ **Definition 20** (Encoding of Parallel Composition). *Let* $f \in \mathrm{RPP}^j$ *and* $g \in \mathrm{RPP}^k$*, and* $\omega_f = \mathrm{isos}(f)$ *and* $\omega_g = \mathrm{isos}(g)$*, we define* $\mathrm{isos}(f \mid\mid g)$ *of type* $Z^{j+k} \leftrightarrow Z^{j+k}$ *as:.*

$$\mathrm{isos}(f \mid\mid g) = \left\{ \begin{array}{ccc} & & \texttt{let}\ (x'_1,\ldots,x'_j) = \omega_f\ (x_1,\ldots,x_j)\ \texttt{in} \\ (x_1,\ldots,x_j,y_1,\ldots,y_k) & \leftrightarrow & \texttt{let}\ (y'_1,\ldots,y'_k) = \omega_g\ (y_1,\ldots,y_k)\ \texttt{in} \\ & & (x'_1,\ldots,x'_j,y'_1,\ldots,y'_k) \end{array} \right\}$$

▶ **Definition 21** (Encoding of Finite Iteration). *Let* $f \in \mathrm{RPP}^k$*, and* $\omega_f = \mathrm{isos}(f)$*, we encode the finite iteration* $\mathbf{It}[f] \in \mathrm{RPP}^{k+1}$ *with the help of an auxiliary iso,* ω_{aux}*, of type* $Z^k \otimes \mathrm{npos} \leftrightarrow Z^k \otimes \mathrm{npos}$ *doing the finite iteration using* npos*, defined as:*

$$\omega_{\mathrm{aux}} = \mathbf{fix} g. \left\{ \begin{array}{ccc} (\overrightarrow{x}, \texttt{fold}\ (\texttt{inj}_l\ ())) & \leftrightarrow & \texttt{let}\ \overrightarrow{y} = \omega_f\ \overrightarrow{x}\ \texttt{in} \\ & & (\overrightarrow{y}, \texttt{fold}\ (\texttt{inj}_l\ ())) \\ (\overrightarrow{x}, \texttt{fold}\ (\texttt{inj}_r\ n)) & \leftrightarrow & \texttt{let}\ (\overrightarrow{y}) = \omega_f\ (\overrightarrow{x})\ \texttt{in} \\ & & \texttt{let}\ (\overrightarrow{z}, n') = g\ (\overrightarrow{y}, n)\ \texttt{in} \\ & & (\overrightarrow{z}, \texttt{fold}\ (\texttt{inj}_r\ n')) \end{array} \right\}$$

We can now properly define $\mathrm{isos}(It[f])$ *of type* $Z^{k+1} \leftrightarrow Z^{k+1}$ *as:*

$$\mathrm{isos}(It[f]) = \left\{ \begin{array}{ccc} (\overrightarrow{x}, \texttt{inj}_l\ ()) & \leftrightarrow & (\overrightarrow{x}, \texttt{inj}_l\ ()) \\ (\overrightarrow{x}, \texttt{inj}_r\ (\texttt{inj}_l\ z)) & \leftrightarrow & \texttt{let}\ (\overrightarrow{y}, z') = \omega_{aux}(\overrightarrow{x}, z)\ \texttt{in} \\ & & (\overrightarrow{y}, \texttt{inj}_r\ (\texttt{inj}_l\ z')) \\ (\overrightarrow{x}, \texttt{inj}_r\ (\texttt{inj}_r\ z)) & \leftrightarrow & \texttt{let}\ (\overrightarrow{y}, z') = \omega_{aux}(\overrightarrow{x}, z)\ \texttt{in} \\ & & (\overrightarrow{y}, \texttt{inj}_r\ (\texttt{inj}_r\ z')) \end{array} \right\}$$

▶ **Definition 22** (Encoding of Selection). *Let* $f, g, h \in \mathrm{RPP}^k$ *and their corresponding isos* $\omega_f = \mathrm{isos}(f), \omega_g = \mathrm{isos}(g), \omega_h = \mathrm{isos}(h)$*. We define* $\mathrm{isos}(If[f,g,h])$ *of type* $Z^{k+1} \leftrightarrow Z^{k+1}$ *as:*

$$\mathrm{isos}(If[f,g,h]) = \left\{ \begin{array}{ccc} (\overrightarrow{x}, \texttt{inj}_r\ (\texttt{inj}_l\ z)) & \leftrightarrow & \texttt{let}\ \overrightarrow{x'} = \omega_f(\overrightarrow{x})\ \texttt{in}\ (\overrightarrow{x'}, \texttt{inj}_r\ (\texttt{inj}_l\ z)) \\ (\overrightarrow{x}, \texttt{inj}_l\ ()) & \leftrightarrow & \texttt{let}\ \overrightarrow{x'} = \omega_g(\overrightarrow{x})\ \texttt{in}\ (\overrightarrow{x'}, \texttt{inj}_l\ ()) \\ (\overrightarrow{x}, \texttt{inj}_r\ (\texttt{inj}_r\ z)) & \leftrightarrow & \texttt{let}\ \overrightarrow{x'} = \omega_h(\overrightarrow{x})\ \texttt{in}\ (\overrightarrow{x'}, \texttt{inj}_r\ (\texttt{inj}_r\ z)) \end{array} \right\}$$

▶ **Theorem 23** (The encoding is well-typed). *Let* $f \in \mathrm{RPP}^k$*, then* $\vdash_\omega \mathrm{isos}(f) : Z^k \leftrightarrow Z^k$.

▶ **Theorem 24** (Simulation). *Let* $f \in \mathrm{RPP}^k$ *and* $n_1,\ldots,n_k$ *elements of* $\mathbb{Z}$ *such that* $f(n_1,\ldots,n_k) = (m_1,\ldots,m_k)$ *then* $\mathrm{isos}(f)(\overline{n_1},\ldots,\overline{n_k}) \to^* (\overline{m_1},\ldots,\overline{m_k})$

▶ Remark 25. Notice that $\mathrm{isos}(f)^\perp \neq \mathrm{isos}(f^{-1})$, due to the fact that $\mathrm{isos}(f)^\perp$ will inverse the order of the *let* constructions, which will not be the case for $\mathrm{isos}(f^{-1})$. They can nonetheless be considered equivalent up to a permutation of *let* constructions and renaming of variable.

4 Proof Theorical Content

We want to relate our langage of isos to proofs in a suitable logic. As mentioned earlier, an iso $\vdash_\omega \omega : A \leftrightarrow B$ corresponds to both a computation sending a value of type A to a result of type B and a computation sending a value of type B to a result of type A. Therefore we want to be able to translate an iso into a proof isomorphism : two proofs π and $\pi^\perp$ of respectively $A \vdash B$ and $B \vdash A$ such that their composition reduces through the cut-elimination to the identity either on A or on B depending on the way we make the cut between those proofs. Since we are working in a linear system with inductive types we will use an extension of Linear Logic called μMALL : linear logic with least and greatest fixed points, which allows us to reason about inductive and coinductive statements. μMALL also allows us to consider infinite derivation trees, which is required as our isos can contain recursive variables. We need to be careful though: infinite derivations cannot always be considered as proofs, hence μMALL comes with a validity criterion on infinite derivations trees (called *pre-proofs*) that tells us whether such derivations are indeed proofs. We recall briefly the basic notions of μMALL, while more details can be found in [2].

4.1 Background on μMALL

Given an infinite set of variables $\mathcal{V} = \{X, Y, \dots\}$, we call *formulas* of μMALL the objects generated by $A, B ::= X \mid \mathbb{1} \mid \mathbb{0} \mid \top \mid \bot \mid A \otimes B \mid A \parr B \mid A \oplus B \mid A \,\&\, B \mid \mu X.A \mid \nu X.A$ where μ and ν bind the variable X in A. The negation on formula is defined in the usual way: $X^\perp = X, \mathbb{0}^\perp = \top, \mathbb{1}^\perp = \bot, (A \parr B)^\perp = A^\perp \otimes B^\perp, (A \oplus B)^\perp = A^\perp \,\&\, B^\perp, (\nu X.A)^\perp = \mu X.A^\perp$ having $X^\perp = X$ is harmless since we only deal with closed formulas.

We call an *occurrence*, a word of the form $\alpha \cdot w$ where $\alpha \in \mathfrak{A}_{\text{fresh}}$ an infinite set of *atomic* addresses and its dual $\mathfrak{A}^\perp_{\text{fresh}} = \{\alpha^\perp \mid \alpha \in \mathfrak{A}_{\text{fresh}}\}$ and w a word over $\{l, r, i\}^*$ (for *left*, *right* and *inside*) and *formulas occurrences* $F, G, H, \dots$ as a pair of a *formula* and an *occurrence*, written A_α. Finally we write Σ, Φ for *formula contexts*: sets of formulas occurrences. We write $A_\alpha \equiv B_\beta$ when $A = B$. Negation is lifted to formulas with $(A_\alpha)^\perp = A^\perp_{\alpha^\perp}$ where $(\alpha \cdot w)^\perp = \alpha^\perp \cdot w$ and $(\alpha^\perp \cdot w)^\perp = \alpha \cdot w$. In general, we write α, β for occurrences.

The connectives need then to be lifted to occurrences as well:

- Given $\# \in \{\otimes, \oplus, \parr, \&\}$, if $F = A_{\alpha l}$ and $G = B_{\alpha r}$ then $(F \# G) = (A \# B)_\alpha$
- Given $\# \in \{\mu, \nu\}$ if $F = A_{\alpha i}$ then $\#X.F = (\#X.A)_\alpha$

Occurrences allow us to follow a subformula uniquely inside a derivation. Since in μMALL we only work with formula occurrences, we simply use the term *formula*.

The (possibly infinite) derivation trees of μMALL, called *pre-proofs* are coinductively generated by the rules given in Figure 3. We say that a formula is *principal* when it is the formula that the rule is being applied to.

Among the infinite derivations that μMALL offer we can look at the *circular* ones: an infinite derivation is circular if it has finitely many different subtrees. The circular derivation can therefore be represented in a more compact way with the help of *back-edges*: arrows in the derivation that represent a repetition of the derivation. Derivations with back-edge are represented with the addition of sequents marked by a back-edge label, noted $\vdash^f$, and an additional rule, $\dfrac{}{\vdash \Sigma}\ be(f)$, which represent a back-edge pointing to the sequent $\vdash^f$. We take the convention that from the root of the derivation from to rule $be(f)$ there must be exactly one sequent annotated by f.

$$\frac{F \equiv G}{\vdash F^\perp, G}\ id \qquad \frac{\vdash \Sigma, F \quad \vdash \Phi, F^\perp}{\vdash \Sigma, \Phi}\ cut \qquad \frac{}{\vdash \top, \Sigma}\ \top$$

$$\frac{}{\vdash \mathbb{1}}\ \mathbb{1} \qquad \frac{\vdash F, G, \Sigma}{\vdash F \parr G, \Sigma}\ \parr \qquad \frac{\vdash F, \Sigma \quad \vdash G, \Phi}{\vdash F \otimes G, \Sigma, \Phi}\ \otimes$$

$$\frac{\vdash F, \Sigma \quad \vdash G, \Sigma}{\vdash F \,\&\, G, \Sigma}\ \& \qquad \frac{\vdash F_i, \Sigma}{\vdash F_1 \oplus F_2, \Sigma}\ \oplus^i\ i \in \{1,2\} \qquad \frac{\Sigma}{\vdash \Sigma, \bot}\ \bot$$

$$\frac{\vdash F[X \leftarrow \mu X.F], \Sigma}{\vdash \mu X.F, \Sigma}\ \mu \qquad \frac{\vdash F[X \leftarrow \nu X.F], \Sigma}{\vdash \nu X.F, \Sigma}\ \nu$$

Figure 3 Rules for μMALL.

▶ **Example 26.** An infinite derivation and two different circular representations with back-edges.

$$\dfrac{\dfrac{\vdots}{\vdash \mu X.X}\ \mu}{\vdash \mu X.X}\ \mu \qquad \dfrac{\dfrac{}{\vdash \mu X.X}\ be(f)}{\vdash^f \mu X.X}\ \mu \qquad \dfrac{\dfrac{\dfrac{}{\vdash \mu X.X}\ be(f)}{\vdash \mu X.X}\ \mu}{\vdash^f \mu X.X}\ \mu$$

While a circular proof has multiple finite representations (depending on where the back-edge is placed), they can all be mapped back to the same infinite derivation via an infinite unfolding of the back-edge and forgetting the back-edge labels:

▶ **Definition 27** (Unfolding). *We define the unfolding of a circular derivation P with a valuation v from back-edge labels to derivations by:*

- $\mathcal{U}\left(P : \dfrac{P_1, \dots, P_n}{\vdash \Sigma}\ r\ , v \right) = \dfrac{\mathcal{U}(P_1, v), \dots, \mathcal{U}(P_n, v)}{\vdash \Sigma}\ r$
- $\mathcal{U}(be(f), v) = v(f)$
- $\mathcal{U}\left(P : \dfrac{P_1, \dots, P_n}{\vdash^f \Sigma}\ r\ , v \right) = \left(\pi = \dfrac{\mathcal{U}(P_1, v'), \dots, \mathcal{U}(P_n, v')}{\vdash \Sigma}\ r \right)$ *with* $v'(g) = \pi$ *if* $g = f$ *else* $v(g)$.

μMALL comes with a validity criterion on pre-proofs that determines when a pre-proof can be considered as a proof: mainly, whether or not each infinite branch can be justified by a form of coinductive reasoning. The criterion also ensures that the cut-elimination procedure holds. For that, we can define a notion of *thread* [3, 2]: an infinite sequence of tuples of formulas, sequents and directions (either up or down). Intuitively, these threads follow some formula starting from the root of the derivation and start by going up. If the thread encounters an axiom rule, it will bounce back and start going down in the dual formula of the axiom rule. It may bounce back again, when going down on a cut rule, if it follows the cut-formula. A thread will be called *valid* when it is non-stationary (does not follow a formula that is never a principal formula of a rule), and when in the set of formulas appearing infinitely often, the minimum formula (according to the subformula ordering) is a ν formula. For the multiplicative fragment, we say that a pre-proof is valid if for all infinite branches, there exists a valid thread, while for the additive part, we require a notion of *additive slices* and *persistent slices* which we do not detail here. Example 31 features an example of a valid proof together with its thread. More details can be found in [2].

4.2 Translating isos into μMALL

We start by giving the translation from isos to pre-proofs, and then show that they are actually proofs, therefore obtaining a *static* correspondence between programs and proofs. We then show that our translation entails a *dynamic* correspondence between the evaluation procedure of our language and the cut-elimination procedure of μMALL. This will imply that the proofs we obtain are indeed isomorphisms, meaning that if we cut the aforementioned proofs π and $\pi^\perp$, performing the cut-elimination procedure would give either the identity on A or the identity on B.

The derivation we obtain are *circular*, and we therefore translate the isos directly into finite derivations with back-edge, written $circ(\omega)$. We can define another translation into infinite derivations as the composition of $circ()$ with the unfolding: $[\![\omega]\!] = \mathcal{U}(circ(\omega))$.

Because we need to keep track of which formula is associated to which variable from the typing context, the translation uses a slightly modified version of μMALL in which contexts are split in two parts, written $\Upsilon;\Theta$, where Υ is a list of formulas and Θ is a set of formulas associated with a term-variable (written $x : F$). When starting the translation of an iso of type $A \leftrightarrow B$, we start in the context $[A_\alpha];\emptyset$ (for some address α) and end in the context $[];\Theta$. The additional information of the variable in Θ is here to make sure we know how to split the contexts accordingly when needed later during the translation, with respect to the way they are split in the typing derivation. We write $\overline{\Theta} = \{F \mid x : F \in \Theta\}$ and $\underline{\Theta} = \{x : A \mid x : A_\alpha \in \Theta\}$. We also use another rule which allow to send the first formula from Υ to Θ and associate it a variable to the formula:

$$\frac{\Upsilon; x : F, \Theta \vdash G}{F :: \Upsilon; \Theta \vdash G}\ ex(x)$$

Given a derivation ι in this system, we write $\lVert\iota\rVert$ for the function that sends ι into a derivation of μMALL where (i) we remove all occurrence of the exchange rule (ii) the contexts $[];\Theta$ becomes $\overline{\Theta}$.

Given an iso $\omega : A \leftrightarrow B$ and initial addresses α, β, its translation into a derivation of μMALL of $A_\alpha \vdash B_\beta$ is described with three separate phases:

Iso Phase. The first phase consists in travelling through the syntactical definition of an iso, keeping as information the last encountered iso-variable bounded by a **fix** $f.\omega$ and calling the negative phase when encountering an iso of the form $\{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\}$ and attaching to the formulas A and B two distinct addresses α and β and to the sequent as a label of name of the last encountered iso-variable. Later on during the translation, this phase will be recalled when encountering another iso in one of the e_i, and, if said iso correspond to an iso-variable, we will create a back-edge pointing towards the corresponding sequents.

Negative Phase. Starting from some context $[A_\alpha], \Theta$, the negative phase consists into decomposing the formula A according to the way in which the values of type A on the left-hand-side of ω are decomposed. The negative phase works as follows: we consider a set of (list of values, typing judgement), written (l, ξ) where each element of the set corresponds to one clause $v \leftrightarrow e$ of the given iso and ξ is the typing judgement of e. The list of values corresponds to what is left to be decomposed in the left-hand-side of the clause (for instance if v is a pair $\langle v_1, v_2\rangle$ the list will have two elements to decompose). Each element of the list Υ will correspond to exactly one value in the list l. If the term that needs to be decomposed is a variable x, then we will apply the $ex(x)$ rule, sending the formula to the context Θ. The negative phase ends when the list is empty and hence when $\Upsilon = []$. When it is the

case, we can start decomposing ξ and the *positive phase* start. The negative phase is defined inductively on the first element of the list of every set, which are known by typing to have the same prefix, and is given in Figure 4.

Positive phase. The translation of an expression is pretty straightforward: each *let* and iso-application is represented by two `cut` rules, as usual in Curry-Howard correspondence. Keeping the variable-formula pair in the derivation is here to help us know how to split accordingly the context Θ when needed, while Υ is always empty and is therefore omitted. While the positive phase carry over the information of the last-seen iso-variable, it is not noted explicitly as it is only needed when calling the Iso Phase. The positive phase is given in Figure 5.

▶ Remark 28. While μMALL is presented in a one-sided way, we write $\Sigma \vdash \Phi$ for $\vdash \Sigma^{\perp}, \Phi$ in order to stay closer to the formalism of the type system of isos.

▶ **Definition 29.** *$circ(\omega, S, \alpha, \beta) = \pi$ takes a well-typed iso, a singleton set S of an iso-variable corresponding to the last iso-variable seen in the induction definition of ω and two fresh addresses α, β and produces a circular derivation of the variant of μMALL described above with back-edges. $circ(\omega, S, \alpha, \beta)$ is defined inductively on ω:*

- $circ(\mathbf{fix}\ f.\omega, S, \alpha, \beta) = circ(\omega, \{f\}, \alpha, \beta)$
- $circ(f, \{f\}, \alpha, \beta) = \dfrac{}{A_\alpha \vdash B_\beta}\ be(f)$
- $circ(\{(v_i \leftrightarrow e_i)_{i\in I}\} : A \leftrightarrow B, \{f\}, \alpha, \beta) = \left\Vert\ \dfrac{\mathtt{Neg}(([v_i], \xi_i)_{i\in I})}{A_\alpha \vdash^f B_\beta}\ \right\Vert$ *where ξ_i is the typing derivation of e_i.*

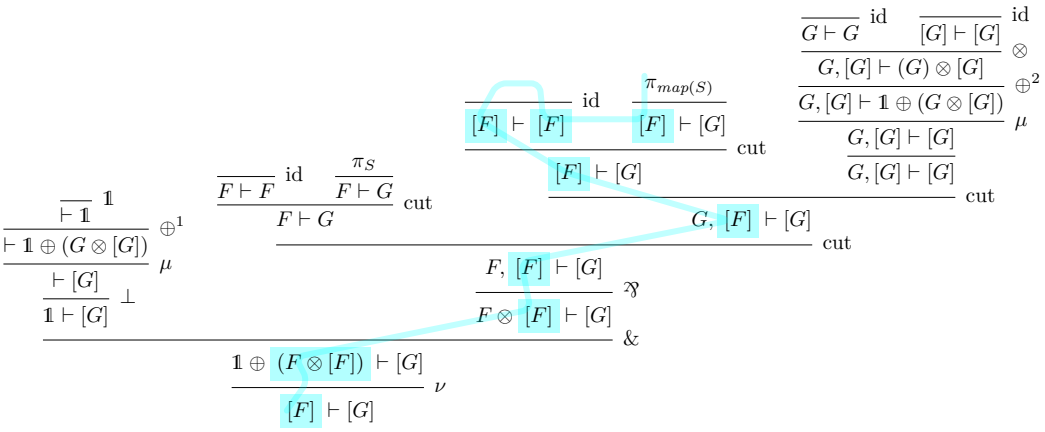
▶ **Example 30.** The translation $\llbracket circ(\omega, \emptyset, \alpha, \beta) \rrbracket$ of the iso ω from Example 9 is, with $F = A_{\alpha l}, G = B_{\alpha rl}, H = C_{\alpha rr}$ and $F' = A_{\beta rl}, G' = B_{\beta rr}, H' = C_{\beta l}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[];a : F \vdash F'}\ \mathrm{id}}{[];a : F \vdash F' \oplus G'}\ \oplus^1}{[];a : F \vdash H' \oplus (F' \oplus G')}\ \oplus^2}{[F];\emptyset \vdash H' \oplus (F' \oplus G')}\ \mathrm{ex(a)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{[];b : G \vdash G'}\ \mathrm{id}}{[];b : G \vdash F' \oplus G'}\ \oplus^2}{[];b : G \vdash H' \oplus (F' \oplus G')}\ \oplus^2}{[G];\emptyset \vdash H' \oplus (F' \oplus G')}\ \mathrm{ex(b)} \qquad \dfrac{\dfrac{\dfrac{}{[];c : H \vdash H'}\ \mathrm{id}}{[];c : H \vdash H' \oplus (F' \oplus G')}\ \oplus^1}{[H];\emptyset \vdash H' \oplus (F' \oplus G')}\ \mathrm{ex(c)}}{[G \oplus H];\emptyset \vdash H' \oplus (F' \oplus G')}\ \parr}{[F \oplus (G \oplus H)];\emptyset \vdash H' \oplus (F' \oplus G')}\ \parr$$

▶ **Example 31.** Considering the iso swap of type $A \otimes B \leftrightarrow B \otimes A$ and its μMALL proof

$$\pi_S = \dfrac{\dfrac{\dfrac{}{A_{\gamma l} \vdash A_{\gamma' r}}\ id \qquad \dfrac{}{B_{\gamma r} \vdash B_{\gamma' l}}\ id}{A_{\gamma l}, B_{\gamma r} \vdash (B \otimes A)_{\gamma'}}\ \otimes}{(A \otimes B)_\gamma \vdash (B \otimes A)_{\gamma'}}\ \parr$$

Following Example 10 we give its corresponding proof $\pi_{map(S)}$ where $F = (A \otimes B)_{\alpha irl}$ and $G = (B \otimes A)_{\beta irl}$, then $[F]$ and $[G]$ are respectively of address α and β:



We painted in blue the pre-thread that follows the *focus* of the structurally recursive criterion. During the *negative phase* which consists of the $\nu, \&, \parr, \bot$ rules the pre-thread is going up, at each time going into the subformula corresponding to the *focus*. Then, during the *positive phase* the pre-thread is not active during the multiple *cut* rules until it reaches the *id* rule, where the pre-thread bounces and starts going down before bouncing back up again in the *cut* rule, into the infinite branch, where the behavior of the pre-thread will repeat itself. One can then show that this pre-thread is indeed a thread, according to [2] and that it is valid: among the formulas it visits infinitely often, the minimal formula is $[F]$, which is a ν formula as since $[F]$ is on the left hand side of the sequent, we get $[F] = (\mu X.\mathbb{1} \oplus (F \otimes X))^{\perp}$.

▶ **Lemma 32.** *Given $\pi = circ(\omega)$, for each infinite branch of π, only a single iso-variable is visited infinitely often.*

Proof. Since we have at most one iso-variable, we never end up in the case that between an annotated sequent $\vdash^f$ and a back-edge pointing to f we encounter another annotated sequent. ◀

Among the terms that we translate, the translation of a value yields what we call a *Purely Positive Proof*: a finite derivation whose only rules have for active formula the sole formula on the right of the sequent. Any such derivation is trivially a valid pre-proof.

▶ **Definition 33** (Purely Positive Proof). *A Purely Positive Proof is a finite, cut-free proof whose rules are only $\oplus^i, \otimes, \mu, \mathbb{1}, id$ for $i \in \{1, 2\}$.*

▶ **Lemma 34** (Values are Purely Positive Proofs). *Given $x_1 : A^1, \ldots, x_n : A^n \vdash v : A$ then* $\dfrac{[\![v]\!]}{[]; x_1 : A^1_{\alpha_1}, \ldots, x_n : A^n_{\alpha_n} \vdash A_\alpha}$ *is a purely positive proof.*

We can then define the notion of *bouncing-cut* and their origin:

▶ **Definition 35** (Bouncing-Cut). *A Bouncing-cut is a cut of the form :* $\dfrac{\dfrac{\pi}{\Sigma \vdash G} \quad \dfrac{}{G \vdash F}\, be(f)}{\Sigma \vdash F}\, cut$

Due to the syntactical restrictions of the language we get the following:

▶ **Property 36** (Origin of Bouncing-Cut). *Given a well-typed iso, every occurrence of a rule $be(f)$ in $[\![circ(\omega)]\!]$ is a premise of a bouncing-cut.*

$$\frac{\mathtt{Neg}(\{(\mathtt{inj}_l\ v_j :: l_j, \xi_j)_{j\in J}\} \cup \{(\mathtt{inj}_r\ v_k :: l_k, \xi_k)_{k\in K}\})}{F_1 \oplus F_2 :: \Upsilon; \Theta \vdash G} =$$

$$\frac{\dfrac{\mathtt{Neg}(\{(v_j :: l_j, \xi_j)_{j\in J}\})}{F_1 :: \Upsilon; \Theta \vdash G} \qquad \dfrac{\mathtt{Neg}(\{(v_k :: l_k, \xi_k)_{k\in K}\})}{F_2 :: \Upsilon; \Theta \vdash G}}{F_1 \oplus F_2 :: \Upsilon; \Theta \vdash G}\ \&$$

$$\frac{\mathtt{Neg}(\{([], \xi)\})}{[]; \Theta \vdash G} = \frac{\mathtt{Pos}(\xi)}{[]; \Theta \vdash G} \qquad \frac{\mathtt{Neg}(\{((), l), \xi)\})}{\mathbb{1} :: \Upsilon; \Theta \vdash G} = \frac{\dfrac{\mathtt{Neg}(\{l, \xi\})}{\Upsilon; \Theta \vdash G}}{\mathbb{1} :: \Upsilon; \Theta \vdash G}\ \top$$

$$\frac{\mathtt{Neg}(\{(\langle v_i^1, v_i^2\rangle :: l_i, \xi_i)_{i\in I}\})}{F_1 \otimes F_2 :: \Upsilon; \Theta \vdash G} = \frac{\dfrac{\mathtt{Neg}(\{(v_i^1 :: v_i^2 :: l_i, \xi_i)_{i\in I}\})}{F_1, F_2 :: \Upsilon; \Theta \vdash G}}{F_1 \otimes F_2 :: \Upsilon; \Theta \vdash G}\ \parr$$

$$\frac{\mathtt{Neg}(\{(\mathtt{fold}\ v_i :: l_i, \xi_i)_{i\in I}\})}{\mu X.F :: \Upsilon; \Theta \vdash G} = \frac{\dfrac{\mathtt{Neg}(\{(v_i :: l_i, \xi_i)_{i\in I}\})}{F[X \leftarrow \mu X.F] :: \Upsilon; \Theta \vdash G}}{\mu X.F :: \Upsilon; \Theta \vdash G}\ \nu$$

$$\frac{\mathtt{Neg}(\{(x :: l, \xi)\})}{F :: \Upsilon; \Theta \vdash G} = \frac{\dfrac{\mathtt{Neg}(\{l, \xi\})}{\Upsilon; \Theta, x : F \vdash G}}{F :: \Upsilon; \Theta \vdash G}\ ex(x)$$

Figure 4 Negative Phase.

In particular, when following a thread going up into a *bouncing-cut*, it will always start from the left-hand-side of the sequent, before going back down on the right-hand-side of the sequent. It will also always bounce back up on the bouncing-cut to reach the back-edge.

▶ **Theorem 37** (*Validity of proofs*). *If* $\vdash_\omega \omega : A \leftrightarrow B$ *and* $\pi = [\![circ(\omega, \emptyset, \alpha, \beta)]\!]$ *then* π *satisfies* μ*MALL validity criterion from [2].*

Proof Sketch. In order to show the validity of our derivation we need, for each infinite branch, to build a valid thread. From the previous lemmas and the syntactical constraints of the language, we get that any infinite branch is completely defined by a single iso-variable, which allows us to reason entirely about a single recursive iso **fix** $f.\{v_1 \leftrightarrow e_1 \mid \ldots \mid v_n \leftrightarrow e_n\}$. For each infinite branch, we will build a pre-thread that follows the focus of the primitive recursive criterion. We know that the focus is a *strict subvariable* of the argument that is called recursively, as a consequence we can split the constructed thread into two parts, p_0 and p_1, corresponding respectively to the *negative phase* and the *positive phase*. We also know that, each argument of a recursive call gives us a *purely positive proof* which is made only of tensors. We can show that the size of p_0 is bigger than p_1 and also that p_1 is a prefix of p_0. This allows us to make sure that our pre-thread is a thread where the *visible part* always encounters a ν formula. Finally, the inductive type is decomposed in the negative phase and not in the positive phase (as the right-hand side of a recursive call is purely made of tensors), we can show that (i) the thread is never stationary and (ii) the thread has for minimal recurring formula that is visited infinitely often a ν formula, hence satisfying validity. ◀

$$\mathtt{Pos}\left(\ \frac{}{\vdash_e () : \mathbb{1}}\ \right) = \frac{}{[];\emptyset \vdash \mathbb{1}_\alpha}\ \mathbb{1}$$

$$\mathtt{Pos}\left(\ \frac{}{x : A \vdash_e x : A}\ \right) = \frac{}{[];x : A_\alpha \vdash A_\beta}\ id$$

$$\mathtt{Pos}\left(\ \frac{\xi}{\underline{\Theta} \vdash_e t : A_1 \quad / \quad \underline{\Theta} \vdash_e \mathtt{inj}_l\ t : A_1 \oplus A_2}\ \right) = \frac{\mathtt{Pos}(\xi)}{\Theta \vdash (A_1)_{\alpha l} \quad / \quad [];\Theta \vdash (A_1 \oplus A_2)_\alpha}\ \oplus^1$$

$$\mathtt{Pos}\left(\ \frac{\xi}{\underline{\Theta} \vdash_e t : A_2 \quad / \quad \underline{\Theta} \vdash_e \mathtt{inj}_r\ t : A_1 \oplus A_2}\ \right) = \frac{\mathtt{Pos}(\xi)}{[];\Theta \vdash (A_2)_{\alpha r} \quad / \quad [];\Theta \vdash (A_1 \oplus A_2)_\alpha}\ \oplus^2$$

$$\mathtt{Pos}\left(\ \frac{\xi_1 \qquad \xi_2}{\underline{\Theta_1} \vdash_e t_1 \quad \underline{\Theta_2} \vdash_e t_2 : A_2 \quad / \quad \underline{\Theta_1}, \underline{\Theta_2} \vdash_e \langle t_1, t_2 \rangle : A_1 \otimes A_2}\ \right) = \frac{\mathtt{Pos}(\xi_1) \qquad \mathtt{Pos}(\xi_2)}{[];\Theta_1 \vdash (A_1)_{\alpha l} \quad [];\Theta_2 \vdash (A_2)_{\alpha r} \quad / \quad [];\Theta_1, \Theta_2 \vdash (A_1 \otimes A_2)_\alpha}\ \otimes$$

$$\mathtt{Pos}\left(\ \frac{\xi}{\underline{\Theta} \vdash_e t : A[X \leftarrow \mu X.A] \quad / \quad \underline{\Theta} \vdash_e \mathtt{fold}\ t : \mu X.A}\ \right) = \frac{\mathtt{Pos}(\xi)}{[];\Theta \vdash (A[X \leftarrow \mu X.A])_{\alpha i} \quad / \quad [];\Theta \vdash (\mu X.A)_\alpha}\ \mu$$

$$\mathtt{Pos}\left(\ \frac{\xi_1 \qquad \xi_2}{\underline{\Theta_1} \vdash_e t_1 : A_1 \otimes \cdots \otimes A_n \quad \underline{\Theta_2}, x_1 : A_1, \ldots, x_n : A_n \vdash_e t_2 : B \quad / \quad \underline{\Theta_1}, \underline{\Theta_2} \vdash_e \mathtt{let}\,(x_i)_{i \in I} = t_1\ \mathtt{in}\ t_2 : B}\ \right) =$$

$$\frac{\mathtt{Pos}(\xi_1) \qquad \mathtt{Neg}([(x_i)_{i \in I}], \xi_2))}{[];\Theta_1 \vdash F_1 \otimes \cdots \otimes F_n \quad [F_1 \otimes \cdots \otimes F_n];\Theta_2 \vdash B_\alpha \quad / \quad [];\Theta_1, \Theta_2 \vdash B_\alpha}\ \mathtt{cut}$$

$$\mathtt{Pos}\left(\ \frac{\xi}{\Psi \vdash_\omega \omega : A \leftrightarrow B \quad \underline{\Theta} \vdash_e t : A \quad / \quad \underline{\Theta}; \Psi \vdash_e \omega\ t : B}\ \right) = \frac{\mathtt{Pos}(\xi) \qquad circ(\omega, \{f\}, \alpha, \beta)}{[];\Theta \vdash A \quad [A];\emptyset \vdash B_\beta \quad / \quad [];\Theta \vdash B_\beta}\ \mathtt{cut}$$

Figure 5 Positive Phase.

We can also show that the rewriting rules of the language simulate the cut-elimination procedure, as it is described in [2]:

▶ **Theorem 38** (Simulation). *Provided an iso* $\vdash_\omega \omega : A \leftrightarrow B$ *and values* $\vdash_e v : A$ *and* $\vdash_e v' : B$*, let* $\pi = \mathtt{Pos}(\omega\ v)$ *and* $\pi' = \mathtt{Pos}(v)$*, if* $\omega\ v \to^* v'$ *then* $\pi \to^*_{cut\text{-}elim} \pi'$*.*

Proof sketch. The proof relies on the definition of a novel explicit substitution rewriting system for the language, called $\to_{e\beta}$. Explicit substitution are represented as a series of *let* constructs where the substitution of a variable by a value only occurs when we reach the term $\mathtt{let}\, x = v\ \mathtt{in}\ x$. Each rewriting step of this system represents exactly one step of the cut-elimination procedure of μMALL. Then we only need to show that the explicit substitution rewriting system matches, meaning that if $\sigma = \{\overrightarrow{x} \mapsto \overrightarrow{v}\}$ then $\mathtt{let}\ \overrightarrow{x} = \overrightarrow{v}\ \mathtt{in}\ e \to^*_{e\beta} \sigma(e)$. ◀

This leads to the following corollary:

▶ **Corollary 39** (Isomorphism of proofs.). *Given a well-typed iso* $\vdash_\omega \omega : A \leftrightarrow B$ *and two well-typed close value* v_1 *of type* A *and* v_2 *of type* B *and the proofs* $\pi : F_1 \vdash G_1$*,* $\pi^\perp : G_2 \vdash F_1$*,* $\phi : F_3$*,* $\psi : G_2$ *corresponding respectively to the translation of* $\omega, \omega^\perp, v_1, v_2$ *then:*

$$\frac{\phi}{\vdash F_3} \rightsquigarrow \frac{\dfrac{\dfrac{\phi}{\vdash F_3} \quad \dfrac{\pi}{F_1 \vdash G_1}\ cut}{\vdash G_1} \qquad \dfrac{\pi^\perp}{G_2 \vdash F_2}\ \mathtt{cut}}{\vdash F_2} \qquad \frac{\dfrac{\dfrac{\psi}{\vdash G_3} \quad \dfrac{\pi^\perp}{F_2 \vdash G_2}\ cut}{\vdash F_2} \qquad \dfrac{\pi}{F_1 \vdash G_1}\ \mathtt{cut}}{\vdash G_1} \rightsquigarrow \frac{\psi}{\vdash G_3}$$

5 Conclusion

Summary of the contribution. We presented a linear, reversible language with inductive types. We showed how ensuring non-overlapping and exhaustivity is enough to ensure the reversibility of the isos. The language comes with both an expressivity result that shows that any Primitive Recursive Functions can be encoded in this language as well as an interpretation of programs into μMALL proofs. The latter result rests on the fact that our isos are *structurally recursive*.

Future works. We showned a one-way encoding from isos to proofs of μMALL, it is clear that there exists proof-isomorphisms of μMALL that does not correspond to an iso of our language, for instance taking the reversible map function on streams. Therefore, a first extension to our work would be to consider a two-way encoding and adding coinductive types in the language. This would require relaxing the condition on recursive isos, as termination would be no longer possible to ensure. This is a focus of our forthcoming research.

A second direction for future work is to consider quantum computation, by extending our language with linear combinations of terms. We plan to study purely quantum recursive types and generalised quantum loops: in [20], lists are the only recursive type which is captured and recursion is terminating. The logic μMALL would help in providing a finer understanding of termination and non-termination

References

1 David Baelde. Least and greatest fixed points in linear logic. *ACM Transactions on Computational Logic (TOCL)*, 13(1):1–44, 2012.

2 David Baelde, Amina Doumane, Denis Kuperberg, and Alexis Saurin. *Bouncing Threads for Circular and Non-Wellfounded Proofs: Towards Compositionality with Circular Proofs*, pages 1–13. Association for Computing Machinery, New York, NY, USA, 2022. doi:10.1145/3531130.3533375.

3 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)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2016.42.

4 Henk Barendregt. The lambda calculus: its syntax and semantics. *Studies in logic and the foundations of Mathematics*, 1984.

5 Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

6 Kostia Chardonnet, Louis Lemonnier, and Benoît Valiron. Categorical semantics of reversible pattern-matching. In Ana Sokolova, editor, Proceedings 37th Conference on *Mathematical Foundations of Programming Semantics,* Hybrid: Salzburg, Austria and Online, 30th August - 2nd September, 2021, volume 351 of *Electronic Proceedings in Theoretical Computer Science*, pages 18–33. Open Publishing Association, 2021. doi:10.4204/EPTCS.351.2.

7 Haskell B Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences*, 20(11):584–590, 1934.

8 Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of theoretical physics*, 21(3):219–253, 1982.

9 Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C Pierce. Linear dependent types for differential privacy. In *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 357–370, 2013.

10 William A Howard. The formulae-as-types notion of construction. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, 44:479–490, 1980.

11 Rosham P. James and Amr Sabry. Theseus: A high-level language for reversible computing. Draft, available at `https://legacy.cs.indiana.edu/~sabry/papers/theseus.pdf`, 2014.

12 Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. Rustbelt: Securing the foundations of the rust programming language. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–34, 2017.

13 Robin Kaarsgaard and Mathys Rennela. Join inverse rig categories for reversible functional programming, and beyond, 2021. Draft, available at `arXiv:2105.09929`.

14 Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

15 Kenji Maillard, Cătălin Hriţcu, Exequiel Rivas, and Antoine Van Muylder. The next 700 relational program logics. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–33, 2019.

16 Kenichi Morita and Yoshikazu Yamaguchi. A universal reversible turing machine. In *International Conference on Machines, Computations, and Universality*, pages 90–98. Springer, 2007.

17 Luca Paolini, Mauro Piccolo, and Luca Roversi. A class of recursive permutations which is primitive recursive complete. *Theoretical Computer Science*, 813:218–233, 2020. URL: `https://www.sciencedirect.com/science/article/pii/S0304397519307558`.

18 John C Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55–74. IEEE, 2002.

19 Hartley Rogers Jr. *Theory of recursive functions and effective computability*. MIT press, 1987.

20 Amr Sabry, Benoit Valiron, and Juliana Kaizer Vizzotto. From symmetric pattern-matching to quantum control. In Christel Baier and Ugo Dal Lago, editors, *Proceedings of the 21st International Conference on Foundations of Software Science and Computation Structures (FOSSACS'18)*, volume 10803 of *Lecture Notes in Computer Science*, pages 348–364, Thessaloniki, Greece, 2018. Springer. `doi:10.1007/978-3-319-89366-2_19`.

21 Nikhil Swamy, Cătălin Hriţcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, et al. Dependent types and multi-monadic effects in f. In *Proceedings of the 43rd annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 256–270, 2016.

Measure-Theoretic Semantics for Quantitative Parity Automata

Corina Cîrstea
University of Southampton, UK

Clemens Kupke
University of Strathclyde, UK

Abstract

Quantitative parity automata (QPAs) generalise non-deterministic parity automata (NPAs) by adding weights from a certain semiring to transitions. QPAs run on infinite word/tree-like structures, modelled as coalgebras of a polynomial functor F. They can also arise as certain products between a quantitative model (with branching modelled via the same semiring of quantities, and linear behaviour described by the functor F) and an NPA (modelling a qualitative property of F-coalgebras). We build on recent work on semiring-valued measures to define a way to measure the set of paths through a quantitative branching model which satisfy a qualitative property (captured by an unambiguous NPA running on F-coalgebras). Our main result shows that the notion of *extent* of a QPA (which generalises non-emptiness of an NPA, and is defined as the solution of a nested system of equations) provides an equivalent characterisation of the measure of the accepting paths through the QPA. This result makes recently-developed methods for computing nested fixpoints available for model checking qualitative, linear-time properties against quantitative branching models.

2012 ACM Subject Classification Theory of computation → Program verification

Keywords and phrases parity automaton, coalgebra, measure theory

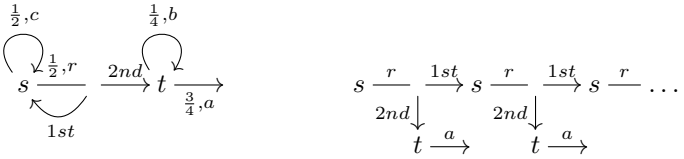
Digital Object Identifier 10.4230/LIPIcs.CSL.2023.14

Funding Research carried out as part of the Leverhulme Trust Research Project Grant RPG-2020-232.

1 Introduction

When model checking linear-time properties over non-deterministic or probabilistic models, the standard approach is to formalise the property in question as an automaton running over infinite words, and to consider the product of this automaton with the model, in order to answer the questions: *Does there exist a path through the model which conforms to a property automaton?* and *What is the probability of exhibiting a path which conforms to an automaton?* (see e.g. [1][Sections 4.6 and 28.6], [2]). Generalising this approach, we consider state-based system models whose transitions carry weights from a partial semiring. Instances of such systems include non-deterministic systems (with weights from the boolean semiring), probabilistic systems (with weights from the probabilistic semiring), and resource-aware systems (with weights from the tropical semiring). Thus, our work can also answer the following question, using similar automata-based techniques: *What is the minimal amount of resources needed to exhibit a path which conforms to a property automaton?*

In addition to a more general notion of branching, our models also allow a more general notion of path: whereas in existing approaches paths are sequences (of states and transition labels), with each transition resulting in a *single* successor state, here individual transitions can have *finitely-many* successor states, and thus paths can be tree-shaped. This allows us to model systems with *dynamic structure*, as illustrated by the following example:



The model above (left), with initial state s, has standard transitions (labels b, c) which result in a *single* successor state, but also transitions resulting in *two* successors states (label r), or *zero* successor states (label a). One can view this as modelling a probabilistic server which accepts requests (r transition) or carries out other work (c transition), both with probability $\frac{1}{2}$. Following a request, a *new* process is created to deal with the request (state t), and the server itself continues in state s. To model this behaviour, the r transition has *two* successor states; these are ordered, as indicated by the labels on the arrows leading to them. Then, an a-transition models successfully answering a request, while a b-transition models doing other work instead. A possible execution of this system, where the server repeatedly accepts new requests and the newly created processes immediately answer them, is pictured above (right).

We use automata over infinite words (similarly to existing work [1, 2]) but also over infinite trees (given that paths can be tree-shaped), to formalise correctness properties of system executions. Such properties have a *qualitative* interpretation over paths, but also a quantitative interpretation over states in our models. For instance, in the previous example, one might want to formalise (and verify!) the property that every server request is eventually answered. While existing approaches typically use Büchi/Rabin automata to describe ω-regular properties of infinite words [1, 2], here we choose the related formalism of *parity automata* for several reasons: (i) it is as expressive as Büchi/Rabin automata over infinite words, (ii) unlike Büchi automata, they have the full expressive power needed to capture all regular languages of infinite trees [10, 7], and (iii) their acceptance conditions can be described using the solutions of nested systems of equations.

In order to uniformly treat a variety of branching types (with transition weights taken from a semiring) and transition types (linear- or tree-shaped, or a combination), we model systems as *coalgebras*; their type incorporates branching behaviour (described by a monad) and linear behaviour (described by a polynomial endofunctor). We model system executions also as coalgebras (with no branching), and as a result our automata operate on coalgebras.

The question we are concerned with is: *Given a quantitative branching model and a qualitative property of paths, with the latter formalised as a parity automaton, what is the* degree *(e.g. probability/cost) with which the property holds in the quantitative model?* We answer this question in two ways: one which is measure-theoretic and naturally captures the intuition that we are *measuring*, in some generalised sense, the accepting runs of a quantitative automaton (building on results in [4] on semiring-valued measures); and another which is more amenable to computation (using the notion of *extent* from [6]). After defining these two ways of measuring the set of accepting runs of a QPA, our main result establishes their equivalence. The implications of this result are two-fold. On the one hand, the result formally confirms that the notion of extent defined in [6] achieves its intended purpose in key example semirings: it measures the existence of an accepting path in the non-deterministic case; the probability of exhibiting an accepting path in the probabilistic case (and thus instantiates to known results in this case); and the minimal cost required to exhibit an accepting path, in the resource-aware case. On the other hand, since the latter characterisation is in terms of the solution of a nested system of equations, methods for computing such solutions (including those recently developed in [11, 3, 12]) become available for model checking qualitative, linear-time properties against quantitative branching models. In the last part of the paper, we show how the standard automata-based approach to model checking linear-time properties

over non-deterministic and probabilistic models [1, 2] generalises to quantitative branching models. We defer computational aspects to future work, as this requires adapting techniques in [3] to our more general notion of system of equations.

At the heart of our main result is a characterisation, due to [15], of the accepting paths of a parity automaton as the solution of a nested system of equations. This allows us to relate, via a semiring-valued measure, the set of accepting paths of a QPA and its extent (also defined as the solution of a system of equations). The proof of this result is non-trivial, partly because semiring-valued measures are not well-behaved w.r.t. intersections.

The paper is structured as follows: Section 2 introduces relevant concepts, including systems of equations and their solutions, qualitative and quantitative parity automata, and semiring-valued measures. Section 3 shows the equivalence of two approaches to measuring accepting runs: via semiring-valued measures and via extents. Next, Section 4 shows how this result can be used to model-check qualitative, linear-time properties against quantitative branching models. Section 5 summarises our contributions and outlines future work.

Related Work. [4] considers quantitative, linear-time fixpoint logics interpreted over the same type of quantitative branching models. Semiring-valued measures are introduced in op. cit., and used to provide a measure-theoretic semantics for these logics. This is then proved equivalent to the original semantics for the logics. However, these logics suffer from limited expressiveness on tree-shaped linear behaviours (they cannot express conjunctions and arbitrary disjunctions). Here we address this limitation, while also taking a more fundamental approach to formalising linear-time properties, namely as automata. Beyond the increased generality, a key difference compared to [4] is that our proofs now exploit a characterisation of the accepting paths of a QPA as the solution of a nested system of equations. Thus, by working at the level of automata, the link between the extent-based semantics and the measure-theoretic semantics becomes conceptually clearer. As added benefit, the move to automata connects our work to existing algorithmic approaches for solving nested systems of equations, thereby paving the way for applications in model checking.

Quantitative verification of weighted systems has been considered in a number of other works, including [8, 9, 14]. Our approach differs from these in that we restrict to *qualitative* properties of paths through a quantitative branching model, and we measure to what degree these hold in such models. One immediate drawback of the increased generality in [8, 9] is that the meaning of quantitative formulas is conceptually less clear, and is defined separately for each model type (namely quantitative transition systems and quantitative Markov chains). The same holds for the model checking algorithms, which are tailored to the underlying semantic model and not generic. In contrast, our quantitative notion of acceptance has an intuitive measure-theoretic description, and our model checking approach (computation of nested extents) is parameterised by the semiring used to model weighted branching.

2 Background

2.1 Nested Systems of Equations

▶ **Definition 1.** *Let $L_0, \ldots, L_n$ be complete lattices. A* nested system of equations *E has the form*

$$\begin{bmatrix} x_0 & =_\nu & f_0(x_0, \ldots, x_n) \\ x_1 & =_\mu & f_1(x_0, \ldots, x_n) \\ \vdots & & \\ x_n & =_\eta & f_n(x_0, \ldots, x_n) \end{bmatrix} \tag{1}$$

where η is either μ, if n is odd, or ν, if n is even, and where for $i \in \{0,\dots,n\}$, $f_i : L_0 \times \dots \times L_n \to L_i$ is a monotone function and the variable x_i takes values in the lattice L_i.

For $u_i \in L_i$, we write $E[x_i := u_i]$ for the system of $n-1$ equations obtained by removing the ith equation and substituting x_i by u_i in the remaining equations. We write η_i for either ν or μ, depending on whether i is even or odd. The *solution* of a system of equations is defined similarly to [11, 3].

▶ **Definition 2.** *The* solution $\mathsf{sol}(E)$ *of the nested system of equations E in (1) is defined by induction on the number of equations:*

$$\mathsf{sol}() = ()$$
$$\mathsf{sol}(E) = (\mathsf{sol}(E[x_n := v_n]), v_n), \text{ where } v_n = \eta_n(\lambda x.f_n(\mathsf{sol}(E[x_n := x]), x))$$

In other words, to solve a nested system of equations with variables $x_0,\dots,x_n$, the system of equations $E[x_n := x]$ is solved by viewing x as a parameter, its solution is substituted in the nth equation, and this equation is then solved to obtain the nth component v_n of the solution of E. The value v_n is finally substituted in the parameterised solutions for $E[x_n := x]$ to obtain solutions for the remaining variables. When solving the ith equation, the greatest, respectively least solution is taken, depending on whether i is even or odd. Given the system of equations in (1), $i \in \{0,\dots,n\}$ and values $v_k \in L_k$ for $k \in \{i+1,\dots,n\}$, we write $f_i^{v_{i+1},\dots,v_n} : L_i \to L_i$ for the map $x \mapsto f_i(\mathsf{sol}(E[x_i := x, x_{i+1} := v_{i+1},\dots,x_n := v_n]), x, v_{i+1},\dots,v_n)$.

Sufficient conditions for the existence and uniqueness of the individual fixpoints required in the definition of $\mathsf{sol}(E)$ are provided by Kleene's fixpoint theorem.

▶ **Theorem 3** (Kleene). *Let $\mathsf{Op} : (L,\sqsubseteq) \to (L,\sqsubseteq)$ be a monotone function on a complete lattice. The (transfinite) ascending chain $\mathsf{Op}^\beta(\bot)$, with β ranging over ordinals, is defined by: $\mathsf{Op}^0(\bot) = \bot$, $\mathsf{Op}^{\alpha+1}(\bot) = \mathsf{Op}(\mathsf{Op}^\alpha(\bot))$ for any ordinal α, and $\mathsf{Op}^\alpha(\bot) = \sqcup_{\beta<\alpha}\mathsf{Op}^\beta(\bot)$ for any limit ordinal α. Then, the least fixpoint of Op is $\mathsf{Op}^\gamma(\bot)$ for some ordinal γ. The greatest fixpoint of Op is characterised dually, via the (transfinite) descending chain $\mathsf{Op}^\beta(\top)$.*

▶ Remark 4. Thm. 3 implies that $\eta_i(f_i^{v_{i+1},\dots,v_n}) \sqsubseteq \eta_i(f_i^{v'_{i+1},\dots,v'_n})$ if $v_{i+1} \sqsubseteq v'_{i+1},\dots,v_n \sqsubseteq v'_n$.

2.2 Monads Weighted in Partial Semirings

▶ **Definition 5.** *A* partial commutative monoid *(p.c.m.) $(S,+,0)$ is given by a set S together with a partial operation $+ : S \times S \to S$ and an element $0 \in S$, such that:*

- *$s+0$ is defined for all $s \in S$ and moreover, $s+0 = s$,*
- *$(s+t)+u$ is defined if and only if $s+(t+u)$ is defined, and in that case $(s+t)+u = s+(t+u)$,*
- *whenever $s+t$ is defined, so is $t+s$ and moreover, $s+t = t+s$.*

A partial commutative semiring *is a tuple $S := (S,+,0,\bullet,1)$ with $(S,+,0)$ a p.c.m. and $(S,\bullet,1)$ a commutative monoid, with $\bullet$ distributing over sums; that is, for all $s,t,u \in S$, $s\bullet 0 = 0$, and whenever $t+u$ is defined, then so is $s\bullet t + s\bullet u$ and moreover, $s\bullet t + s\bullet u = s\bullet(t+u)$.*

The addition operation of any partial commutative semiring induces a pre-order $\sqsubseteq$ on S:

$$x \sqsubseteq y \quad \text{if and only if} \quad \text{there exists } z \in S \text{ such that } x+z = y \tag{2}$$

for $x,y \in S$. It then follows from the axioms of a partial commutative semiring that $0 \sqsubseteq s$ for all $s \in S$, and that $\sqsubseteq$ is preserved by $+$ and $\bullet$ in each argument (see [5] for details).

▶ **Assumption 6.** *Similarly to [4], we make the following assumptions:*

- $(S, \sqsubseteq)$ *is a complete lattice and has the unit* 1 *of* $\bullet$ *as top element;*
- $+$ *preserves joins of increasing countable chains and meets of decreasing countable chains, in each argument;*
- $\bullet$ *preserves both suprema and infima in each argument; moreover, the following holds for all* $A_i \subseteq S$ *with* $i \in \omega$*, whenever* $\sum_{i\in\omega} \inf A_i$ *is defined:*

$$\sum_{i\in\omega} \inf A_i = \inf \{ \sum_{i\in\omega} a_i \mid a_i \in A_i \text{ for } i \in \omega, \sum_{i\in\omega} a_i \text{ is defined} \} \tag{3}$$

The countable (partial) addition operation used in the last condition is defined by $\sum_{i\in\omega} s_i := \sup_{n\in\omega}(s_0 + \ldots + s_n)$. If S is partial, this countable sum is defined iff all sums $s_0 + \ldots + s_n$ with $n \in \omega$ are defined. This definition exploits the fact that $s \sqsubseteq s + t$ for any $s, t \in S$ for which $s + t$ is defined, together with the existence of joins of increasing countable chains.

▶ **Example 7.** As concrete semirings we consider the *boolean semiring* $(\{0,1\}, \vee, 0, \wedge, 1)$, the partial *probabilistic semiring* $([0,1], +, 0, *, 1)$, the *tropical semiring* $\mathbb{N}^\infty = (\mathbb{N}^\infty, \min, \infty, +, 0)$ (with $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$) and its bounded variants $S_B = ([0, B] \cup \{\infty\}, \min, \infty, +_B, 0)$ with $B \in \mathbb{N}$, where for $m, n \in [0, B] \cup \{\infty\}$ we have

$$m +_B n = \begin{cases} m + n, & \text{if } m + n \leq B \\ \infty, & \text{otherwise} \end{cases} .$$

The associated orders are $\leq$ on $\{0,1\}$ and $[0,1]$, and $\geq$ on $\mathbb{N}^\infty$ and $[0, B] \cup \{\infty\}$. As shown in [4], all these orders satisfy Assumption 6. Note that we allow the semiring $(S, +, 0, \bullet, 1)$ to be partial in order to also cover probabilistic branching.

▶ Remark 8. When the semiring $(S, +, 0, \bullet, 1)$ is partial, we will also consider the total semiring $(S', \oplus, 0, \bullet, 1)$, where $S' = S$ and $\oplus$ is given by

$$s \oplus t = \begin{cases} s + t, & \text{if } s + t \text{ is defined} \\ 1, & \text{otherwise} \end{cases} .$$

It is easy to check that this semiring satisfies Assumption 6 whenever $(S, +, 0, \bullet, 1)$ does. In particular, the induced order is not changed when moving from S to S'.

▶ **Example 9.** The total semiring $([0,1], \oplus, 0, *, 1)$ associated to the probabilistic semiring has $\oplus : [0,1] \times [0,1] \to [0,1]$ given by addition truncated above at 1.

We use monads weighted in partial semirings to model systems with weighted branching. For a partial semiring satisfying Assumption 6, the monad $(\mathsf{T}_S, \eta, \sqcup)$ is given by

$$\mathsf{T}_S(X) = \{ \varphi : X \to S \mid \mathsf{supp}(\varphi) \text{ is finite}, \sum_{x\in\mathsf{supp}(\varphi)} \varphi(x) \text{ is defined} \},$$

$$\eta_X : X \to \mathsf{T}_S X,\ \eta_X(x)(y) = \begin{cases} 1 & \text{if } y = x \\ 0 & \text{otherwise} \end{cases},$$

$$\sqcup_X : \mathsf{T}_S(\mathsf{T}_S X) \to \mathsf{T}_S X,\ \sqcup_X(\Phi)(x) = \sum_{\varphi\in\mathsf{supp}(\Phi)} \Phi(\varphi) \bullet \varphi(x) \text{ for } \Phi \in \mathsf{T}_S(\mathsf{T}_S X) \subseteq S^{(S^X)}$$

where $\mathsf{supp}(\varphi) = \{x \in X \mid \varphi(x) \neq 0\}$ is the *support* of φ. For a function $f : X \to Y$ we put

$$\mathsf{T}_S(f)(\sum_{i\in I} c_i x_i) = \sum_{i\in I} c_i f(x_i)$$

where we use the formal sum notation $\sum_{i\in I} c_i x_i$, with I finite, to denote the element of $\mathsf{T}_S(X)$ mapping $x \in X$ to $(\sum_{j\in J_x} c_j) \in S$ with $J_x = \{i \mid x_i = x\}$, and all $x \notin \{x_i \mid i \in I\}$ to $0 \in S$. Our choice of notation for the monad multiplication avoids unnecessary overloading of the symbol μ, which we use to denote both a least fixpoint and a measure.

2.3 Coalgebras with Branching and their Linear Behaviour

Recall that a *coalgebra for a functor* G (cf. [13]) is a pair (C, γ) with C a set of states and $\gamma : C \to GC$ a *transition map*. A *pointed coalgebra* is a tuple (C, γ, c) with (C, γ) a coalgebra and $c \in C$ a designated state.

We use polynomial functors $F : \mathsf{Set} \to \mathsf{Set}$ of the form $FX = \coprod_{i\in I} X^{j_i}$, with $j_i \in \omega$ for $i \in I$, to describe the structure of individual transitions in a system with branching. We view I as a set of transition *labels*, with j_i the *arity* of transitions labelled by i. Our chosen shape for F allows transitions with finitely-many successors. For $i \in I$, we write $\iota_i : \mathsf{Id}^{j_i} \Rightarrow F$ (with $\mathsf{Id} : \mathsf{Set} \to \mathsf{Set}$ the identity functor) for the canonical injection.

We model quantitative branching systems as *pointed* $(\mathsf{T}_S \circ F)$*-coalgebras*, with $(S, +, 0, \bullet, 1)$ as before and F as above. Such coalgebras have weighted transitions $c \xrightarrow{w,i} (c_1, \ldots, c_{j_i})$ with $w \in S$ the transition weight, $i \in I$ the transition label, and $c_1, \ldots, c_{j_i}$ the successor states. In spite of this potential branching *within* individual transitions, we view the functor F as defining a general notion of *linear* behaviour. (The word *linear* here refers to time!) The elements of the final F-coalgebra thus provide a natural notion of maximal (potentially infinite) trace for our models. The *branching* in our systems is modelled via the monad T_S. Our models thus distinguish between *deadlock* (captured by states with no outgoing transitions) and *successful termination* (captured by transitions labelled by $i \in I$ with $j_i = 0$).

▶ **Example 10.** Our model in Section 1 can be viewed as a $(\mathsf{T}_S \circ F)$-coalgebra, with S the probabilistic semiring and $F : \mathsf{Set} \to \mathsf{Set}$ given by $FX = (\{r\} \times X \times X) + (\{b, c\} \times X) + \{a\} \simeq (X \times X) + X + X + 1$. Thus, r-transitions have two successors, b/c-transitions have a single successor, and a-transitions are terminating. In tis case, maximal traces (elements of the final F-coalgebra) can be presented as infinite trees whose nodes are labelled by transitions and have 2, 1 or 0 children, depending on whether they are labelled by r, b/c or a.

Notions of *path* and *path fragment* through a coalgebra with branching are defined below. Informally, a path from a state selects a single transition out of the transitions from that state which have non-zero weight, and continues making similar choices from all successor states of the chosen transition. Thus, a path will typically contain an infinite number of transitions (unless it is terminating). Since paths record the states visited and the transitions taken, they formally correspond to elements of the *final* coalgebra for the functor $C \times F$. Path fragments are similar, except that they contain a finite number of transitions. Technically this means that path fragments correspond to elements of an initial algebra. In order to streamline our presentation we will work with concrete representations of paths and path fragments using trees. We will not formally define trees, but fix some useful notation.

▶ **Notation 11.** *We write $\xi = c(i(\xi_1, \ldots, \xi_{j_i}))$ for the $C \times I$-labelled ranked tree whose root is labelled with $(c, i) \in C \times I$ and whose immediate subtrees are the trees $\xi_1, \ldots, \xi_{j_i}$ where j_i is the arity of the transition label i. Furthermore we write $\xi \rightsquigarrow \xi'$ if $\xi' = \xi_j$ for some $j \in \{1, \ldots, j_i\}$, i.e., $\rightsquigarrow$ denotes the immediate subtree relation.*

▶ **Definition 12.** *Given a set C, a (C-)path is a $C \times I$-labelled ranked tree. The collection of all C-paths will be denoted by Z_C. Let (C, γ) be a $(\mathsf{T}_S \circ F)$-coalgebra. A path $\xi \in Z_C$ is a* path from $c \in C$ in (C, γ) *if ξ has the form $\xi = c(i(\xi_1, \ldots, \xi_{j_i}))$ where for $k \in \{1, \ldots, j_i\}$ we have that ξ_k is a path from some $c_k \in C$ in (C, γ) and where $\gamma(c)(\iota_i(c_1, \ldots, c_{j_i})) \neq 0$.*

To also define path fragments (to be thought of as partial paths, necessarily of finite depth) as labelled trees, we use an additional label $* \notin I$, which we formally treat as a new transition label with arity 0, although its purpose is to indicate the "ends" of a path fragment.

▶ **Definition 13.** *Let (C, γ) be a $(\mathsf{T}_S \circ F)$-coalgebra. A* path fragment from $c \in C$ in (C, γ) *is a $C \times (I \cup \{*\})$-labelled tree $\tau = c(i(\tau_1, \ldots, \tau_{j_i}))$, such that only the leaves of τ can be labelled by $*$, and where for all $k \in \{1, \ldots, j_i\}$ we have that τ_k is a path fragment from $c_k \in C$ in (C, γ) with $\gamma(c)(\iota_i(c_1, \ldots, c_{j_i})) \neq 0$. Given a path fragment q, we will refer to the leaves of τ the form $c(*)$ as* holes.

Equivalently, $c(*)$ is a path fragment from c, and if τ_k is a path fragment from $c_k \in C$ for all $k \in \{1, \ldots, j_i\}$ and $\gamma(c)(\iota_i(c_1, \ldots, c_{j_i})) \neq 0$, then $c(i(\tau_1, \ldots, \tau_{j_i}))$ is a path fragment from c.

▶ **Definition 14.** *A path fragment τ is a* prefix *of a path ξ if ξ is obtained by replacing each leaf of τ of the form $c(*)$ by a path from c. We write $\mathsf{pref}(\xi)$ for the set of prefixes of ξ.*

The set of all paths from $c \in C$ in (C, γ) is denoted Paths_c^γ (or simply Paths_c when γ is clear from the context). For a path fragment τ with holes $c_1(*), \ldots, c_n(*)$, and sets of paths $A_i \subseteq \mathsf{Paths}_{c_i}$ for $i \in \{1, \ldots, n\}$, the set of paths $\tau[A_1/c_1, \ldots, A_n/c_n]$ consists of all paths from c obtained by continuing τ with a path in A_i from each hole $c_i(*)$, for $i \in \{1, \ldots, n\}$.

▶ Remark 15. Our definitions of paths and a path fragments are equivalent to those in [4], where paths (respectively path fragments) are defined as elements of the final $C \times F$-coalgebra (Z_C, ζ_C) (the initial $C \times (\{*\} + F)$-algebra (Φ_C, α_C)). In this representation we have

$$\zeta_c(\xi) = (c, \iota_i(\xi_1, \ldots, \xi_{j_i})) \quad \text{if } \xi = c(i(\xi_1, \ldots, \xi_{j_i})).$$

In what follows, we will use the two definitions interchangeably.

▶ **Example 16.** Below are two paths from s in the $(\mathsf{T}_S \circ F)$-coalgebra from Example 10, depicted as labelled trees:

$$s \xrightarrow{c} s \xrightarrow{c} \ldots \qquad s \xrightarrow{r} \xrightarrow{1st} s \xrightarrow{r} \xrightarrow{1st} s \xrightarrow{r} \ldots \quad \begin{array}{l} \downarrow{\scriptstyle 2nd} \\ t \xrightarrow{a} \end{array} \quad \begin{array}{l} \downarrow{\scriptstyle 2nd} \\ t \xrightarrow{a} \end{array}$$

The second path models an execution where requests arrive at each step and are successfully answered in the next step. The path ξ is of the form $\xi = s(r(\xi, \xi'))$ with $\xi' = t(a())$.

A key notion for the semantics of parity automata is that of an *accepting* path. In our setting, where paths are tree-shaped, a path is accepting if all infinite traces through the path satisfy the parity condition. This is formalised in the next definition.

▶ **Definition 17.** *Let C be a set and let $\Omega : C \to \omega$ be a parity function with finite range. Given a path $\xi \in Z_C$ we call an infinite sequence $\xi_1\xi_2\xi_3 \cdots \in (Z_C)^\omega$ a* trace *through ξ if $\xi = \xi_1$ and for all $i \in \mathbb{N}$ we have $\xi_i \rightsquigarrow \xi_{i+1}$. We call a trace $\xi_1\xi_2\xi_3 \cdots \in (Z_C)^\omega$* good *if the maximal parity that occurs infinitely often in $\Omega(\pi_1(\xi_1))\ \Omega(\pi_1(\xi_2))\ \Omega(\pi_1(\xi_3))\ \ldots$ is even. A path $\xi \in Z_C$ is said to be* accepting *if all traces through ξ are good.*

▶ **Example 18.** Consider again the coalgebra of Example 10, and let $\Omega(s) = 0$ and $\Omega(t) = 1$. Then, both paths in Example 16 are accepting. On the other hand, the path $\xi_1 \in Z_C$ given by $\xi_1 = s(r(\xi_2, \xi_1'))$ with $\xi_2 = s(r(\xi_1, \xi_2'))$, $\xi_1' = t(b(\xi_1'))$ and $\xi_2' = t(a())$ is not accepting, since e.g. the trace $\xi_1\xi_1'\xi_1' \ldots$ is not good. Its corresponding labelled tree is given below:

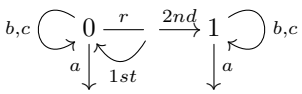
$$s \xrightarrow{r} \xrightarrow{1st} s \xrightarrow{\quad r \quad} \xrightarrow{1st} s \xrightarrow{r} \ldots \quad \begin{array}{l} \downarrow{\scriptstyle 2nd} \\ t \xrightarrow{b} t \xrightarrow{b} \ldots \end{array} \quad \begin{array}{l} \downarrow{\scriptstyle 2nd} \\ t \xrightarrow{a} \end{array}$$

2.4 Qualitative Parity Automata

We use *non-deterministic parity F-automata* to describe qualitative properties of paths.

▶ **Definition 19.** *A* non-deterministic parity F-automaton (NPA) (A, α, a_I, Ω) *is given by a pointed $\mathcal{P}_f \circ F$-coalgebra (A, α, a_I) (with $\mathcal{P}_f : \mathsf{Set} \to \mathsf{Set}$ the finite powerset functor) together with a function $\Omega : A \to \omega$ with finite range, called a* parity map.

▶ **Example 20.** Let $F : \mathsf{Set} \to \mathsf{Set}$ be as in Example 10. The following NPA, with initial state 0 and state parities identical to the state names, captures the property that each request initiates a *simple* process (second successor of the r-transition) which eventually answers the request. Here, a *simple* process is one whose behaviour does not involve any r-transitions. This constraint is captured by not allowing r-transitions from state 1 of the automaton.



The choice of parities ensures that no infinite sequence of b and c transitions is allowed from the second successor of any r-transition, on any accepting run (see below) of this automaton.

A *run* of an NPA on a pointed F-coalgebra records the coalgebra states which the automaton reads, the automaton states visited and the transitions taken.

▶ **Definition 21.** *A* run *of an NPA (A, α, a_I, Ω) on a pointed F-coalgebra (B, β, b_I) is a path $\xi \in Z_{B\times A}$ of the form $\xi = (b_I, a_I)(i(\xi_1, \dots, \xi_{j_i}))$ such that for each $\xi' \in Z_{B\times A}$ reachable from ξ, with $\xi' = (b, a)(k((b_1, a_1)(i_1(\xi_1^1, \dots, \xi_{l_{i_1}}^1)), \dots, (b_j, a_j)(i_j(\xi_1^j, \dots, \xi_{l_{i_j}}^j))))$ we have $\beta(b) = \iota_k(b_1, \dots, b_j)$ and $\alpha(a) \ni \iota_k(a_1, \dots, a_j)$ where $j = j_k$ is the arity of k.*

A run is accepting *if it is accepting in the sense of Def. 17, w.r.t. the parity function $\Omega' : B \times A \to \omega$ given by $\Omega'(b, a) := \Omega(a)$. The automaton (A, α, a_I, Ω)* accepts *the pointed F-coalgebra (B, β, b_I) if there exists an accepting run of (A, α, a_I, Ω) on (B, β, b_I).*

▶ **Example 22.** The following are accepting runs of the automaton in Example 20 on the paths in Example 16 (viewed as F-coalgebras):

$$(s,0) \xrightarrow{c} (s,0) \xrightarrow{c} \dots \qquad (s,0) \xrightarrow{r} \underset{2nd}{\downarrow} \xrightarrow{1st} (s,0) \xrightarrow{r} \underset{2nd}{\downarrow} \xrightarrow{1st} (s,0) \xrightarrow{r} \dots$$
$$(t,1) \xrightarrow{a} \qquad (t,1) \xrightarrow{a}$$

On the other hand, the following run is not accepting:

$$(s,0) \xrightarrow{r} \underset{2nd}{\downarrow} \xrightarrow{1st} (s,0) \xrightarrow{r} \underset{2nd}{\downarrow} \xrightarrow{1st} (s,0) \xrightarrow{r} \dots$$
$$(t,1) \xrightarrow{b} (t,1) \xrightarrow{b} \dots \qquad (t,1) \xrightarrow{a}$$

Unambiguous automata will play an important role in what follows.

▶ **Definition 23** (Unambiguous parity F-automaton)**.** *A non-deterministic parity F-automaton (A, α, a_I, Ω) is called* unambiguous *if for each pointed F-coalgebra (B, β, b_I), there exists at most one accepting run of (A, α, a_I, Ω) on (B, β, b_I).*

Since paths in a $(\mathsf{T}_S \circ F)$-coalgebra (C, γ) carry F-coalgebra structure (see Remark 15), one can consider (accepting) runs of a non-deterministic parity F-automaton on them. The next two sub-sections describe two different ways of *measuring* the set of paths of a pointed $(\mathsf{T}_S \circ F)$-coalgebra which are accepted by a given NPA. Before that, we show how non-deterministic and probabilistic transition systems can be recovered in our framework, and how the associated notion of NPA relates to the standard notion of Büchi automaton.

▶ Remark 24. Let At denote a finite set of atomic propositions. Take $F : \mathsf{Set} \to \mathsf{Set}$ be given by $F = \mathcal{P}(\mathsf{At}) \times \mathsf{Id} \simeq \coprod_{A \subseteq \mathsf{At}} \mathsf{Id}$. Then, non-deterministic (probabilistic) transition systems can be viewed as $(\mathsf{T}_S \circ F)$-coalgebras, with S the boolean (resp. probabilistic) semiring: such transition systems are in one-to-one correspondence with $\mathcal{P}(\mathsf{At}) \times \mathsf{T}_S$-coalgebras, which can be turned into $\mathsf{T}_S \circ (\mathcal{P}(\mathsf{At}) \times \mathsf{Id})$-coalgebras by post-composing the coalgebra maps with the strength map of T_S. Moreover, Büchi automata over the alphabet $\mathcal{P}(\mathsf{At})$ coincide with non-deterministic parity F-automata with $\mathsf{ran}(\Omega) = \{1, 2\}$.

2.5 Quantitative Parity Automata and their Extents

The notion of *ν-extent*, defined next, generalises non-emptiness in non-deterministic coalgebras (existence of a maximal path) to coalgebras with quantitative branching. It assigns, to each coalgebra state, a value in S which "measures" the maximal (completed) paths from it.

▶ **Definition 25** (ν-extent, [6]). *The ν-extent of a $(\mathsf{T}_S \circ F)$-coalgebra (C, γ) is the greatest fixpoint of the operator on S-valued predicates on C, which takes $p : C \to S$ to the composition*

$$C \xrightarrow{\gamma} \mathsf{T}_S FC \xrightarrow{\mathsf{T}_S Fp} \mathsf{T}_S FS \xrightarrow{\mathsf{T}_S(\bullet_F)} \mathsf{T}_S S = \mathsf{T}_S \mathsf{T}_S 1 \xrightarrow{\sqcup_1} \mathsf{T}_S 1 = S$$

where $\bullet_F : FS \to S$ is given by $\bullet_F(\iota_i(s_1, \ldots, s_{j_i})) = s_1 \bullet \ldots \bullet s_{j_i}$ for $i \in I$. We write $\mathsf{ext}^\nu_\gamma : C \to S$ for the ν-extent of (C, γ).

The operator in Definition 25 expresses that the ν-extent of a state is the weighted sum of the ν-extents of its (structured) successors, where in the case of a *structured* successor (tuple of states resulting from an individual transition), the ν-extents of the states in question are multiplied. We will later use the ν-extent to measure certain sets of paths from a given state of a $(\mathsf{T}_S \circ F)$-coalgebra (C, γ). In particular, the set of *all* paths from $c \in C$ in (C, γ) will have measure $\mathsf{ext}^\nu_\gamma(c)$. This is further motivated by the next example.

▶ **Example 26.** When $S = (\{0, 1\}, \vee, 0, \wedge, 1)$, the ν-extent of a state c in a $(\mathsf{T}_S \circ F)$-coalgebra (C, γ) is 1 iff there exists a maximal path from c in (C, γ). When $S = ([0, 1], +, 0, *, 1)$, the ν-extent of a state measures the probability of not deadlocking; in particular, the ν-extent is always 1 provided that all states of (C, γ) have branching governed by a probability *distribution*. Finally, when $S = (\mathbb{N}^\infty, \min, \infty, +, 0)$, the ν-extent of a state c gives the minimal cost of a maximal path from c in (C, γ).

▶ **Example 27.** Consider the $(\mathsf{T}_S \circ F)$-coalgebra (C, γ) from Example 10, with $C = \{s, t\}$. Its ν-extent $\mathsf{ext}^\nu_\gamma : C \to [0, 1]$ is the greatest solution of the following system of equations (one variable for each state, with x being used for state s and y being used for state t):

$$\begin{bmatrix} x & = & \frac{1}{2} * x + \frac{1}{2} * x * y \\ y & = & \frac{1}{4} * y + \frac{3}{4} \end{bmatrix}$$

This gives $\mathsf{ext}^\nu_\gamma(s) = \mathsf{ext}^\nu_\gamma(t) = 1$. Replacing the probabilistic semiring with the tropical one and assigning weight 0 (the top element of $(S, \sqsubseteq)$) to r and a transitions, and weight 1 to b and c transitions, results in a ν-extent of 0 for both s and t.

Quantitative parity automata generalise NPAs by allowing weighted branching.

▶ **Definition 28** (Quantitative parity automaton, [6]). *A* parity (T_S, F)-automaton*, or simply* quantitative parity automaton (QPA)*, (D, δ, d_I, Ω) is given by a pointed $T_S \circ F$-coalgebra (D, δ, d_I) together with a parity map $\Omega : D \to \omega$.*

We will obtain QPAs as products between an unambiguous NPA, representing a qualitative property of pointed F-coalgebras, and a quantitative model. We will then show that such products can be used to measure the degree with which the given property is satisfied in the model (thereby generalising the automata-based approach to model checking non-deterministic and probabilistic systems). This will amount to determining the *nested extent* of the product automaton, to be defined shortly. This generalises the ν-extent of a $(\mathsf{T}_S \circ F)$-coalgebra by also taking into account the state parities. We first define the product automaton.

▶ **Definition 29** (Product automaton). *Let $(S, +, 0, \bullet, 1)$ be a total semiring satisfying Assumption 6. Also, let (A, α, a_I, Ω) be a NPA and let (C, γ, c_I) be a pointed $(\mathsf{T}_S \circ F)$-coalgebra. The* product *of (C, γ, c_I) and (A, α, a_I, Ω) is the QPA with carrier $C \times A$, parity map given by $\Omega(c, a) = \Omega(a)$ for $(c, a) \in C \times A$, and transition map $\mathrm{prod}_{\gamma,\alpha}$ given by:*

$$C \times A \xrightarrow{\gamma \times \alpha} \mathsf{T}_S FC \times \mathcal{P}_f FA \xrightarrow{d_{FC,FA}} \mathsf{T}_S(FC \times FA) \xrightarrow{\langle F\pi_1, F\pi_2\rangle^*} \mathsf{T}_S F(C \times A)$$

where for $X, Y \in \mathsf{Set}$, the map $d_{X,Y} : \mathsf{T}_S X \times \mathcal{P}_f Y \to \mathsf{T}_S(X \times Y)$ is given by:

$$\mathsf{T}_S X \times \mathcal{P}_f Y \xrightarrow{\mathsf{id}_{\mathsf{T}_S X} \times e_Y} T_S X \times \mathsf{T}_S Y \xrightarrow{\mathsf{dst}_{X,Y}} \mathsf{T}_S(X \times Y) \tag{4}$$

with

- *$e : \mathcal{P}_f \Rightarrow \mathsf{T}_S$ the embedding of $\mathcal{P}_f$ into T_S, given by*

$$e_Y(X)(y) = \begin{cases} 1, & \text{if } y \in X \\ 0, & \text{otherwise} \end{cases}, \quad \text{for } X \in \mathcal{P}_f Y,$$

- *$\mathsf{dst}_{X,Y} : \mathsf{T}_S X \times \mathsf{T}_S Y \Rightarrow \mathsf{T}_S(X \times Y)$ the* double strength *of T_S, given by*

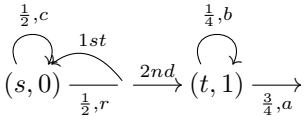
$$\mathsf{dst}_{X,Y}(\varphi, \psi) = \sum_{x \in \mathsf{supp}(\varphi), y \in \mathsf{supp}(\psi)} (\varphi(x) \bullet \psi(y))(x, y), \quad \text{for } \varphi \in \mathsf{T}_S X \text{ and } \psi \in \mathsf{T}_S Y,$$

and where $\langle F\pi_1, F\pi_2\rangle^*$ is pre-composition with $\langle F\pi_1, F\pi_2\rangle : F(C \times A) \to FC \times FA$.

We immediately note that the shape of the functor F makes $\langle F\pi_1, F\pi_2\rangle$ injective, and as a result the transition map of the product automaton has finite support.

Transitions in the product automaton thus arise from matching transitions in the $(\mathsf{T}_S \circ F)$-coalgebra and the NPA, with weights inherited from the coalgebra and parities inherited from the NPA; in particular, a coalgebra transition may match more than one NPA transition. The assumption in Definition 29 that $(S, +, 0, \bullet, 1)$ is total ensures that the natural transformation e is well defined. We will explain in Section 4 why this assumption is harmless.

▶ **Example 30.** The product of the coalgebra in Example 10 with the NPA in Example 20 is:



The next lemma characterises paths in a $(\mathsf{T}_S \circ F)$-coalgebra accepted by an unambiguous NPA using the product automaton. It is proved by simply spelling out the relevant definitions.

▶ **Lemma 31.** *Assume $(S, +, 0, \bullet, 1)$ is a total semiring. Let (A, α, a_I, Ω) be an unambiguous parity automaton and (C, γ, c_I) be a pointed $(\mathsf{T}_S \circ F)$-coalgebra. There is a one-to-one correspondence between accepting paths from (c_I, a_I) in the product of (A, α, a_I, Ω) and (C, γ, c_I), and paths from c_I in (C, γ) accepted by (A, α, a_I, Ω).*

As announced, the notion of *nested extent* of a QPA generalises the ν-extent of a $(\mathsf{T}_S \circ F)$-coalgebra by taking into account the different parities associated to automaton states.

▶ **Definition 32** (Nested extent, [6])**.** *Let (D, δ, d_I, Ω) be a quantitative parity automaton with $\mathsf{ran}(\Omega) = \{0, \dots, n\}$, let $D_k = \{d \in D \mid \Omega(d) = k\}$, and let $\delta_k = \delta \circ \iota_k : D_k \to \mathsf{T}_S F D$ denote the restriction of δ to D_k ($k \in \mathsf{ran}(\Omega)$). The* extent $\mathsf{ext}_\delta = [\mathsf{ext}_{\delta,0}, \dots, \mathsf{ext}_{\delta,n}] : D \to S$ *of (D, δ, Ω) is the solution of the following nested system of equations:*

$$\begin{bmatrix} x_0 & =_\nu & \sqcup_1 \circ \mathsf{T}_S(\bullet_F) \circ T_S F[x_0, \dots, x_n] \circ \delta_0 \\ x_1 & =_\mu & \sqcup_1 \circ \mathsf{T}_S(\bullet_F) \circ T_S F[x_0, \dots, x_n] \circ \delta_1 \\ & \vdots & \\ x_n & =_\eta & \sqcup_1 \circ \mathsf{T}_S(\bullet_F) \circ T_S F[x_0, \dots, x_n] \circ \delta_n \end{bmatrix} \tag{5}$$

with $\eta = \mu$ ($= \nu$) if n is odd (resp. even), variables x_k ($k \in \mathsf{ran}(\Omega)$) taking values in the poset $(S^{D_k}, \sqsubseteq)$ (and therefore $[x_0, \dots, x_n] \in S^D$), and the rhs of the equation for x_k pictured below:

$$D_k \xrightarrow{\delta_k} \mathsf{T}_S F D \xrightarrow{\mathsf{T}_S F[x_0,\dots,x_n]} \mathsf{T}_S F S \xrightarrow{\mathsf{T}_S(\bullet_F)} \mathsf{T}_S S = \mathsf{T}_S \mathsf{T}_S 1 \xrightarrow{\sqcup_1} \mathsf{T}_S 1 = S$$

We write $\mathsf{Op}_{\delta,i} : S^{D_0} \times \ldots \times S^{D_n} \to S^{D_i}$ for the operator used in the rhs of the ith equation.

The existence and uniqueness of a solution for (5) is guaranteed by Kleene's theorem (Thm. 3).

▶ **Example 33.** The nested extent of the product automaton in Example 30 is the solution of the following nested system of equations (where variables x and y are used for the nested extents of states $(s, 0)$, respectively $(t, 1)$):

$$\begin{bmatrix} x & =_\nu & \frac{1}{2} * x + \frac{1}{2} * x * y \\ y & =_\mu & \frac{1}{4} * y + \frac{3}{4} \end{bmatrix}$$

This still gives a nested extent of 1 in each state, essentially because the probability of infinitely-many b-transitions from state $(t, 1)$ is 0.

2.6 Semiring-Valued Measures

We will use *semiring-valued measures* [4] to measure certain sets of paths from a state of a $(\mathsf{T}_S \circ F)$-coalgebra. In particular, we will be able to measure the set of paths accepted by an NPA. Key definitions and results regarding semiring-valued measures are summarised below.

▶ **Definition 34** ([4])**.** *An S-valued* measure *on a σ-algebra $\mathcal{A}$ is a function $\mu : \mathcal{A} \to S$ s.t. (i) $\mu(\emptyset) = 0$, and (ii) if $A_i \in \mathcal{A}$ for $i \in \omega$ are pairwise disjoint, then $\sum_{i \in \omega} \mu(A_i)$ is defined and moreover, $\mu(\bigcup_{i \in \omega} A_i) = \sum_{i \in \omega} \mu(A_i)$.*

▶ **Proposition 35** ([4])**.** *Let $\mu : \mathsf{R} \to S$ be a measure on a field of sets. Then, μ extends to a measure on the σ-algebra generated by R.*

The proof of the above result defines the resulting measure as

$$\mu^*(A) = \inf\{ \sum_{n \in \omega} \mu(E_n) \mid (E_n \in \mathsf{R})_{n \in \omega} \text{ pairwise disjoint}, \ A \subseteq \bigcup_{n \in \omega} E_n \}$$

As in [4], we take R to be the field generated by the so-called *cylinder sets*.

▶ **Definition 36** ([4]). *Let (C,γ) be a $(\mathsf{T}_S \circ F)$-coalgebra, and let $\tau \in \Phi_C$ be a path fragment from c in (C,γ). Its associated* cylinder set *is given by* $\mathsf{Cyl}(\tau) = \{\xi \in \mathsf{Paths}_c \mid \tau \in \mathsf{pref}(\xi)\}$. *A cylinder set $\mathsf{Cyl}(\tau)$ is said to* cover *a path $\xi \in Z_C$ when τ is a prefix of ξ. For $c \in C$, we let $\Sigma_c := \{\, \mathsf{Cyl}(\tau) \mid \tau$ is a path fragment from c in $(C,\gamma)\,\}$.*

Now given a $(\mathsf{T}_S \circ F)$-coalgebra (C,γ) and $c \in C$, it is shown in [4] that finite unions of pairwise-disjoint elements of Σ_c form a field. Then, an S-valued measure on the generated σ-algebra, denoted by $\mathcal{M}_c$, can be defined from an S-valued measure on Σ_c, using Proposition 35. The natural S-valued measure to consider on cylinder sets is $\mu_\gamma : \Sigma_c \to S$ given by:

1. $\mu_\gamma(\emptyset) = 0$,
2. For τ a path fragment from $c \in C$, $\mu_\gamma(\mathsf{Cyl}(\tau))$ is defined by structural induction on τ:
   a. If $\tau = c(*)$, then $\mu_\gamma(\mathsf{Cyl}(\tau)) = \mathsf{ext}^\nu_\gamma(c)$,
   b. If $\tau = c(i(\tau_1,\dots,\tau_{j_i}))$ for some $i \in I$ and for path fragments τ_k from $c_k \in C$ for $k \in \{1,\dots,j_i\}$, then $\mu_\gamma(\mathsf{Cyl}(\tau)) = \gamma(c)(\iota_i(c_1,\dots,c_{j_i})) \bullet \mu_\gamma(\mathsf{Cyl}(\tau_1)) \bullet \dots \bullet \mu_\gamma(\mathsf{Cyl}(\tau_{j_i}))$.

Note that the measure of the set Paths_c of *all* paths from c is not 1 (the top element in S) as one might expect, but $\mathsf{ext}^\nu_\gamma(c)$. This is because we consider maximal (completed) paths only, and assigning measure 1 to Paths_c could result in assigning measure 1 to an *empty* set of paths (when there are no completed paths from c, e.g. because c is a deadlock state).

The above $\mu_\gamma : \Sigma_c \to S$ induces an S-valued measure on the ring generated by Σ_c, given by $\mu_\gamma(\bigcup_{i\in\{1,\dots,n\}} C_i) = \sum_{i\in\{1,\dots,n\}} \mu_\gamma(C_i)$ for each pairwise-disjoint family $(C_i)_{i\in\{1,\dots,n\}}$ with $C_i \in \Sigma_c$. The measure $\mu_\gamma : \mathcal{M}_c \to S$ arising from Proposition 35 is then given by

$$\mu_\gamma(A) = \inf\{ \sum_{n\in\omega} \mu_\gamma(C_n) \mid (C_n \in \Sigma_c)_{n\in\omega} \text{ pairwise disjoint},\ A \subseteq \bigcup_{n\in\omega} C_n\} \tag{6}$$

▶ **Example 37.** When $S = (\{0,1\}, \vee, 0, \wedge, 1)$, $\mu_\gamma(A) = 0$ iff $A = \emptyset$. When $S = ([0,1], +, 0, *, 1)$, and if only probability *distributions* are used in γ, $\mu_\gamma(A)$ gives the likelihood of exhibiting a path in A. When $S = (\mathbb{N}^\infty, \min, \infty, +, 0)$, $\mu_\gamma(A)$ gives the minimal cost of a path in A.

3 Coincidence of Extents with the Measure-Theoretic Semantics

Throughout this section we fix a quantitative parity automaton (C,γ,c_I,Ω). We will use the measures $\mu_\gamma : \mathcal{M}_c \to S$ with $c \in C$ to link a characterisation of the accepting paths of (C,γ,Ω) (Proposition 39 below) with the definition of extent (Definition 32), thereby proving the equivalence of two different ways of measuring the set of accepting paths of a QPA.

The next result shows that extents are preserved by parity-preserving $(\mathsf{T}_S \circ F)$-coalgebra homomorphisms.

▶ **Proposition 38.** *Let (C,γ,Ω) and (D,δ,Ω) be two quantitative parity automata and let $f : (C,\gamma,\Omega) \to (D,\delta,\Omega)$ be a $(\mathsf{T}_S \circ F)$-coalgebra homomorphism which preserves parities; that is, $\Omega(f(c)) = \Omega(c)$ for $c \in C$. Then, $\mathsf{ext}_\gamma(c) = \mathsf{ext}_\delta(f(c))$ for all $c \in C$.*

To relate the extent of (C,γ,Ω) with the set of *accepting* paths of (C,γ,Ω), we characterise the accepting paths of a QPA as the solution of a nested system of equations. For $i \in \mathsf{ran}(\Omega)$, we let $\mathsf{Paths}_i = \{\xi \in Z_C \mid \exists c \in C.\ \xi \in \mathsf{Paths}_c \text{ and } \Omega(c) = i\}$; that is, Paths_i contains all paths in (C,γ) whose initial state has parity i. The next result is a reformulation of [15, Lemma 4.4]; its proof mirrors that in loc. cit. It is irrelevant that transitions carry weights.

▶ **Proposition 39.** *The accepting paths of a QPA (C, γ, Ω) are the solution of the following nested system of equations, with variables Y_i taking values in the lattice $\mathcal{P}(\mathsf{Paths}_i)$:*

$$\begin{bmatrix} Y_0 & =_\nu & \mathsf{Op}_0(Y_0, \ldots, Y_n) \\ Y_1 & =_\mu & \mathsf{Op}_1(Y_0, \ldots, Y_n) \\ & \vdots & \\ Y_n & =_\eta & \mathsf{Op}_n(Y_0, \ldots, Y_n) \end{bmatrix} \tag{7}$$

where for $k \in \mathsf{ran}(\Omega)$, $\mathsf{Op}_k : \mathcal{P}(\mathsf{Paths}_0) \times \ldots \times \mathcal{P}(\mathsf{Paths}_n) \to \mathcal{P}(\mathsf{Paths}_k)$ is given by

$$\begin{aligned} \mathsf{Op}_k((Y_i)_{i \in \mathsf{ran}(\Omega)}) \; = \{ \, \xi \in \mathsf{Paths}_c \mid \xi = c(i(\xi_1, \ldots, \xi_{j_i})) \text{ for some } c \in C_k, \\ i \in I \text{ and } \xi_l \in Y_{\Omega(\pi_1(\zeta_C(\xi_l)))} \text{ for } l \in \{1, \ldots, j_i\} \, \} \end{aligned}$$

The idea is that the kth component of the solution collects all accepting paths from states with parity k. Now while the domain of the operators $\mathsf{Op}_k : \mathcal{P}(\mathsf{Paths}_0) \times \ldots \times \mathcal{P}(\mathsf{Paths}_n) \to \mathcal{P}(\mathsf{Paths}_k)$ with $i \in \mathsf{ran}(\Omega)$ also includes tuples $(P_0, \ldots, P_n)$ with $P_k \cap \mathsf{Paths}_c$ not measurable for some $k \in \mathsf{ran}(\Omega)$ and $c \in C_k$, we will show that only tuples $(P_0, \ldots, P_n)$ with $P_k \cap \mathsf{Paths}_c$ measurable for $k \in \mathsf{ran}(\Omega)$ and $c \in C_k$ are involved in the construction of the solution of this system of equations, and the solution itself is measurable in the sense of Definition 40 below.

▶ **Definition 40.** *For $k \in \mathsf{ran}(\Omega)$, we call a set of paths $P \subseteq \mathsf{Paths}_k$* measurable *if $P \cap \mathsf{Paths}_c \in \mathcal{M}_c$ for all $c \in C_k$. We write $\mathcal{M}_k := \{ \, P \subseteq \mathsf{Paths}_k \mid P \text{ is measurable} \, \}$, for $k \in \mathsf{ran}(\Omega)$.*

The next result shows that the operators in Proposition 39 restrict to measurable sets and moreover, the solution of the equation system (7) itself consists of measurable sets.

▶ **Proposition 41.** *Let E' be the equation system (7). Then, the following hold:*

1. *For $i \in \mathsf{ran}(\Omega)$ and $P_k \in \mathcal{M}_k$ for $k \in \{i+1, \ldots, n\}$, the operator $\mathsf{Op}_i^{P_{i+1}, \ldots, P_n} : \mathsf{Paths}_i \to \mathsf{Paths}_i$ restricts to an operator on $\mathcal{M}_i$.*
2. *E' restricts to an equation system with variables taking values in $\mathcal{M}_i$, whose solution coincides with the solution of E'.*

Proof. For $i \in \mathsf{ran}(\Omega)$, $\mathcal{P}(\mathsf{Paths}_i)$ is a complete lattice. Also, $\mathcal{M}_i \subseteq \mathcal{P}(\mathsf{Paths}_i)$ is a σ-algebra, with countable directed unions / co-directed intersections computed component-wise – recall that each $P \in \mathcal{M}_i$ is a disjoint union of sets $P_c \in \mathcal{M}_c$ with $c \in C_i$. Then, an easy induction on i shows that, if $P_k \in \mathcal{M}_k$ for $k \in \{i+1, \ldots, n\}$, then $\mathsf{Op}_i^{P_{i+1}, \ldots, P_n}$ restricts to an operator on $\mathcal{M}_i$ – this is because the least/greatest fixpoints required in the definition of $\mathsf{Op}_i^{P_{i+1}, \ldots, P_n}$ are constructed by successively taking limits of ω-chains/ω^{op}-chains of elements of $\mathcal{M}_i$ (see Theorem 3), and the $\mathcal{M}_i$s are closed under countable directed unions / co-directed intersections. As a result, E' restricts to an equation system with variables taking values in $\mathcal{M}_i$, with $i \in \mathsf{ran}(\Omega)$. Moreover, the construction of the solution is the same whether performed in $\mathcal{M}_i$ or in $\mathcal{P}(\mathsf{Paths}_i)$, with $i \in \mathsf{ran}(\Omega)$. This concludes the proof. ◀

We are now ready to state our main result.

▶ **Theorem 42.** *For a quantitative parity automaton (C, γ, Ω) and $c \in C$, we have*

$$\mathsf{ext}_\gamma(c) = \mu_\gamma(\{ \, \xi \in \mathsf{Paths}_c \mid \xi \text{ accepting} \, \}).$$

Proof. By Assumption 6, proving the above equality can be reduced to proving two inequalities. These follow from Lemmas 43 and 46, respectively. ◀

▶ **Lemma 43.** *For a quantitative parity automaton (C, γ, Ω) and $c \in C$, we have*

$$\mu_\gamma(\{\, \xi \in \mathsf{Paths}_c \mid \xi \text{ accepting} \,\}) \sqsubseteq \mathsf{ext}_\gamma(c).$$

Proof. Consider the equation system E in (5), and the restriction of the equation system E' in (7) to *measurable* sets of paths (see Proposition 41). The operators $\mathsf{Op}_i^{P_{i+1},\dots,P_n}$ and $\mathsf{Op}_{\gamma,i}^{e_{i+1},\dots,e_n}$ used to define $\mathsf{sol}(E)$ and $\mathsf{sol}(E')$ are given by:

$$\begin{aligned} \mathsf{Op}_i^{P_{i+1},\dots,P_n}(Y) = &\quad \mathsf{Op}_i(\mathsf{sol}(E[Y_i := Y, Y_{i+1} := P_{i+1}, \dots, Y_n := P_n]), Y, P_{i+1}, \dots, P_n) \\ \mathsf{Op}_{\gamma,i}^{e_{i+1},\dots,e_n}(x) = &\quad \mathsf{Op}_{\gamma,i}(\mathsf{sol}(E'[x_i := x, x_{i+1} := e_{i+1}, \dots, x_n := e_n]), x, e_{i+1}, \dots, e_n) \end{aligned}$$

We prove the following combined statement by induction on $i \in \mathsf{ran}(\Omega)$:

1. Given $P_j \in \mathcal{M}_j$ and $e_j : C_j \to S$ such that $\mu_\gamma(P_j \cap \mathsf{Paths}_c) \sqsubseteq e_j(c)$ for $c \in C_j$, for $j \in \{i+1, \dots, n\}$, we have

$$\begin{array}{ccc} \mathcal{M}_i & \xrightarrow{\mathsf{Op}_i^{P_{i+1},\dots,P_n}} & \mathcal{M}_i \\ {\scriptstyle\mu_\gamma}\downarrow & \sqsupseteq & \downarrow{\scriptstyle\mu_\gamma} \\ S^{C_i} & \xrightarrow[\mathsf{Op}_{\gamma,i}^{e_{i+1},\dots,e_n}]{} & S^{C_i} \end{array} \tag{8}$$

   Here, by slightly abusing notation, we write $\mu_\gamma : \mathcal{M}_i \to S^{C_i}$ for the function taking P_i to the S-valued predicate $e_i : C_i \to S$ given by $e_i(c) = \mu_\gamma(P_i \cap \mathsf{Paths}_c)$ for $c \in C_i$.
2. $\mu_\gamma(\eta_i(\mathsf{Op}_i^{P_{i+1},\dots,P_n})) \sqsubseteq \eta_i(\mathsf{Op}_{\gamma,i}^{e_{i+1},\dots,e_n})$, whenever $P_j \in \mathcal{M}_j$ and $e_j : C_j \to S$ are as above, for $j \in \{i+1, \dots, n\}$.

Since for $i \in \mathsf{ran}(\Omega)$, any $P_i \in \mathcal{M}_i$ is of the form $P_i = \bigcup_{c \in C_i} P_{i,c}$, with $P_{i,c} = P_i \cap \mathsf{Paths}_c$ for $c \in C_i$, it suffices to show that (8) holds when restricted to each $\mathcal{M}_c$ with $c \in C_i$.

- For $i = 0$, the inequality (8) follows from

$$\mu_\gamma(\mathsf{Op}_0^{P_1,\dots,P_n}(P_{0,c})) \;=\; \mathsf{Op}_{\gamma,0}^{\mu_\gamma(P_1),\dots,\mu_\gamma(P_n)}(\mu_\gamma(P_{0,c})) \;\sqsubseteq\; \mathsf{Op}_{\gamma,0}^{e_1,\dots,e_n}(\mu_\gamma(P_{0,c}))$$

  for $P_0 \in \mathcal{M}_0$ and $c \in C_0$. In the above, the equality follows from [4, Proposition 5.12], after noting that $\mathsf{Op}_0^{P_1,\dots,P_n}(P_{0,c})$ can be written as a finite union of sets of the form

$$\{\, \xi \in \mathsf{Paths}_c \mid \zeta_C(\xi) = (c, \iota_i(\xi_1, \dots, \xi_{j_i})) \text{ with } \xi_i \in P_{\Omega(\pi_1(\zeta_C(\xi_i)))} \text{ for } i \in \{1, \dots, j_i\} \,\}$$

  with $i \in I$. On the other hand, the inequality above follows by Remark 4.
  Now let $P_j \in \mathcal{M}_j$ and $e_j : C_j \to S$ be s.t. $\mu_\gamma(P_j \cap \mathsf{Paths}_c) \sqsubseteq e_j(c)$ for $c \in C_j$ and $j \in \{1, \dots, n\}$. Also, let $P_0 = \nu_0(\mathsf{Op}_0^{P_1,\dots,P_n})$ and $e_0 = \nu_0(\mathsf{Op}_{\gamma,0}^{e_1,\dots,e_n})$. We show that $\mu_\gamma(P_0) \sqsubseteq e_0$. We have

$$\mathsf{Op}_{\gamma,0}^{e_1,\dots,e_n}(\mu_\gamma(P_0)) \overset{(\text{by } (8))}{\sqsupseteq} \mu_\gamma(\mathsf{Op}_0^{P_1,\dots,P_n}(P_0)) \overset{(P_0 \text{ is a fixpoint of } \mathsf{Op}_0^{P_1,\dots,P_n})}{=} \mu_\gamma(P_0)$$

  and therefore $\mu_\gamma(P_0)$ is a post-fixpoint of $\mathsf{Op}_{\gamma,0}^{e_1,\dots,e_n}$. Now since e_0 is the greatest post-fixpoint of $\mathsf{Op}_{\gamma,0}^{e_1,\dots,e_n}$, we immediately obtain $\mu_\gamma(P_0) \sqsubseteq e_0$.
- Now assume that the combined statement holds for all $j < i$, with $0 < i \leq n$. To show that it holds for i, we proceed as in the base case. The inequality (8) follows again using [4, Proposition 5.12], Remark 4, and the induction hypothesis (namely $\mu_\gamma(\eta_j(\mathsf{Op}_j^{P_{j+1},\dots,P_n})) \sqsubseteq \eta_j(\mathsf{Op}_{\gamma,j}^{e_{j+1},\dots,e_n})$ for $0 \leq j < i$). To show that $\mu_\gamma(\eta_i(\mathsf{Op}_i^{P_{i+1},\dots,P_n})) \sqsubseteq \eta_i(\mathsf{Op}_{\gamma,i}^{e_{i+1},\dots,e_n})$ whenever $P_j \in \mathcal{M}_j$ and $e_j : C_j \to S$ are such that $\mu_\gamma(P_j \cap \mathsf{Paths}_c) \sqsubseteq e_j(c)$ for $c \in C_j$ and $j \in \{i+1, \dots, n\}$, we distinguish two sub-cases.

- i is even. In this case the proof is similar to the base case.
- i is odd. We consider the ordinal-indexed sequence used to obtain the least fixpoint P_i of $\mathsf{Op}_i^{P_{i+1},\ldots,P_n}$. Induction on ordinals together with (8) and the fact that $\mu_\gamma(\bigcup_{i\in\omega} A_i) = \sup_{i\in\omega}\mu_\gamma(A_i)$ for any increasing chain $A_0 \subseteq A_1 \subseteq \ldots$ can be used to show that $\mu_\gamma((\mathsf{Op}_i^{P_{i+1},\ldots,P_n})^\alpha(\emptyset)) \sqsubseteq (\mathsf{Op}_{\gamma,i}^{\mu_\gamma(P_{i+1}),\ldots,\mu_\gamma(P_n)})^\alpha(0)$:
  - For $\alpha = 0$, $\mu_\gamma(\emptyset) = 0 \sqsubseteq 0$.
  - For $\alpha = \beta+1$, assuming $\mu_\gamma((\mathsf{Op}_i^{P_{i+1},\ldots,P_n})^\beta(\emptyset)) \sqsubseteq (\mathsf{Op}_{\gamma,i}^{\mu_\gamma(P_{i+1}),\ldots,\mu_\gamma(P_n)})^\beta(0)$, we have
$$\mu_\gamma((\mathsf{Op}_i^{P_{i+1},\ldots,P_n})^{\beta+1}(\emptyset)) \overset{(\text{by (8)})}{\sqsubseteq} (\mathsf{Op}_{\gamma,i}^{\mu_\gamma(P_{i+1}),\ldots,\mu_\gamma(P_n)})(\mu_\gamma((\mathsf{Op}_i^{P_{i+1},\ldots,P_n})^\beta(\emptyset)) \overset{(\text{I.H.})}{\sqsubseteq} (\mathsf{Op}_{\gamma,i}^{\mu_\gamma(P_{i+1}),\ldots,\mu_\gamma(P_n)})^{\beta+1}(0)$$
  - For α a limit ordinal, we have
$$\mu_\gamma((\mathsf{Op}_i^{P_{i+1},\ldots,P_n})^\alpha(\emptyset)) = \sup_{\beta<\alpha}\mu_\gamma((\mathsf{Op}_i^{P_{i+1},\ldots,P_n})^\beta(\emptyset)) \overset{(\text{I.H.})}{\sqsubseteq} \sup_{\beta<\alpha}(\mathsf{Op}_{\gamma,i}^{\mu_\gamma(P_{i+1}),\ldots,\mu_\gamma(P_n)})^\beta(\emptyset)) \overset{(\text{Remark 4})}{\sqsubseteq} \sup_{\beta<\alpha}(\mathsf{Op}_{\gamma,i}^{e_{i+1},\ldots,e_n})^\beta(\emptyset))$$
    The equality above uses that $(\mathsf{Op}_i^{P_{i+1},\ldots,P_n})^\alpha(\emptyset)$ is the union of an increasing *countable* chain.

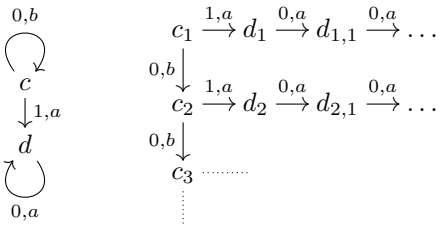
This concludes the proof of $\mu_\gamma(\{\,\xi \in \mathsf{Paths}_c \mid \xi \text{ accepting}\,\}) \sqsubseteq \mathsf{ext}_\gamma(c)$ for $c \in C$. ◀

We note in passing that, although the inequality (8) can be turned into an equality (by strengthening the relationship between the P_js and the e_js), this equality can not be used to prove the inequality $\mathsf{ext}_\gamma(c) \sqsubseteq \mu_\gamma(\{\,\xi \in \mathsf{Paths}_c \mid \xi \text{ accepting}\,\})$ in a similar way (by following the construction of the solutions of the two operators involved), since μ_γ does not behave well w.r.t. countable intersections (see [4, Example 5.10]).

We now turn to proving the second inequality. For this, we will use the so-called *unfolding* of a pointed $(\mathsf{T}_S \circ F)$-coalgebra.

▶ **Definition 44.** *The* unfolding *of a pointed $(\mathsf{T}_S \circ F)$-coalgebra (C, γ, c_I) is the pointed $(\mathsf{T}_S \circ F)$-coalgebra (B, β, b_I), where B contains a copy b_I of the initial state c_I, and for each copy $b \in B$ of some $c \in C$ and each transition $c \xrightarrow{w,i} (c_1, \ldots, c_{j_i})$ in (C, γ), (B, β) contains (new) copies $b_1, \ldots, b_{j_i}$ of $c_1, \ldots, c_{j_i}$ and a transition $b \xrightarrow{w,i} (b_1, \ldots, b_{j_i})$. If (C, γ, c_I) is a QPA, the states of (B, β, b_I) inherit parities from the corresponding states of C.*

▶ **Example 45.** Let $S = (\mathbb{N}^\infty, \min, \infty, +, 0)$ and $F = \{a, b\} \times \mathsf{Id} \simeq \mathsf{Id} + \mathsf{Id}$. The unfolding of the pointed $(\mathsf{T}_S \circ F)$-coalgebra on the left is the infinite tree on the right:



Now to motivate the proof of the next lemma, consider the automaton obtained by putting $\Omega(c) = 1$ and $\Omega(d) = 0$ in the above coalgebra. Then, the states of the unfolding inherit parities from c and d, and one can show that the extent of the unfolding coincides with the extent of the original (pointed) coalgebra; that is, $\mathsf{ext}_\gamma(c) = \mathsf{ext}_\beta(c_1)$. Now recall that $\mu_\gamma(\{\xi \in \mathsf{Paths}_c \mid$

ξ accepting$\}$) is given by $\inf\{\mu_\gamma[\mathcal{C}] \mid \mathcal{C}$ is a pairwise-disjoint cylinder set cover for $\{\xi \in \mathsf{Paths}_c \mid \xi \text{ accepting}\}\}$. So to prove that $\mathsf{ext}_\gamma(c) \sqsubseteq \mu_\gamma(\{\xi \in \mathsf{Paths}_c \mid \xi \text{ accepting}\})$, it would suffice to show that $\mathsf{ext}_\gamma(c) \sqsubseteq \mu_\gamma[\mathcal{C}]$ for *every* such cover $\mathcal{C}$. Let us consider, in the above example, one particular cover for $\{\xi \in \mathsf{Paths}_c \mid \xi \text{ accepting}\}$, given by: $C_1 = \mathsf{Cyl}(c(a(d(*))))$, $C_2 = \mathsf{Cyl}(c(b(c(a(d(*))))))$, ... We can use this cover to separate the unfolding of our automaton into a countable number of automata: one automaton $(B^k, \beta_k, b_I^k, \Omega_k)$ for each cylinder set C_k of $\mathcal{C}$, whose paths are precisely the paths in the unfolding covered by C_k (up to a renaming of the states in the unfolding to the original states in C), and one automaton $(B^0, \beta_0, b_I^0, \Omega_0)$ whose paths are those (non-accepting) paths not covered by any $C_k \in \mathcal{C}$:

$$c_1^1 \xrightarrow{1,a} d_1 \xrightarrow{0,a} \cdots\cdots$$
$$c_1^2 \xrightarrow{0,b} c_2^1 \xrightarrow{1,a} d_2 \xrightarrow{0,a} \cdots\cdots$$
$$c_1^3 \xrightarrow{0,b} c_2^2 \xrightarrow{0,b} c_3^1 \xrightarrow{1,a} d_3 \xrightarrow{0,a} \cdots\cdots$$
$$\cdots$$
$$c_1^0 \xrightarrow{0,b} c_2^0 \xrightarrow{0,b} c_3^0 \xrightarrow{0,b} \cdots\cdots$$

Then, to prove $\mathsf{ext}_\beta(c_1) \sqsubseteq \mu_\gamma[\mathcal{C}]$ (which would then give $\mathsf{ext}_\beta(c) \sqsubseteq \mu_\gamma[\mathcal{C}]$), it would suffice to prove the following:

- $\mathsf{ext}_\beta(c_1) \sqsubseteq \mathsf{ext}_{\beta_0}(c_1^0) + \sum_{k\in\{1,2,\ldots\}} \mu_{\beta_k}(C'_k)$,
- $\mathsf{ext}_{\beta_0}(c_1^0) = 0$, and
- $\mu_{\beta_k}(C'_k) = \mu_\gamma(C_k)$, where for $k \in \{1,2,\ldots\}$, the cylinder set C'_k is obtained from the cylinder set C_k by suitably renaming the states which label paths in C_k to states of B^k.

It turns out that all these statements can be proved in general, for any cover $\mathcal{C}$, as shown by (the proof of) the next lemma.

▶ **Lemma 46.** *For a quantitative parity automaton (C, γ, c_I, Ω), we have*

$$\mathsf{ext}_\gamma(c_I) \sqsubseteq \mu_\gamma(\{\xi \in \mathsf{Paths}_{c_I} \mid \xi \textit{ accepting}\}).$$

Proof (Sketch). We will use the fact that $\mu_\gamma(\{\xi \in \mathsf{Paths}_{c_I} \mid \xi \text{ accepting}\}) = \inf\{\mu_\gamma[\mathcal{C}] \mid \mathcal{C}$ is a pairwise-disjoint cylinder set cover for $\{\xi \in \mathsf{Paths}_{c_I} \mid \xi \text{ accepting}\}\}$. We fix a pairwise-disjoint cylinder set cover $\mathcal{C} = \{C_1, C_2, \ldots\}$ for $\{\xi \in \mathsf{Paths}_{c_I} \mid \xi \text{ accepting}\}$, and prove $\mathsf{ext}_\gamma(c_I) \sqsubseteq \mu_\gamma[\mathcal{C}]$. To this end, we write (B, β, b_I, Ω) for the unfolding of (C, γ, c_I, Ω). Also, for $k \in \{1,2,\ldots\}$, we let $(B^k, \beta_k, b_I^k, \Omega_k)$ denote the part of the automaton (B, β, b_I, Ω) covered by C_k (defined similarly to Example 45). Finally, we let $(B^0, \beta_0, b_I^0, \Omega_0)$ denote the part of the automaton (B, β, b_I, Ω) not covered by any C_k, with $k \in \{1,2,\ldots\}$. (The fact that (B, β, b_I, Ω) is a *tree* unfolding is needed here.) The required inequality is now a consequence of the following three statements:

1. $\mathsf{ext}_\gamma(c_I) = \mathsf{ext}_\beta(b_I)$.
2. If an automaton has *no* accepting paths, then it has extent 0.
3. $\mathsf{ext}_\beta(b_I) \sqsubseteq \mathsf{ext}_{\beta_0}(b_I^0) + \sum_{k\in\{1,2,\ldots\}} \mathsf{ext}^\nu_{\beta_k}(b_I^k)$.

The first statement follows immediately from applying Proposition 38 to the map sending each copy of a state in C to the original state in C. The proof of the second statement, omitted here due to space limitations, uses the computation of extent (see Thm. 3) to construct an accepting path from an automaton state with extent $\neq 0$. The proof of the third statement is by induction on $i \in \mathsf{ran}(\Omega)$ (see below). Then, using all these statements, we have:

$$\mathsf{ext}_\gamma(c_I) = \mathsf{ext}_\beta(b_I) \sqsubseteq \mathsf{ext}_{\beta_0}(b_I^0) + \sum_{k\in\{1,2,\ldots\}} \mathsf{ext}^\nu_{\beta_k}(b_I^k) = \sum_{k\in\{1,2,\ldots\}} \mu_\gamma(C_k) = \mu_\gamma[\mathcal{C}]$$

The second equality above uses the fact that the automaton $(B^0, \beta_0, b_I^0, \Omega_0)$ has no accepting paths (and therefore its extent is 0), together with the fact that, for $k \in \{1, 2, \ldots\}$, the automaton $(B^k, \beta_k, b_I^k, \Omega_k)$ contains (copies of) exactly those paths of (C, γ, Ω) which are covered by the cylinder set C_k (and therefore $\mathsf{ext}^\nu_{\beta_k}(b_I^k) = \mu_\gamma(C_k)$). This concludes the proof of the fact that $\mathsf{ext}_\gamma(c_I) \sqsubseteq \mu_\gamma[\mathcal{C}]$. Since this holds for every cover $\mathcal{C}$ for $\mu_\gamma(\{\, \xi \in \mathsf{Paths}_{c_I} \mid \xi \text{ accepting} \,\})$, we now obtain $\mathsf{ext}_\gamma(c_I) \sqsubseteq \mu_\gamma(\{\, \xi \in \mathsf{Paths}_{c_I} \mid \xi \text{ accepting} \,\})$ as required.

It remains to prove the third statement above. Now when the semiring S is partial, although the sum on the rhs of this statement is defined (it is equal to $\mu_\gamma[\mathcal{C}]$), some of the sums appearing later in the proof may not be defined. For this reason, we will interpret these sums in the *total* semiring $(S, \oplus, 0, \bullet, 1)$ (see Remark 8).

We will prove the following more general statement, in $(S, \oplus, 0, \bullet, 1)$:

$$\mathsf{ext}_\beta(b) \sqsubseteq \mathsf{ext}_{\beta_0}(b^0) + \sum_{k\in\{1,2,\ldots\}} \mathsf{ext}^\nu_{\beta_k}(b^k) \tag{9}$$

for each $b \in B$, where for $k \in \{0, 1, \ldots\}$, b^k is the copy of b which belongs to (B^k, β_k, Ω_k). For this, we prove by induction on $i \in \mathsf{ran}(\Omega)$ that:

$$(\eta_i(\mathsf{Op}^{e_{i+1},\ldots,e_n}_{\beta,i}))(b) \sqsubseteq (\eta_i(\mathsf{Op}^{e^0_{i+1},\ldots,e^0_n}_{\beta_0,i}))(b^0) + \sum_{k\in\{1,2,\ldots\}} \mathsf{ext}^\nu_{\beta_k}(b^k) \tag{10}$$

for each $b \in B_i$, whenever $e_j : B_j \to S$, $e_j^0 : B_j^0 \to S$ are such that $e_j \sqsubseteq e_j^0 + \sum_{k\in\{1,2,\ldots\}} (\mathsf{ext}^\nu_{\beta_k} \circ \iota_j)$ for $j \in \{i+1, \ldots, n\}$. In the above, ι_j denotes the inclusion of the set of states with parity j into the entire set of states. We immediately note that (10) holds trivially for those $b \in B_i$ for which the whole of Paths_b is covered by $\mathcal{C}$ – this follows from the definitions of extent and ν-extent, together with the pairwise-disjointness of the cylinder sets in $\mathcal{C}$. Therefore it suffices to show that (10) holds on states some of whose outgoing transitions belong to $(B^0, \beta_0, b_I^0, \Omega)$.

- Consider, first, the case when $i = 0$. Then, induction on ordinals can be used to show that $(\mathsf{Op}^{e_{i+1},\ldots,e_n}_{\beta,i})^\alpha(\top)(b) \sqsubseteq (\mathsf{Op}^{e^0_{i+1},\ldots,e^0_n}_{\beta_0,i})^\alpha(\top)(b^0) + \sum_{k\in\{1,2,\ldots\}} \mathsf{ext}^\nu_{\beta_k}(b^k)$ holds for all $b \in B_0$ and all ordinals α:
  - For $\alpha = 0$, the statement is trivial (both sides equal $1 \in S$).
  - For $\alpha = \gamma + 1$, assume that $(\mathsf{Op}^{e_1,\ldots,e_n}_{\beta,0})^\gamma(\top)(b) \sqsubseteq (\mathsf{Op}^{e^0_1,\ldots,e^0_n}_{\beta_0,0})^\gamma(\top)(b^0) + \sum_{k\in\{1,2,\ldots\}} \mathsf{ext}^\nu_{\beta_k}(b^k)$ holds for all $b \in B_0$. We then have, for $b \in B_0$:

$$(\mathsf{Op}^{e_1,\dots,e_n}_{\beta,0})^{\gamma+1}(\top)(b) =$$

(definition of $\mathsf{Op}^{e_1,\dots,e_n}_{\beta,0}$)

$$\sum_{b \xrightarrow{i,w} b' \in B_0} w \bullet (\mathsf{Op}^{e_1,\dots,e_n}_{\beta,0})^{\gamma}(\top)(b') + \sum_{b \xrightarrow{i,w} b' \in B_j, j \neq 0} w \bullet e_j(b') \sqsubseteq$$

(I.H., assumption on e_j, e^0_j)

$$\sum_{b \xrightarrow{i,w} b' \in B_0} w \bullet \left((\mathsf{Op}^{e^0_1,\dots,e^0_n}_{\beta_0,0})^{\gamma}(\top)(b') + \sum_{k \in \{1,2,\dots\}} \mathsf{ext}^{\nu}_{\beta_k}(b'^k) \right) +$$

$$\sum_{b \xrightarrow{i,w} b' \in B_j, j \neq 0} w \bullet \left(e^0_j(b') + \sum_{k \in \{1,2,\dots\}} \mathsf{ext}^{\nu}_{\beta_k}(b'^k) \right) =$$

(distributivity of $\bullet$ over finite sums, definition of $\mathsf{Op}^{e^0_1,\dots,e^0_n}_{\beta_0,0}$ and $\mathsf{ext}^{\nu}_{\beta_k}(b^k)$)

$$(\mathsf{Op}^{e^0_1,\dots,e^0_n}_{\beta_0,0})^{\gamma+1}(\top)(b) + \sum_{k \in \{1,2,\dots\}} \mathsf{ext}^{\nu}_{\beta_k}(b^k)$$

- For α a limit ordinal, the statement follows from $(\mathsf{Op}^{e_{i+1},\dots,e_n}_{\beta,0})^{\alpha}(\top)$ and $(\mathsf{Op}^{e^0_{i+1},\dots,e^0_n}_{\beta_0,0})^{\alpha}(\top)$ being obtained as infima of decreasing chains.

This then yields the required statement for $i = 0$.

- The induction step is proved similarly, additionally making use of the induction hypothesis.

We have thus proves the inequality (9) in the *total* semiring $(S, \oplus, 0, \bullet, 1)$. This now gives $\mathsf{ext}_{\beta}(b_I) \sqsubseteq \mathsf{ext}_{\beta_0}(b^0_I) + \sum_{k \in \{1,2,\dots\}} \mathsf{ext}^{\nu}_{\beta_k}(b^k_I)$ in $(S, \oplus, 0, \bullet, 1)$. However, since the sum in the rhs *is* defined in $(S, +, 0, \bullet, 1)$ (it coincides with $\mu_\gamma[\mathcal{C}]$)), it follows that the same inequality also holds in $(S, +, 0, \bullet, 1)$. This concludes the proof. ◀

Theorem 42 yields characterisations of the notion of extent in all our example semirings.

▶ **Example 47.** When $(S, +, 0, \bullet, 1)$ is the boolean semiring, a state in a QPA has extent 0 iff it admits no accepting paths. When $(S, +, 0, \bullet, 1)$ is the probabilistic semiring, the extent of a state measures the likelihood of an accepting path. When $(S, +, 0, \bullet, 1)$ is the tropical semiring, the extent of a state gives the minimal cost of an accepting path from that state.

4 Model Checking Qualitative Properties in Quantitative Models

We now show how to use Theorem 42 to model check qualitative properties captured by F-automata against $(\mathsf{T}_S \circ F)$-coalgebras. When the F-automaton is non-deterministic, its product with a $(\mathsf{T}_S \circ F)$-coalgebra is only defined when the semiring is total. However, even if the product *is* defined, accepting paths through the product are not, in general, in one-to-one correspondence with paths through the coalgebra which conform to the automaton. For this, unambiguity of the automaton is required. This is why in what follows we restrict to qualitative properties captured by *unambiguous* F-automata. We first consider the case when the semiring is total, and then show how to extend our result to a partial semiring.

We instantiate Theorem 42 to the product of an unambiguous NPA (Definition 23) with a $(\mathsf{T}_S \circ F)$-coalgebra in order to prove the following result:

▶ **Theorem 48.** *Assume* $(S, +, 0, \bullet, 1)$ *is* total. *Let* (A, α, a_I, Ω) *with* $\mathsf{ran}(\Omega) \subseteq \{0, \ldots, n\}$ *be an unambiguous automaton, let* (C, γ, c_I) *be a pointed* $(\mathsf{T}_S \circ F)$*-coalgebra, and let* $(D, \delta, (c_I, a_I), \Omega)$ *be the product of* (C, γ, c_I) *and* (A, α, a_I, Ω) *(Definition 29). Then, the extent* $\mathsf{ext}_\delta : D \to S$ *of* $(D, \delta, (c_I, a_I), \Omega)$ *satisfies* $\mu_\gamma(\{\xi \in \mathsf{Paths}^\gamma_{c_I} \mid \xi \text{ accepted by } (A, \alpha, a_I, \Omega)\}) = \mathsf{ext}_\delta(c_I, a_I)$.

Proof. We have:

$$\mathsf{ext}_\delta(c_I, a_I) = \mu_\delta(\{\, \xi \in \mathsf{Paths}^\delta_{(c_I, a_I)} \mid \xi \text{ acc.} \,\}) = \mu_\gamma(\{\xi \in \mathsf{Paths}^\gamma_{c_I} \mid \xi \text{ accepted by } (A, \alpha, a_I, \Omega)\})$$

The first equality follows by Theorem 42, whereas the second equality follows by Lemma 31 and because measuring the sets of paths in question in δ, respectively γ, yields the same result (since weights of δ-transitions are inherited from γ). ◀

Theorem 48 thus states that, assuming that the automaton (A, α, a_I, Ω) is unambiguous, the extent of its product with a model (C, γ, c_I) can be used to compute the measure of the set of paths from c_I which conform to the automaton.

When the semiring S is partial, the product of (C, γ, c_I) and (A, α, a, Ω) is not always a $\mathsf{T}_S \circ F$-automaton. To deal with this, we view (C, γ, c_I) as a $\mathsf{T}_{S'} \circ F$-coalgebra (where $S' = (S, \oplus, 0, \bullet, 1)$ is as in Remark 8), to which Theorem 42 applies. However, in order to generalise Theorem 48 to *partial* semirings, we must additionally show that the S-valued measure of the set of paths from c in (C, γ) which are accepted by (A, α, a, Ω) coincides with the S'-valued measure of the same set of paths. The next lemma establishes this.

▶ **Lemma 49.** *Let* (C, γ, c_I) *be a pointed* $(\mathsf{T}_S \circ F)$*-coalgebra. Then,* $\mu^S_\gamma(P) = \mu^{S'}_\gamma(P)$ *for any measurable set* P *of paths from* c *in* (C, γ) *(where the superscripts of the resulting measures indicate the semiring these measures are valued into).*

Proof. We have:

$$\begin{aligned}
\mu^S_\gamma(P) &\overset{(\text{def. of } \mu^S_\gamma)}{=} \inf\{\, \sum_{C \in \mathcal{C}} \mu^S_\gamma(C) \mid \mathcal{C} \text{ is a countable, pairwise-disjoint cover for } P \,\} \\
&\overset{(*)}{=} \inf\{\, \sum_{C \in \mathcal{C}} \mu^{S'}_\gamma(C) \mid \mathcal{C} \text{ is a countable, pairwise-disjoint cover for } P \,\} \\
&\overset{(\text{def. of } \mu^{S'}_\gamma)}{=} \mu^{S'}_\gamma(P)
\end{aligned}$$

The equality $(*)$ above follows from the fact that all sums in the lhs are defined. ◀

Our second main result is now a direct consequence of Theorem 48 and Lemma 49.

▶ **Theorem 50.** *Let* $(S, +, 0, \bullet, 1)$ *be a partial semiring and let* $(S' = S, \oplus, 0, \bullet, 1)$ *be as in Remark 8. Let* (A, α, a, Ω) *be an unambiguous* F*-automaton, and let* (C, γ, c_I) *be a pointed* $(\mathsf{T}_S \circ F)$*-coalgebra. Finally, let* (D, δ, d, Ω) *be the product of* $(C, \iota \circ \gamma, c_I)$ *and* (A, α, a, Ω). *Then, the following holds:* $\mu^S_\gamma(\{\, \xi \in \mathsf{Paths}^\gamma_c \mid \xi \text{ accepted by } (A, \alpha, a, \Omega) \,\} = \mathsf{ext}^{S'}_\delta(c, a)$.

In other words, to measure the set of paths in a model (C, γ, c_I) which conform to a qualitative property captured by an unambiguous parity automaton (A, α, a, Ω), one can simply compute the extent of the product automaton, in the extended semiring $(S, \oplus, 0, \bullet, 1)$.

5 Conclusions

We provided a characterisation of the measure of the set of accepting paths of a QPA, as the solution of a nested system of equations. We also showed how to use this characterisation to model check qualitative linear-time properties against quantitative models. Future work will investigate computational results and the expressive power of unambiguous automata, and will use techniques from [3] to approximate nested extents.

References

1 Christel Baier, Luca de Alfaro, Vojtech Forejt, and Marta Kwiatkowska. Model checking probabilistic systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 963–999. Springer, 2018. doi:10.1007/978-3-319-10575-8_28.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

3 Paolo Baldan, Barbara König, Christina Mika-Michalski, and Tommaso Padoan. Fixpoint games on continuous lattices. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29, 2019. doi:10.1145/3302515.

4 Corina Cîrstea. Linear time logics – a coalgebraic perspective. arXiv:1612.07844.

5 Corina Cîrstea. From branching to linear time, coalgebraically. *Fundamenta Informaticae*, 150:1–28, 2017. doi:10.3233/FI-2017-1474.

6 Corina Cîrstea, Shunsuke Shimizu, and Ichiro Hasuo. Parity Automata for Quantitative Linear Time Logics. In F. Bonchi and B. König, editors, *7th Conference on Algebra and Coalgebra in Computer Science (CALCO 2017)*, volume 72 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:18, 2017. doi:10.4230/LIPIcs.CALCO.2017.7.

7 Thomas Colcombet. Forms of determinism for automata (invited talk). In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th – March 3rd, 2012, Paris, France*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1–23, 2012. doi:10.4230/LIPIcs.STACS.2012.1.

8 Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching metrics for quantitative transition systems. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming*, pages 97–109. Springer, 2004.

9 Marco Faella, Axel Legay, and Mariëlle Stoelinga. Model checking quantitative linear time logic. *Electr. Notes Theor. Comput. Sci.*, 220:61–77, 2008. doi:10.1016/j.entcs.2008.11.019.

10 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*, 2002. doi:10.1007/3-540-36387-4.

11 Ichiro Hasuo, Shunsuke Shimizu, and Corina Cîrstea. Lattice-theoretic progress measures and coalgebraic model checking. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016*, pages 718–732. ACM, 2016. doi:10.1145/2837614.2837673.

12 Daniel Hausmann and Lutz Schröder. Quasipolynomial computation of nested fixpoints. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS 2021, Proceedings*, volume 12651 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2021. doi:10.1007/978-3-030-72016-2_3.

13 Bart Jacobs. *Introduction to coalgebra*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016.

14 Claus Thrane, Uli Fahrenberg, and Kim Larsen. Quantitative analysis of weighted transition systems. *Journal of Logic and Algebraic Programming*, 79:689–703, 2010. doi:10.1016/j.jlap.2010.07.010.

15 Natsuki Urabe, Shunsuke Shimizu, and Ichiro Hasuo. Coalgebraic trace semantics for Büchi and parity automata. arXiv:1606.09399.

Realizing Continuity Using Stateful Computations

Liron Cohen
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Vincent Rahli
University of Birmingham, UK

Abstract

The principle of continuity is a seminal property that holds for a number of intuitionistic theories such as System T. Roughly speaking, it states that functions on real numbers only need approximations of these numbers to compute. Generally, continuity principles have been justified using semantical arguments, but it is known that the modulus of continuity of functions can be computed using effectful computations such as exceptions or reference cells. This paper presents a class of intuitionistic theories that features stateful computations, such as reference cells, and shows that these theories can be extended with continuity axioms. The modulus of continuity of the functionals on the Baire space is directly computed using the stateful computations enabled in the theory.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Constructive mathematics

Keywords and phrases Continuity, Stateful computations, Intuitionism, Extensional Type Theory, Constructive Type Theory, Realizability, Theorem proving, Agda

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.15

Supplementary Material *Model (Agda Formalization)*: `https://github.com/vrahli/opentt`

Funding *Liron Cohen*: This research was partially supported by Grant No. 2020145 from the United States-Israel Binational Science Foundation (BSF).

1 Introduction

Continuity is a seminal property in intuitionistic theories which contradicts classical mathematics but is generally accepted by constructivists. Roughly speaking, the principle states that functions on real numbers only need approximations of these numbers to compute. Brouwer, in particular, assumed his so-called *continuity principle for numbers* to derive that all real-valued functions on the unit interval are uniformly continuous [15, 11, 4, 5, 28]. The continuity principle for numbers, sometimes referred to as the weak continuity principle, states that all functions on the Baire space (i.e., $\mathcal{B} \coloneqq \mathsf{Nat}^{\mathsf{Nat}}$) have a modulus of continuity. More concretely, given a function F of type $\mathcal{B} \to \mathsf{Nat}$ and a function α of type $\mathcal{B}$, the principle states that $F(\alpha)$ can only depend on an initial segment of α, and the length of the smallest such segment is the modulus of continuity of F at α. This is standardly formalized as follows, where $\mathcal{B}_n \coloneqq \{x : \mathsf{Nat} \mid x < n\} \to \mathsf{Nat}$ is the set of finite sequences of length n:

$$\mathtt{WCP} = \mathbf{\Pi} F{:}\mathcal{B} \to \mathsf{Nat}.\mathbf{\Pi}\alpha{:}\mathcal{B}.{\downarrow}\,\mathbf{\Sigma} n{:}\mathsf{Nat}.\mathbf{\Pi}\beta{:}\mathcal{B}.(\alpha{=}\beta{\in}\mathcal{B}_n) \to (F(\alpha){=}F(\beta){\in}\mathsf{Nat})$$

A number of theories have been shown to satisfy Brouwer's continuity principle, or uniform variants, such as N-HA$^{\omega}$ by Troelstra [29, p.158], MLTT by Coquand and Jaber [8, 9], System T by Escardó [13], CTT by Rahli and Bickford [24], BTT by Baillon, Mahboubi and Pedrot [3], to cite only a few (see Sec. 5 for further details). These proofs often rely on a semantical forcing-based approach [8, 9], where the forcing conditions capture the amount of information needed when applying a function to a sequence in the Baire space, or through suitable models that internalize (C-Spaces in [34]) or exhibit continuous behavior (e.g., dialogue trees in [13, 3]).

Not only can functions on the Baire space be proved to be continuous, but using effectful computations, as for example described in [20], one can *compute* the modulus of continuity of such a function. However, as shown for example by Kreisel [16, p.154], Troelstra [30, Thm.IIA], and Escardó and Xu [12, 33], continuity is not an extensional property in the sense that two equal functions might have different moduli of continuity. Therefore, to realize continuity, the existence of a modulus of continuity has to be truncated as explained, e.g., in [12, 33, 24, 25], which is what the $\downarrow$ operator achieves in `WCP`. Following the effectful approach, continuity was shown to be realizable in [24, 25] using exceptions.

Instead of using exceptions, a more straightforward way to compute the modulus of continuity of a function on the Baire space is to use reference cells. This was explained, e.g., in [20], where the use of references can be seen as the programming counterparts of the more logical forcing conditions. The computation using references is more efficient than when using exceptions as it allows computing the modulus of continuity of a function F at a point α simply by executing F on α, while recording the highest argument that α is applied to, while using exceptions requires repeatedly searching for the modulus of continuity.

Following this line of work, in this paper we show how to use stateful computations to realize a continuity principle. This allows deriving constructive type theories that include continuity axioms where the modulus of continuity is internalized in the sense that it is computed by an expression of the underlying programming language. Concretely, we do so for $\mathrm{TT}^{\Box}_{\mathcal{C}}$ [6], which is a family of extensional type theories parameterized by a type modality $\Box$, and a choice type $\mathcal{C}$, which are presented in more details in Sec. 2. More precisely, we prove in this paper that all $\mathrm{TT}^{\Box}_{\mathcal{C}}$ functions are continuous for some instances of $\Box$ and $\mathcal{C}$: namely for "non-empty" equality modalities, and reference-like stateful choice operators. Our proof is for a variant of the weak continuity principle (see Thm. 13), which we show to be inhabited by a program that relies on a choice operator to keep track of the modulus of continuity of a given function, following Longley's method [20]. This variant is restricted to "pure" functions F, α, and β without side effects, and Sec. 4.1 discusses issues arising with impure functions.

Roadmap. After recalling in Sec. 2 the main aspects of $\mathrm{TT}^{\Box}_{\mathcal{C}}$ that are relevant to the results presented in this paper, Sec. 3 instantiates and extends $\mathrm{TT}^{\Box}_{\mathcal{C}}$ with additional components, which are, in turn, used in Sec. 4 to validate continuity using stateful computations. One key contribution of this paper, discussed in Sec. 3, is the fact that $\mathrm{TT}^{\Box}_{\mathcal{C}}$ now allows computations to modify the current world, which is accounted for in its forcing interpretation. Another key contribution, discussed in Sec. 4, is the internalization of the modulus of continuity of functions, in the sense that it can be computed by a $\mathrm{TT}^{\Box}_{\mathcal{C}}$ expression and used to validate the continuity principle. Finally, Sec. 5 concludes and discusses the related work on continuity.

2 Background

This section recalls $\mathrm{TT}^{\Box}_{\mathcal{C}}$, a family of type theories parameterized by a choice operator $\mathcal{C}$, and a metatheoretical modality $\Box$, which allows typing the choice operators. See [6] for further details. The choice operators are time-progressing elements that we will in particular instantiate with references. Sec. 3 carves out a sub-family for which we can validate computationally relevant continuity rules as shown in Sec. 4.

2.1 Metatheory

Our metatheory is Agda's type theory [1]. The results presented in this paper have been formalized in Agda, and the formalization is available here: `https://github.com/vrahli/opentt/`. We use $\forall, \exists, \wedge, \vee, \to, \neg$ in place of Agda's logical connectives in this paper. Agda

$$
\begin{array}{llllll}
v \in \mathsf{Value} ::= & vt & \text{(type)} & \mid \lambda x.t & \text{(lambda)} & \mid \star \quad \text{(constant)} \\
& \mid \underline{n} & \text{(number)} & \mid \mathtt{inl}(t) & \text{(left injection)} & \boxed{\mid \delta \quad \text{(choice name)}} \\
& \mid \langle t_1, t_2 \rangle & \text{(pair)} & \mid \mathtt{inr}(t) & \text{(right injection)} & \\
vt \in \mathsf{Type} ::= & \mathbf{\Pi} x{:}t_1.t_2 & \text{(product)} & \mid \{x : t_1 \mid t_2\} & \text{(set)} & \mid t_1 + t_2 \quad \text{(disjoint union)} \\
& \mid \mathbf{\Sigma} x{:}t_1.t_2 & \text{(sum)} & \mid t_1 = t_2 \in t & \text{(equality)} & \boxed{\mid \downarrow t \quad \text{(time truncation)}} \\
& \mid \mathbb{U}_i & \text{(universe)} & \mid \mathsf{Nat} & \text{(numbers)} & \\
t \in \mathsf{Term} ::= & x & \text{(variable)} & \mid v & \text{(value)} & \boxed{\mid\ !t \quad \text{(read)}} \\
& \mid t_1\ t_2 & \text{(application)} & \mid \mathtt{fix}(t) & \text{(fixpoint)} & \mid \mathtt{let}\ x, y = t_1\ \mathtt{in}\ t_2 \quad \text{(pair destructor)} \\
& \mid \mathtt{case}\ t\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_1 \mid \mathtt{inr}(y) \Rightarrow t_2 & & & & \text{(injection destructor)}
\end{array}
$$

$$
\begin{array}{ll}
(\lambda x.t)\ u \ \mapsto_{\boxed{w}}\ t[x \backslash u] & \mathtt{let}\ x, y = \langle t_1, t_2 \rangle\ \mathtt{in}\ t \mapsto_{\boxed{w}} t[x \backslash t_1; y \backslash t_2] \\
\mathtt{fix}(v) \ \mapsto_{\boxed{w}}\ v\ \mathtt{fix}(v) & \mathtt{case}\ \mathtt{inl}(t)\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_1 \mid \mathtt{inr}(y) \Rightarrow t_2 \mapsto_{\boxed{w}} t_1[x \backslash t] \\
\boxed{!\delta \mapsto_w \mathsf{choice?}(w, \delta)} & \mathtt{case}\ \mathtt{inr}(t)\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_1 \mid \mathtt{inr}(y) \Rightarrow t_2 \mapsto_{\boxed{w}} t_2[y \backslash t]
\end{array}
$$

Figure 1 Core syntax (above) and small-step operational semantics (below).

provides an hierarchy of types annotated with universe labels which we omit for simplicity. Following Agda's terminology, we refer to an Agda type as a *set*, and reserve the term *type* for $\mathrm{TT}^{\Box}_{\mathcal{C}}$'s types. We use $\mathbb{P}$ as the type of sets that denote propositions; $\mathbb{N}$ for the set of natural numbers; and $\mathbb{B}$ for the set of Booleans true and false. We use induction-recursion to define the forcing interpretation in Sec. 3.2, where we use function extensionality to interpret universes. We do not discuss this further here and the interested reader is referred to forcing.lagda in the Agda code for further details.

2.2 Worlds

To capture the time progression notion which underlines choice operators, $\mathrm{TT}^{\Box}_{\mathcal{C}}$ is parameterized by a Kripke frame [18, 19] defined as follows:

▶ **Definition 1** (Kripke Frame). *A Kripke frame consists of a set of* worlds $\mathcal{W}$ *equipped with a reflexive and transitive binary relation* $\sqsubseteq$.

Let w range over $\mathcal{W}$. We sometimes write $w' \sqsupseteq w$ for $w \sqsubseteq w'$. Let $\mathcal{P}_w$ be the collection of predicates on world extensions, i.e., functions in $\forall w' \sqsupseteq w.\mathbb{P}$. Note that due to $\sqsubseteq$'s transitivity, if $P \in \mathcal{P}_w$ then for every $w' \sqsupseteq w$ it naturally extends to a predicate in $\mathcal{P}_{w'}$. We further define the following notations for quantifiers. $\forall^{\sqsubseteq}_w(P)$ states that $P \in \mathcal{P}_w$ is true for all extensions of w, i.e., $P\ w'$ holds in all worlds $w' \sqsupseteq w$. $\exists^{\sqsubseteq}_w(P)$ states that $P \in \mathcal{P}_w$ is true at an extension of w, i.e., $P\ w'$ holds for some world $w' \sqsupseteq w$. For readability, we sometime write $\forall^{\sqsubseteq}_w(w'.P)$ (or $\exists^{\sqsubseteq}_w(w'.P)$) instead of $\forall^{\sqsubseteq}_w(\lambda w'.P)$ (or $\exists^{\sqsubseteq}_w(\lambda w'.P)$), respectively.

2.3 $\mathrm{TT}^{\Box}_{\mathcal{C}}$'s Syntax and Operational Semantics

Fig. 1 recalls $\mathrm{TT}^{\Box}_{\mathcal{C}}$'s syntax and operational semantics, where the blue boxes highlight the time-related components, and where x belongs to a set of variables Var. For simplicity, numbers are considered to be primitive. The constant $\star$ is there for convenience, and is used in place of a term, when the particular term used is irrelevant. Terms are evaluated according to the operational semantics presented in Fig. 1's lower part. In what follows, we use all letters as metavariables for terms. Let $t[x \backslash u]$ stand for the capture-avoiding substitution of all the free occurrences of x in t by u.

Types are syntactic forms that are given semantics in Sec. 3.2 via a forcing interpretation. The type system contains standard types such as dependent products of the form $\mathbf{\Pi} x{:}t_1.t_2$ and dependent sums of the form $\mathbf{\Sigma} x{:}t_1.t_2$. For convenience we write $t_1 \to t_2$ for the non-dependent $\mathbf{\Pi}$ type; `True` for $\underline{0}{=}\underline{0}{\in}\mathsf{Nat}$; `False` for $\underline{0}{=}\underline{1}{\in}\mathsf{Nat}$; and $\neg T$ for $(T \to \texttt{False})$.

Fig. 1's lower part presents $\mathrm{TT}^{\Box}_{\mathcal{C}}$'s small-step operational semantics, where $t_1 \mapsto_w t_2$ expresses that t_1 reduces to t_2 in one step of computation *w.r.t. the world* w. We omit the congruence rules that allow computing within terms such as: if $t_1 \mapsto_w t_2$ then $t_1(u) \mapsto_w t_2(u)$. We denote by $\Downarrow$ the reflexive transitive closure of $\mapsto$, i.e., $a \Downarrow_w b$ states that a computes to b in ≥ 0 steps. We also write $a \Downarrow\!\!\Downarrow_w b$ if a computes to b in all extensions of w, i.e., if $\forall^{\sqsubseteq}_w(w'.a \Downarrow_{w'} b)$. We write $\sim_w$ for the symmetric and transitive closure of $\Downarrow_w$.

$\mathrm{TT}^{\Box}_{\mathcal{C}}$ includes time-progressing notions that rely on worlds to record choices and provides operators to access the choices stored in a world, which we now recall. Choices are referred to through their names. A concrete example of such choices are reference cells in programming languages, where a variable name pointing to a reference cell is the name of the corresponding reference cell. To this end, $\mathrm{TT}^{\Box}_{\mathcal{C}}$'s computation system is parameterized by a set $\mathcal{N}$ of *choice names*, that is equipped with a decidable equality, and an operator that given a list of names, returns a name not in the list. This can be given by, e.g., nominal sets [23]. In what follows we let δ range over $\mathcal{N}$, and take $\mathcal{N}$ to be $\mathbb{N}$ for simplicity. $\mathrm{TT}^{\Box}_{\mathcal{C}}$ is further parameterized over abstract operators and properties recalled in Defs. 2 and 4–6, which we show how to instantiate in Ex. 7. Definitions such as Def. 2 provide axiomatizations of operators, and in addition informally indicate their intended use. Choices are defined abstractly as follows:

▶ **Definition 2** (Choices). *Let* $\mathcal{C} \subseteq \mathsf{Term}$ *be a set of* choices,[1] *and let* κ *range over* $\mathcal{C}$. *We say that a computation system contains* $\langle \mathcal{N}, \mathcal{C} \rangle$*-choices if there exists a partial function* $\mathsf{choice?} \in \mathcal{W} \to \mathcal{N} \to \mathcal{C}$. *Given* $w \in \mathcal{W}$ *and* $\delta \in \mathcal{N}$, *the returned choice, if it exists, is meant to be the last choice made for* δ *according to* w. $\mathcal{C}$ *is said to be* non-trivial *if it contains two values* $\boldsymbol{\kappa_0}$ *and* $\boldsymbol{\kappa_1}$, *which are computationally different, i.e., such that* $\neg(\boldsymbol{\kappa_0} \sim_w \boldsymbol{\kappa_1})$ *for all* w.

A choice name δ can be used in a computation to access choices from a world as follows: $!\delta \mapsto_w \mathsf{choice?}(w, \delta)$ (as shown in Fig. 1). This allows getting the last δ-choice from the current world w. The quotienting type operator $\wr$ allows assigning types to such expressions that compute to different values in different worlds. For example, as defined in Fig. 2, while Nat is the type of expressions that when they compute to $\underline{i}$ in w_1 must also compute to $\underline{i}$ in $w_2 \sqsupseteq w_1$, $\wr\mathsf{Nat}$ is the type of expressions that can compute to $\underline{i}$ in w_1 and to another number $\underline{j}$ in $w_2 \sqsupseteq w_1$. This is used to assign types to computations involving choices. For example, $!\delta$ inhabits $\wr\mathsf{Nat}$ when its choices are numbers.

Note that the above definition of $\mathsf{choice?}$ is a slight simplification of the more general notion of choices presented in [6]. There, the $\mathsf{choice?}$ function was of type $\mathcal{W} \to \mathcal{N} \to \mathbb{N} \to \mathcal{C}$. The additional $\mathbb{N}$ component enables a more general notion of choice operators, including ones in which the history is recorded. In references, which is the notion of choices we especially focus on in this paper, one only maintains the latest update and so the $\mathbb{N}$ component becomes moot. Thus, for simplicity of presentation, we elide the $\mathbb{N}$ component in this paper, but full details are available in the Agda implementation.

$\mathrm{TT}^{\Box}_{\mathcal{C}}$ also includes the notion of a *restriction*, which allows assuming that the choices made for a given choice name all satisfy a pre-defined constraint. Here again we simplify the concept for choices without history tracking.

[1] To guarantee that $\mathcal{C} \subseteq \mathsf{Term}$, one can for example extend the syntax to include a designated constructor for choices, or require a coercion $\mathcal{C} \to \mathsf{Term}$. We opted for the latter in our formalization.

▶ **Definition 3** (Restrictions). *A restriction* $r \in \mathsf{Res}$ *is a pair* $\langle res, d \rangle$ *consisting of a function* $res \in \mathcal{C} \to \mathbb{P}$ *and a default choice* $d \in \mathcal{C}$ *such that* $(res\ d)$ *holds. Given such a pair* r*, we write* $r_{\cdot \mathsf{d}}$ *for* d*.*

Intuitively, *res* specifies a restriction on the choices that can be made at any point in time and d provides a default choice that meets this restriction (e.g., for reference cells, this default choice is used to initialize a cell). For example, the restriction $\langle \lambda\kappa.\kappa \in \mathbb{N}, 0 \rangle$ requires choices to be numbers and provides 0 as a default value. To reason about restrictions, we require the existence of a "compatibility" predicate as follows.

▶ **Definition 4** (Compatibility). *We say that* $\mathcal{C}$ *is* compatible *if there exists a predicate* $\mathsf{comp} \in \mathcal{N} \to \mathcal{W} \to \mathsf{Res} \to \mathbb{P}$*, intended to guarantee that restrictions are satisfied, and which is preserved by* $\sqsubseteq$*:* $\forall(\delta : \mathcal{N})(w_1, w_2 : \mathcal{W})(r : \mathsf{Res}).w_1 \sqsubseteq w_2 \to \mathsf{comp}(\delta, w_1, r) \to \mathsf{comp}(\delta, w_2, r)$.

$\mathrm{TT}^{\square}_{\mathcal{C}}$ further requires the ability to create new choice names as follows.

▶ **Definition 5** (Extendability). *We say that* $\mathcal{C}$ *is* extendable *if there exists a function* $\nu\mathcal{C} \in \mathcal{W} \to \mathcal{N}$*, where* $\nu\mathcal{C}(w)$ *is intended to return a new choice name not present in* w*, and a function* $\mathsf{start}\nu\mathcal{C} \in \mathcal{W} \to \mathsf{Res} \to \mathcal{W}$*, where* $\mathsf{start}\nu\mathcal{C}(w, r)$ *is intended to return an extension of* w *with the new choice name* $\nu\mathcal{C}(w)$ *with restriction* r*, satisfying the following properties:*

- *Starting a new choice extends the current world:* $\forall(w : \mathcal{W})(r : \mathsf{Res}).w \sqsubseteq \mathsf{start}\nu\mathcal{C}(w, r)$
- *Initially, the only possible choice is the default value of the given restriction, i.e.:* $\forall(r : \mathsf{Res})(w : \mathcal{W})(\kappa : \mathcal{C}).\mathsf{choice?}(\mathsf{start}\nu\mathcal{C}(w, r), \nu\mathcal{C}(w)) = \kappa \to \kappa = r_{\cdot \mathsf{d}}$
- *A choice is initially compatible with its restriction:* $\forall(w : \mathcal{W})(r : \mathsf{Res}).\mathsf{comp}(\nu\mathcal{C}(w), \mathsf{start}\nu\mathcal{C}(w, r), r)$

Lastly, $\mathrm{TT}^{\square}_{\mathcal{C}}$ requires the ability to update a choice as follows.

▶ **Definition 6** (Mutability). *We say that* $\mathcal{C}$ *is* mutable *if there exists a function* $\mathsf{update} \in \mathcal{W} \to \mathcal{N} \to \mathcal{C} \to \mathcal{W}$ *such that if* $w \in \mathcal{W}$*,* $\delta \in \mathcal{N}$*,* $\kappa \in \mathcal{C}$*, then* $w \sqsubseteq \mathsf{update}(w, \delta, \kappa)$.

From this point on, we will only discuss choices C that are compatible, extendable and mutable. The abstract notion of choice operators has many concrete instances. This paper focuses on one concrete instance – mutable references.

▶ **Example 7** (References). Reference cells, which are values that allow a program to indirectly access a particular object, are choice operators since they can point to different objects over their lifetime. Formally, we define references to numbers, Ref, as follows (see worldInstanceRef.lagda for details):

Non-trivial Choices Let $\mathcal{N} \coloneqq \mathbb{N}$ and $\mathcal{C} \coloneqq \mathbb{N}$, which is non-trivial, e.g., take $\boldsymbol{\kappa_0} \coloneqq \underline{0}$ and $\boldsymbol{\kappa_1} \coloneqq \underline{1}$.

Worlds Worlds are lists of cells, where a cell is a quadruple of (1) a choice name, (2) a restriction, (3) a choice, and (4) a Boolean indicating whether the cell is mutable. $\sqsubseteq$ is the reflexive transitive closure of two operations that allow (i) creating a new reference cell, and (ii) updating an existing reference cell. We define $\mathsf{choice?}(w, \delta)$ so that it simply accesses the content of the δ cell in w.

Compatible $\mathsf{comp}(\delta, w, r)$ states that a reference cell named δ with restriction r was created in the world w (using operation of type (i) described above), and that the current value of the cell satisfies r.

Extendable $\nu\mathcal{C}(w)$ returns a reference name not occurring in w; and $\mathsf{start}\nu\mathcal{C}(w, r)$ adds a new reference cell to w with name $\nu\mathcal{C}(w)$ and restriction r (using operation of type (i) mentioned above).

Mutable $\mathsf{update}(w, \delta, \kappa)$ updates the reference δ with the choice κ if δ occurs in w, and otherwise returns w (using operation of type (ii) mentioned above).

3 Instantiating $\mathrm{TT}_{\mathcal{C}}^{\square}$

To validate continuity, we need to internalize some semantical properties of $\mathrm{TT}_{\mathcal{C}}^{\square}$ that were introduced in [6] and recalled in Sec. 2. Concretely, we instantiate (and extend) $\mathrm{TT}_{\mathcal{C}}^{\square}$ with the following components, which are formally defined next.

- An operator that allows us to make a choice. This has far-reaching consequences, as a computation can now modify its current world. We generalize $\mathrm{TT}_{\mathcal{C}}^{\square}$'s semantics accordingly. This is an internalization of the mutability requirement.
- An operator to generate a "fresh" choice name. This is an internalization of the extendability requirement.
- A type that states the "purity" of an expression, i.e., that the expression has no side effects. This will allow us to formalize the variant of the continuity principle we validate. Sec. 4.1 provides further details.

3.1 Syntax & Operational Semantics

We extend $\mathrm{TT}_{\mathcal{C}}^{\square}$'s syntax as follows:

$$
\begin{aligned}
t \in \mathsf{Term} \ &::= \cdots \mid \boxed{\mathtt{choose}(t_1, t_2)} \mid \boxed{\boldsymbol{\nu} x.t} \mid \mathtt{let}\ x = t_1\ \mathtt{in}\ t_2 \\
&\quad \mid \mathtt{if}\ t_1 < t_2\ \mathtt{then}\ t_3\ \mathtt{else}\ t_4 \mid t_1 + t_2 \\
vt \in \mathsf{Type} \ &::= \cdots \mid \boxed{\mathtt{pure}} \mid t_1 \cap t_2 \mid \lfloor t \rfloor
\end{aligned}
$$

As in Sec. 2.3, the blue boxes highlight the time-related component. The term $\mathtt{pure}$ is the type of "pure" terms, i.e. terms that do no contain choice names. The term $t_1 \cap t_2$ is an intersection type, which is inhabited by the inhabitants of both t_1 and t_2. Finally, $\lfloor t$ turns a type t into a subsingleton type that equates all elements of t. The expression $\mathtt{choose}(\delta, t)$ makes the δ-choice t; while $\boldsymbol{\nu} x.t$ creates a "fresh" choice name w.r.t. t, thereby internalizing the notion of extendability presented in Def. 5. The term $\mathtt{let}\ x = t_1\ \mathtt{in}\ t_2$ is a call-by-value operator that allows evaluating t_1 to a value before proceeding with t_2. We write $t_1; t_2$ for $\mathtt{let}\ x = t_1\ \mathtt{in}\ t_2$ where x does not occur free in t_2.

We extend $\mathrm{TT}_{\mathcal{C}}^{\square}$'s operational semantics as follows. We turn the ternary relation $a \Downarrow_w b$ into a four-place relations $a \Downarrow_{w_2}^{w_1} b$ which captures that a computes to b starting from the world w_1 and updating it so that the resulting world is w_2 at the end of the computation. Most computations do not modify the current world except $\mathtt{choose}(t_1, t_2)$.

$$
\begin{array}{ll}
(\lambda x.t)\ u \ \mapsto_w^w \ t[x \backslash u] & \mathtt{let}\ x, y = \langle t_1, t_2 \rangle\ \mathtt{in}\ t \mapsto_w^w t[x \backslash t_1; y \backslash t_2] \\
\mathtt{fix}(v) \ \mapsto_w^w \ v\ \mathtt{fix}(v) & \mathtt{case}\ \mathtt{inl}(t)\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_1 \mid \mathtt{inr}(y) \Rightarrow t_2 \mapsto_w^w t_1[x \backslash t] \\
!\delta \mapsto_w^w \mathsf{choice?}(w, \delta) & \mathtt{case}\ \mathtt{inr}(t)\ \mathtt{of}\ \mathtt{inl}(x) \Rightarrow t_1 \mid \mathtt{inr}(y) \Rightarrow t_2 \mapsto_w^w t_2[y \backslash t]
\end{array}
$$

In addition we now have the following computations:

$$
\begin{array}{ll}
\mathtt{if}\ \underline{n} < \underline{m}\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 & \mapsto_w^w t_1, \text{if } n < m \\
\mathtt{if}\ \underline{n} < \underline{m}\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 & \mapsto_w^w t_2, \text{if } m \leq n \\
\underline{n} + \underline{m} & \mapsto_w^w \underline{n+m} \\
\mathtt{let}\ x = v\ \mathtt{in}\ t_2 & \mapsto_w^w t_2[x \backslash v]
\end{array}
$$

The semantics of $\mathtt{choose}(t_1, t_2)$ is defined as follows:

$$
\mathtt{choose}(\delta, t) \mapsto_{\mathsf{update}(w,\delta,t)}^{w} \star
$$

Choosing a δ-choice t using $\texttt{choose}(\delta, t)$ results in a corresponding update of the current world, namely $\mathsf{update}(w, \delta, t)$. The computation returns $\star$, which is reminiscent of reference updates in OCaml for example, which are of type `unit`. As mentioned in Def. 2, we require $\mathcal{C} \subseteq \mathsf{Term}$ so that choices can be included in computations. In addition, because $\mathsf{update} \in \mathcal{W} \to \mathcal{N} \to \mathcal{C} \to \mathcal{W}$, for $\mathsf{update}(w, \delta, t)$ to be well-defined for $t \in \mathsf{Term}$, we require a coercion from Term to $\mathcal{C}$ so that t can be turned into a choice, and update can be applied to that choice. This coercion is left implicit for readability. We further require that applying this coercion to a choice κ returns κ, which is used to validate the assumption Ass_3 discussed in Sec. 4.2.

▶ **Example 8.** We saw in Ex. 7, that $\mathcal{C}$ can for example be instantiated to be $\mathbb{N}$. A coercion from Term to $\mathcal{C}$ can then turn $\underline{n}$ into n and all other terms to 0, which satisfies the requirement that choices are mapped to the same choices.

Finally, we describe how $\boldsymbol{\nu} x.t$ computes. Intuitively, it selects a "fresh" choice name δ and instantiate the variable x with δ. Formally, it computes as follows:

$$\boldsymbol{\nu} x.t \mapsto^{w}_{\mathsf{start}\nu\mathcal{C}(w,\mathtt{r})} t[x \backslash \nu\mathcal{C}(w)]$$

where $\mathtt{r}$ is the restriction $\langle \lambda c.(c \in \mathbb{N}), 0 \rangle$, which constrains the choices to be numbers, with default value 0. Other restrictions could be supported, for example by adding different $\boldsymbol{\nu}$ symbols to the language and by selecting during computation the appropriate restriction based on the $\boldsymbol{\nu}$ operator at hand. This is however left for future work as we especially focus here on the choices presented in Ex. 8.

▶ Remark 9 (Freshness). The fresh operator used in [24] computes $\boldsymbol{\nu} x.a$ by reducing a to b, and then returning $\boldsymbol{\nu} x.b$, thereby never generating new fresh names. As opposed to that fresh operator, which was based on nominal sets, the one introduced in this paper cannot put back the "fresh" constructor at each step of the small step derivation, otherwise a multi-step computation would not be able to use a choice name to keep track of the modulus of continuity of a function across multiple computation steps by recording it in the current world. One consequence of this is that this fresh operator cannot guarantee that it generates a truly "fresh" name that does not occur anywhere else (therefore, it does not satisfy the nominal axioms). For example $(\boldsymbol{\nu} x.x)\ \delta$ might generate the name δ because it does not occur in the local expression $\boldsymbol{\nu} x.x$.

Formally, $a \Downarrow^{w_1}_{w_2} b$ is the reflexive and transitive closure of $\mapsto$, i.e., it holds if a in world w_1 computes to b in world w_2 in 0 or more steps. Thanks to the properties of $\mathsf{start}\nu\mathcal{C}$ presented in Def. 5, and the properties of update presented in Def. 6, computations respect $\sqsubseteq$:

▶ **Lemma 10** (Computations respect $\sqsubseteq$). *If $a \Downarrow^{w_1}_{w_2} b$ then $w_1 \sqsubseteq w_2$.*

3.2 Forcing Interpretation

$\mathrm{TT}^{\Box}_{\mathcal{C}}$'s semantics is similar to the one presented in [6], which we recall and extend in Fig. 2. It is interpreted via a forcing interpretation in which the forcing conditions are worlds. This interpretation is defined using induction-recursion as follows: (1) the inductive relation $w \vDash T_1 \equiv T_2$ expresses type equality in the world w; (2) the recursive function $w \vDash t_1 \equiv t_2 \in T$ expresses equality in a type. We further use the following abstractions: $w \vDash \mathtt{type}(T)$ for $w \vDash T \equiv T$, $w \vDash t \in T$ for $w \vDash t \equiv t \in T$, and $w \vDash T$ for $\exists(t : \mathsf{Term}).w \vDash t \in T$. Note that a major difference is that while $a \Downarrow_w b$ is still defined as $\forall^{\sqsubseteq}_{w}(w'.a \Downarrow_{w'} b)$ as in [6], $a \Downarrow_{w'} b$ is now defined as $\exists(w'' : \mathcal{W}).a \Downarrow^{w'}_{w''} b$ to account for the fact that computations can now update the current

Numbers:

- $w \vDash \mathsf{Nat}\equiv\mathsf{Nat} \iff \mathtt{True}$
- $w \vDash t\equiv t'\in\mathsf{Nat} \iff \Box_w(w'.\exists(n:\mathbb{N}).t \Downarrow_{w'} \underline{n} \wedge t' \Downarrow_{w'} \underline{n})$

Products:

- $w \vDash \mathbf{\Pi}x{:}A_1.B_1\equiv\mathbf{\Pi}x{:}A_2.B_2 \iff \mathsf{Fam}_w(A_1,A_2,B_1,B_2)$
- $w \vDash f\equiv g\in\mathbf{\Pi}x{:}A.B \iff \Box_w(w'.\forall(a_1,a_2:\mathsf{Term}).w' \vDash a_1\equiv a_2\in A \to w' \vDash f\ a_1\equiv g\ a_2\in B[x\backslash a_1])$

Sums:

- $w \vDash \mathbf{\Sigma}x{:}A_1.B_1\equiv\mathbf{\Sigma}x{:}A_2.B_2 \iff \mathsf{Fam}_w(A_1,A_2,B_1,B_2)$
- $w \vDash p_1\equiv p_2\in\mathbf{\Sigma}x{:}A.B \iff \Box_w(w'.\exists(a_1,a_2,b_1,b_2 : \mathsf{Term}).w' \vDash a_1\equiv a_2\in A \wedge w' \vDash b_1\equiv b_2\in B[x\backslash a_1] \wedge p_1 \Downarrow_{w'} \langle a_1,b_1\rangle \wedge p_2 \Downarrow_{w'} \langle a_2,b_2\rangle)$

Sets:

- $w \vDash \{x:A_1 \mid B_1\}\equiv\{x:A_2 \mid B_2\} \iff \mathsf{Fam}_w(A_1,A_2,B_1,B_2)$
- $w \vDash a_1\equiv a_2\in\{x:A \mid B\} \iff \Box_w(w'.\exists(b_1,b_2:\mathsf{Term}).w' \vDash a_1\equiv a_2\in A \wedge w' \vDash b_1\equiv b_2\in B[x\backslash a_1])$

Disjoint unions:

- $w \vDash A_1+B_1\equiv A_2+B_2 \iff w \vDash A_1\equiv A_2 \wedge w \vDash B_1\equiv B_2$
- $w \vDash a_1\equiv a_2\in A+B \iff \Box_w(w'.\exists(u,v:\mathsf{Term}).(a_1 \Downarrow_{w'} \mathtt{inl}(u) \wedge a_2 \Downarrow_{w'} \mathtt{inl}(v) \wedge w' \vDash u\equiv v\in A) \vee (a_1 \Downarrow_{w'} \mathtt{inr}(u) \wedge a_2 \Downarrow_{w'} \mathtt{inr}(v) \wedge w' \vDash u\equiv v\in B))$

Equalities:

- $w \vDash (a_1{=}b_1\in A)\equiv(a_2{=}b_2\in B) \iff w \vDash A\equiv B \wedge \forall^{\sqsubseteq}_w(w'.w' \vDash a_1\equiv a_2\in A) \wedge \forall^{\sqsubseteq}_w(w'.w' \vDash b_1\equiv b_2\in B)$
- $w \vDash a_1\equiv a_2\in(a{=}b\in A) \iff \Box_w(w'.w' \vDash a\equiv b\in A)$
  (note that a_1 and a_2 can be any term here)

Time-Quotiented types:

- $w \vDash \wr A\equiv\wr B \iff w \vDash A\equiv B$
- $w \vDash a\equiv b\in\wr A \iff \Box_w(w'.(\lambda a,b.\exists(c,d:\mathsf{Value}).a \sim_w c \wedge b \sim_w d \wedge w \vDash c\equiv d\in A)^+\ a\ b)$

Subsingletons:

- $w \vDash \lfloor A\equiv\lfloor B \iff w \vDash A\equiv B$
- $w \vDash a\equiv b\in\lfloor A \iff \Box_w(w'.w' \vDash a\equiv a\in A \wedge w' \vDash b\equiv b\in A)$

Purity:

- $w \vDash \mathtt{pure}\equiv\mathtt{pure} \iff \mathtt{True}$
- $w \vDash a_1\equiv a_2\in\mathtt{pure} \iff \mathtt{namefree}(a_1) \wedge \mathtt{namefree}(a_2)$

Binary intersections:

- $w \vDash A_1 \cap B_1\equiv A_2 \cap B_2 \iff w \vDash A_1\equiv A_2 \wedge w \vDash B_1\equiv B_2$
- $w \vDash a_1\equiv a_2\in A \cap B \iff \Box_w(w'.w' \vDash a_1\equiv a_2\in A \wedge w' \vDash a_1\equiv a_2\in B)$

Modality closure:

- $w \vDash T_1\equiv T_2 \iff \Box_w(w'.\exists(T_1',T_2':\mathsf{Term}).T_1 \Downarrow_{w'} T_1' \wedge T_2 \Downarrow_{w'} T_2' \wedge w' \vDash T_1'\equiv T_2')$
- $w \vDash t_1\equiv t_2\in T \iff \Box_w(w'.\exists(T':\mathsf{Term}).T \Downarrow_{w'} T' \wedge w' \vDash t_1\equiv t_2\in T')$

Figure 2 Forcing Interpretation.

world. We also define $a \Downarrow_{!w} b$ as $\forall^{\sqsubseteq}_w(w'.a \Downarrow^{w'}_{w'} b)$, capturing the fact that the computation does not change the initial world (this is used in Thm. 12). Fig. 2 defines in particular the semantics of `pure`, which is inhabited by name-free terms, where $\mathtt{namefree}(t)$ is defined recursively over t and returns false iff t contains a choice name δ or a fresh operator of the form $\boldsymbol{\nu}x.t$. There, we write R^+ for R's transitive closure, which is used to prove the transitivity of time-quotiented types, in the sense of Thm. 12. We also write $\mathsf{Fam}_w(A_1,A_2,B_1,B_2)$ for $w \vDash A_1\equiv A_2 \wedge \forall^{\sqsubseteq}_w(w'.\forall(a_1,a_2:\mathsf{Term}).w' \vDash a_1\equiv a_2\in A_1 \to w' \vDash B_1[x\backslash a_1]\equiv B_2[x\backslash a_2])$.

This forcing interpretation is parameterized by a family of abstract modalities $\Box$, which we sometimes refer to simply as a modality, which is a function that takes a world w to its modality $\Box_w \in \mathcal{P}_w \to \mathbb{P}$. We often write $\Box_w(w'.P)$ for $\Box_w \lambda w'.P$. As in [6], to guarantee that this interpretation yields a standard type system in the sense of Thm. 12, we require that the modalities satisfy certain properties reminiscent of standard modal axiom schemata [10], which we repeat here for ease of read:

▶ **Definition 11** (Equality modality). *The modality $\Box$ is called an* equality modality *if it satisfies the following properties:*

- $\Box_\mathbf{1}$ *(monotonicity of $\Box$):* $\forall(w:\mathcal{W})(P:\mathcal{P}_w).\forall w' \sqsupseteq w.\ \Box_w P \to \Box_{w'} P.$
- $\Box_\mathbf{2}$ *(K, distribution axiom):* $\forall(w:\mathcal{W})(P,Q:\mathcal{P}_w).\ \Box_w(w'.P\ w' \to Q\ w') \to \Box_w P \to \Box_w Q$
- $\Box_\mathbf{3}$ *(C4, i.e., $\Box$ follows from $\Box\Box$):* $\forall(w:\mathcal{W})(P:\mathcal{P}_w).\ \Box_w(w'.\Box_{w'} P) \to \Box_w P$
- $\Box_\mathbf{4}$*:* $\forall(w:\mathcal{W})(P:\mathcal{P}_w).\forall^{\sqsubseteq}_w(P) \to \Box_w P$
- $\Box_\mathbf{5}$ *(T, reflexivity axiom):* $\forall(w:\mathcal{W})(P:\mathbb{P}).\ \Box_w(w'.P) \to P$

▶ **Theorem 12.** *Given a computation system with choices $\mathcal{C}$ and an equality modality $\Box$, $TT^{\Box}_{\mathcal{C}}$ is a standard type system in the sense that its forcing interpretation induced by $\Box$ satisfy the following properties (where free variables are universally quantified):*

transitivity:	$w \Vdash T_1 \equiv T_2 \to w \Vdash T_2 \equiv T_3 \to w \Vdash T_1 \equiv T_3$	$w \Vdash t_1 \equiv t_2 \in T \to w \Vdash t_2 \equiv t_3 \in T \to w \Vdash t_1 \equiv t_3 \in T$
symmetry:	$w \Vdash T_1 \equiv T_2 \to w \Vdash T_2 \equiv T_1$	$w \Vdash t_1 \equiv t_2 \in T \to w \Vdash t_2 \equiv t_1 \in T$
computation:	$w \Vdash T \equiv T \to T \Downarrow_{!w} T' \to w \Vdash T \equiv T'$	$w \Vdash t \equiv t \in T \to t \Downarrow_{!w} t' \to w \Vdash t \equiv t' \in T$
monotonicity:	$w \Vdash T_1 \equiv T_2 \to w \sqsubseteq w' \to w' \Vdash T_1 \equiv T_2$	$w \Vdash t_1 \equiv t_2 \in T \to w \sqsubseteq w' \to w' \Vdash t_1 \equiv t_2 \in T$
locality:	$\Box_w(w'.w' \Vdash T_1 \equiv T_2) \to w \Vdash T_1 \equiv T_2$	$\Box_w(w'.w' \Vdash t_1 \equiv t_2 \in T) \to w \Vdash t_1 \equiv t_2 \in T$
consistency:	$\neg w \Vdash t \in \mathtt{False}$	

Proof. The proof relies on the properties of the equality modality. For example: $\Box_\mathbf{1}$ is used to prove monotonicity when $w \Vdash T_1 \equiv T_2$ is derived by closing under $\Box_w$; $\Box_\mathbf{2}$ and $\Box_\mathbf{4}$ are used, e.g., to prove the symmetry and transitivity of $w \Vdash t \equiv t' \in \mathsf{Nat}$; $\Box_\mathbf{3}$ is used to prove locality; and $\Box_\mathbf{5}$ is used to prove consistency. See props3.lagda for further details. ◀

As indicated in Thm. 12, and as opposed to the counterpart of the theorem in [6], $w \Vdash T \equiv T$ and $w \Vdash t \equiv t \in T$ are no longer closed under all computations. For example, when $T := \mathsf{Nat}$, if $t \Downarrow_w t'$ and $t \Downarrow_w \underline{n}$, does not necessarily give us that $t' \Downarrow_w \underline{n}$. An example is $t := (\mathtt{choose}(\delta, \underline{1}); \mathtt{if}\ !\delta\ <\ \underline{1}\ \mathtt{then}\ \underline{0}\ \mathtt{else}\ \underline{1})$, which reduces to $t' := (\mathtt{if}\ !\delta\ <\ \underline{1}\ \mathtt{then}\ \underline{0}\ \mathtt{else}\ \underline{1})$ and also to $\underline{1}$ in all worlds, but t' does not reduce to $\underline{1}$ in all worlds, because δ could be initialized differently in different worlds. However, the following holds by transitivity of $\Downarrow_w$: $t' \Downarrow_w t \to w \Vdash t \equiv t \in \mathsf{Nat} \to w \Vdash t \equiv t' \in \mathsf{Nat}$. Similarly, the following also holds by transitivity of $\Downarrow_w$: $w \Vdash T \equiv T \to T' \Downarrow_w T \to w \Vdash T \equiv T'$. Finally, note that, as indicated in Thm. 12, this semantics is closed under β-reduction, as β-reduction does not modify the current world.

4 Proof of Continuity

We can now state the version of Brouwer's continuity principle that we validate in this paper, along with its realizer. For this we first introduce the following notation: $\mathbf{\Pi}_\mathbf{p} a{:}A.B := \mathbf{\Pi} a{:}(A \cap \mathtt{pure}).B$, which quantifies over pure elements of type A.

▶ **Theorem 13** (Continuity Principle). *The following continuity principle, referred to as* CONT_p*, is valid w.r.t. the semantics presented in Sec. 3.2:*

$$\mathbf{\Pi}_\mathbf{p} F{:}\mathcal{B} \to \mathsf{Nat}.\mathbf{\Pi}_\mathbf{p}\alpha{:}\mathcal{B}.\downarrow \mathbf{\Sigma} n{:}\mathsf{Nat}.\mathbf{\Pi}_\mathbf{p}\beta{:}\mathcal{B}.(\alpha{=}\beta \in \mathcal{B}_n) \to (F(\alpha){=}F(\beta) \in \mathsf{Nat}) \tag{1}$$

and is inhabited by

$$\lambda F.\lambda\alpha.\langle \mathtt{mod}(F,\alpha), \lambda\beta.\lambda e.\star\rangle \tag{2}$$

where $\mathtt{mod}(F,\alpha)$ *is the modulus of continuity of the function* $F \in \mathcal{B} \to \mathsf{Nat}$ *at* $\alpha \in \mathcal{B}$ *and is computed by the following expression:*

$$\mathtt{mod}(F,\alpha) \coloneqq \boldsymbol{\nu} x.(\mathtt{choose}(x,\underline{0});F(\mathtt{upd}(x,\alpha));!x + \underline{1})$$
$$\mathtt{upd}(\delta,\alpha) \coloneqq \lambda x.(\mathtt{let}\ y = x\ \mathtt{in}\ ((\mathtt{if}\ !\delta\ <\ y\ \mathtt{then}\ \mathtt{choose}(\delta,y)\ \mathtt{else}\ \star);\alpha(y)))$$

More precisely, the following is true for any world w*:*

$$w \vDash \lambda F.\lambda\alpha.\langle \mathtt{mod}(F,\alpha), \lambda\beta.\lambda e.\star\rangle \in \mathsf{CONT}_{\mathsf{p}}$$

The rest of this section describes the proof of this theorem (see `continuity` in continuity7.lagda for details). First, we intuitively explain how $\mathtt{mod}(F,\alpha)$ computes the modulus of continuity of a function F at a point α. This is done using the following steps:

1. selecting, using $\boldsymbol{\nu}$, a fresh choice name δ (the variable x gets replaced with the freshly generated name δ when computing `mod`), with the appropriate restriction (here a restriction that requires choices to be numbers as mentioned in Sec. 3.1);
2. setting δ to 0 using $\mathtt{choose}(x,\underline{0})$ (where x is δ when this expression computes);
3. applying F to a modified version of α, namely $\mathtt{upd}(\delta,\alpha)$, which computes as α, except that in addition it increases δ's value every time α is applied to a number larger than the last chosen one;
4. returning the last chosen number using $!x$ (again x is δ when this expression computes), increased by one in order to return a number higher than any number F applies α to.

We divide the proof of the validity of the continuity principle, i.e., that it is inhabited by the expression presented in Eq. (2), into the following three components, where $F \in \mathcal{B} \to \mathsf{Nat}$ and $\alpha \in \mathcal{B}$:

- Proving that the modulus is a number, i.e., $\mathtt{mod}(F,\alpha) \in \mathsf{Nat}$;
- Proving that $\mathtt{mod}(F,\alpha)$ returns the highest number that α is applied to in the computation it performs;
- Given $\beta \in \mathcal{B}$, proving that $F(\alpha)$ and $F(\beta)$ return the same number if α and β agree up to $\mathtt{mod}(F,\alpha)$.

4.1 Purity

According to Nat's semantics, to prove that $\mathtt{mod}(F,\alpha) \in \mathsf{Nat}$ w.r.t. a world w, we have to prove it computes to the same number in all extensions of w. However, this will not be the case if F or α have side effects. For example, if F is $\lambda f.f(!\delta_0);0$, for some choice name δ_0, then it could happen that f gets applied to 0 in some world w_1 if $!\delta_0$ returns 0, and to 1 in some world $w_2 \sqsupseteq w_1$ if $!\delta_0$ returns 1. As $\mathtt{mod}(F,\alpha)$ returns the highest number that F applies its argument to, then $\mathtt{mod}(F,\alpha)$ would in this instance return different numbers in different extensions, and would therefore not inhabit Nat.

Therefore, to validate a version of continuity which requires the modulus of continuity to be time-invariant as in Eq. (1), one can require that both F and α are pure (i.e., name-free) terms. Thanks to $\mathbf{\Pi}_{\boldsymbol{p}}$, we get to assume that both F and α are in `pure` and therefore are name-free. Note that it would not be enough to use the following pattern: $\mathbf{\Pi}F{:}\mathcal{B} \to \mathsf{Nat}.(F{=}F \in \mathtt{pure}) \to \ldots$, because then for the continuity principle to even be a type, we would have to prove that F is name-free to prove that $F{=}F \in \mathtt{pure}$ is a type, only knowing that $F \in \mathcal{B} \to \mathsf{Nat}$, which is not true in general.

Let us now mention a potential solution to avoid such a purity requirement, as well as some difficulties it involves, which we leave investigating to future work. One could try to validate instead the following version of the continuity axiom, where $\mathcal{B}_{\downarrow n} = \{x : \mathsf{Nat} \mid x <_\downarrow n\} \to \mathsf{Nat}$, assuming the existence of some type $x <_\downarrow n$ that can relate an $x \in \mathsf{Nat}$ with an $n \in \downarrow\mathsf{Nat}$:

$$\mathbf{\Pi}F{:}\mathcal{B} \to \mathsf{Nat}.\mathbf{\Pi}\alpha{:}\mathcal{B}.\downarrow\mathbf{\Sigma}n{:}\downarrow\mathsf{Nat}.\mathbf{\Pi}\beta{:}\mathcal{B}.(\alpha{=}\beta{\in}\mathcal{B}_{\downarrow n}) \to (F(\alpha){=}F(\beta){\in}\mathsf{Nat})$$

A first difficulty with this is the type $x <_\downarrow n$, which to prove that it holds in some world w would require proving that x is equal to all possible values that n can take in extensions of w. Another related difficulty is that it is at present unclear whether this rule can be validated constructively. More precisely, proving its validity would require:

(1) Proving that $\mathtt{mod}(F,\alpha) \in \downarrow\mathsf{Nat}$, which is now straightforward.

(2) Next, we have to prove that $\mathbf{\Pi}\beta{:}\mathcal{B}.(\alpha{=}\beta{\in}\mathcal{B}_{\downarrow\mathtt{mod}(F,\alpha)}) \to (F(\alpha){=}F(\beta){\in}\mathsf{Nat})$, i.e., given $\beta \in \mathcal{B}$ such that $\alpha{=}\beta{\in}\mathcal{B}_{\downarrow\mathtt{mod}(F,\alpha)}$, we have to prove $F(\alpha){=}F(\beta){\in}\mathsf{Nat}$. The assumption $\alpha{=}\beta{\in}\mathcal{B}_{\downarrow\mathtt{mod}(F,\alpha)}$ tells us that given $k \in \mathsf{Nat}$ such that $k <_\downarrow \mathtt{mod}(F,\alpha)$, $\alpha(k){=}\beta(k){\in}\mathsf{Nat}$. As mentioned above, for $k <_\downarrow \mathtt{mod}(F,\alpha)$ to be true, it must be that k is less than $\mathtt{mod}(F,\alpha)$ in all extensions of the current world. However, without the purity constraint, $\mathtt{mod}(F,\alpha)$ can compute to different numbers in different extensions.

Going back to our goal $F(\alpha){=}F(\beta){\in}\mathsf{Nat}$, given the semantics of Nat presented in Fig. 2, to prove this it is enough to assume that $F(\mathtt{upd}(\delta,\alpha))$ computes to some number $\underline{m}$ in some world w, and to prove that $F(\beta)$ also computes to $\underline{m}$ in w. We can then inspect the computation $F(\mathtt{upd}(\delta,\alpha)) \Downarrow^w_{w_1} \underline{k}$, where δ is the name generated by $\mathtt{mod}(F,\alpha)$, and show that it can be converted into a computation $F(\beta) \Downarrow^w_{w_2} \underline{k}$, by replacing $\alpha(\underline{i})$ with $\beta(\underline{i})$, whenever we encounter such an expression. To do this, we need to know that $\alpha(\underline{i})$ and $\beta(\underline{i})$ compute to the same number using $\alpha{=}\beta{\in}\mathcal{B}_{\downarrow\mathtt{mod}(F,\alpha)}$. However, we only know that $\underline{i}$ is less than $\mathtt{mod}(F,\alpha)$ in w, which is not enough to use this assumption, as $\underline{i}$ might be greater than $\mathtt{mod}(F,\alpha)$ in some other world w'. We can address this issue using classical logic to prove that there exists a $w' \sqsupseteq w$ such that for all $w'' \sqsupseteq w$, the smallest number that α is applied to in the computation of $\mathtt{mod}(F,\alpha)$ w.r.t. w' is less than the number that $\mathtt{mod}(F,\alpha)$ computes to w.r.t. w''. In the argument sketched above we can then use w' instead of w.

4.2 Assumptions

Before we prove that the continuity principle is inhabited, we will summarize here the assumptions we will be making to prove this result, where r is a restriction that requires choices to be numbers (see continuity-conds.lagda for details):

$$(\mathrm{Ass}_1)\quad \forall(w:\mathcal{W})(P:\mathcal{P}_w).\,\Box_w P \to \exists^{\sqsubseteq}_w(P)$$

$$(\mathrm{Ass}_2)\quad \forall(\delta:\mathcal{N})(w:\mathcal{W})(n:\mathbb{N}).\,\mathsf{comp}(\delta,w,r) \to \forall^{\sqsubseteq}_{\mathsf{update}(w,\delta,\underline{n})}(w'.\exists(k:\mathbb{N}).\mathsf{choice?}(w',\delta) = \underline{k})$$

$$(\mathrm{Ass}_3)\quad \forall(\delta:\mathcal{N})(w:\mathcal{W})(k:\mathbb{N}).\mathsf{comp}(\delta,w,r) \to \mathsf{choice?}(\mathsf{update}(w,\delta,\underline{k}),\delta) = k$$

Ass_1 requires the modality $\Box$ to be non-empty in the sense that for $\Box_w P$ to be true, it has to be true for at least one extension of w. This is true about all topological bar spaces (see →∃𝕎 in mod.lagda), and therefore about the Kripke, Beth, and Open modalities which are derived from such spaces [6, Sec.6.2]. Ass_2 requires that the "last" choice of a r-compatible choice name δ is indeed a number. Ass_3 guarantees that retrieving a choice that was just made will return that choice.

The last two assumptions are true about Ref, the formalization of references to numbers presented in Ex. 7 (see for example contInstanceKripkeRef.lagda for the proof that $\mathrm{TT}^{\Box}_{\mathcal{C}}$ instantiated with a Kriple modality and references satisfies these properties). In addition both are true about another kind of stateful computations, namely a variant of the formalization of free choice sequences [15, 31, 27, 26, 17, 32, 21] presented in [6, Ex.5], where new choices are pre-pended as opposed to being appended in [6] (see for example contInstanceKripkeCS.lagda for the proof that $\mathrm{TT}^{\Box}_{\mathcal{C}}$ instantiated with a Kriple modality and choice sequences satisfies these properties).

4.3 The Modulus is a Number

In this section we prove that $\mathtt{mod}(F,\alpha) \in \mathsf{Nat}$. More precisely, we prove the following (see `testM-NAT` in continuity1.lagda for details):

▶ **Theorem 14** (The Modulus is a Number). *If* $\mathtt{namefree}(F)$, $\mathtt{namefree}(\alpha)$, $w \vDash F{\in}\mathsf{Nat}^{\mathcal{B}}$, *and* $w \vDash \alpha{\in}\mathcal{B}$, *for some world* w, *then*

$$\Box_w(w'.\exists(n:\mathbb{N}).\mathtt{mod}(F,\alpha) \Downarrow_{w'} \underline{n}) \tag{3}$$

To prove the above, we will make use of the fact that $w \vDash \mathtt{upd}(\delta,\alpha){\in}\mathcal{B}$ and therefore also $w \vDash F(\mathtt{upd}(\delta,\alpha)){\in}\mathsf{Nat}$, i.e., by the semantics of Nat presented in Fig. 2, we have for some fresh name δ:

$$\Box_w(w'.\exists(n:\mathbb{N}).F(\mathtt{upd}(\delta,\alpha)) \Downarrow_{w'} \underline{n}). \tag{4}$$

But for this we first need to start computing $\mathtt{mod}(F,\alpha)$ to generate a fresh name δ according to the current world. If that current world is some world $w' \sqsupseteq w$ (obtained, for example, using $\Box_4$ from Def. 11 on Eq. (3)), then we need to be able to get that $F(\mathtt{upd}(\delta,\alpha))$ computes to a number w.r.t. w', which Eq. (4) might not provide. This is the reason for assumption Ass_1.

Going back to the proof of Eq. (3), we use $\Box_4$, and have to prove $\exists(n:\mathbb{N}).\mathtt{mod}(F,\alpha) \Downarrow_{w_1} \underline{n}$ for some $w_1 \sqsupseteq w$. We then:

- (A) first have to find a number n such that $\mathtt{mod}(F,\alpha)$ computes to $\underline{n}$ w.r.t. w_1,
- (B) and then that it does so also for all $w_1' \sqsupseteq w_1$.

Let us prove (A) first. We now start computing $\mathtt{mod}(F,\alpha)$ w.r.t. w_1. We generate a fresh name $\delta := \nu\mathcal{C}(w_1)$, and have to prove that $\mathtt{choose}(\delta,\underline{0});F(\mathtt{upd}(\delta,\alpha));!\delta + \underline{1}$ computes to a number w.r.t. $w_2 := \mathtt{start}\nu\mathcal{C}(w_1,r)$ that satisfies $\mathtt{comp}(\delta,w_2,r)$ (by the properties of $\mathtt{start}\nu\mathcal{C}$ presented in Def. 5). We keep computing this expression and have to prove that $F(\mathtt{upd}(\delta,\alpha));!\delta + \underline{1}$ computes to a number w.r.t. $w_3 := \mathtt{update}(w_2,\delta,\underline{0})$.

From Ass_1 and Eq. (4), we obtain $w_5 \sqsupseteq w$ and $n \in \mathbb{N}$ such that $F(\mathtt{upd}(\delta,\alpha)) \Downarrow_{w_5} \underline{n}$, from which we obtain by definition that there exists a w_6 such that $F(\mathtt{upd}(\delta,\alpha)) \Downarrow^{w_5}_{w_6} \underline{n}$. Now, because F and α are name-free, we can derive that there exists a w_4 such that $F(\mathtt{upd}(\delta,\alpha)) \Downarrow^{w_3}_{w_4} \underline{n}$ (see `differNF⇓APPLY-upd` in terms7.lagda). It now remains to prove that $\underline{n};!\delta + \underline{1}$, computes to a number w.r.t. w_4. It is then enough to prove that $!\delta$ computes to a number $\underline{k}$ w.r.t. w_4, in which case $\underline{n};!\delta + \underline{1}$ computes to $\underline{k{+}1}$ w.r.t. w_4. To prove this we make use of Ass_2 which states that r constrains the δ-choices to be numbers. Using this and the facts that $\mathtt{comp}(\delta,w_2,r)$ and $w_2 \sqsubseteq w_4$ (by $\sqsubseteq$'s transitivity since $w_3 \sqsubseteq w_4$ by Lem. 10 and $w_2 \sqsubseteq w_3$ by Def. 6), we deduce that there exists a $k \in \mathbb{N}$ such that $\mathtt{choice?}(w_4,\delta) = \underline{k}$, and therefore $!\delta$ computes to $\underline{k}$ w.r.t. w_4, and $\underline{n};!\delta + \underline{1}$ computes to $\underline{k{+}1}$ w.r.t. w_4, which concludes the proof of (A).

To prove $\exists(n : \mathbb{N}).\mathtt{mod}(F,\alpha) \Downarrow_{w_1} \underline{n}$, we then instantiate the formula with $k{+}1$, and have to prove $\mathtt{mod}(F,\alpha) \Downarrow_{w_1} \underline{k{+}1}$. We already know that $\mathtt{mod}(F,\alpha) \Downarrow^{w_1}_{w_4} \underline{k{+}1}$ (i.e., part (A)), and we now prove our statement labeled (B) above, i.e., that it does so in all extensions of w_1 too.

To prove (B) we assume a $w_1' \sqsupseteq w_1$ and have to prove that $\mathtt{mod}(F,\alpha)$ computes to $\underline{k{+}1}$ w.r.t. w_1'. As before, we start computing $\mathtt{mod}(F,\alpha)$ w.r.t. w_1', and generate a fresh name $\delta' := \nu\mathcal{C}(w_1')$, and have to prove that $F(\mathtt{upd}(\delta',\alpha));!\delta' + \underline{1}$ computes to $\underline{k{+}1}$ w.r.t. $w_3' := \mathtt{update}(w_2',\delta',\underline{0})$, where $w_2' := \mathtt{start}\nu\mathcal{C}(w_1', r)$. As F and α are name-free, $t_1 := F(\mathtt{upd}(\delta,\alpha))$ and $t_2 := F(\mathtt{upd}(\delta',\alpha))$ behave the same except that when t_1 updates δ with a number, t_2 updates δ' with that number.

Using a syntactic simulation method, we will prove that because t_1 and t_2 are "similar" (which is captured by Def. 15 below), $\mathtt{choice?}(w_3,\delta) = \mathtt{choice?}(w_3',\delta')$, and $t_1 \Downarrow^{w_3}_{w_4'} t_1'$, then $t_2 \Downarrow^{w_3'}_{w_4'} t_2'$ such that t_1' and t_2' are also "similar" and $\mathtt{choice?}(w_4,\delta) = \mathtt{choice?}(w_4',\delta')$. Note that $\mathtt{choice?}(w_3,\delta)$ and $\mathtt{choice?}(w_3',\delta')$ return the same choice because $\mathtt{choice?}(w_3,\delta) = \mathtt{choice?}(\mathtt{update}(w_2,\delta,\underline{0}),\delta) = 0$ and $\mathtt{choice?}(w_3',\delta') = \mathtt{choice?}(\mathtt{update}(w_2',\delta',\underline{0}),\delta') = 0$. To derive these equalities, we need assumption Ass_3 that relates $\mathtt{choice?}$ and $\mathtt{update}$.

Let us now define the simulation mentioned above (see `differ` in terms3.lagda for details):

▶ **Definition 15.** *The similarity relation* $t_1 \sim_{\delta_1,\delta_2,\alpha} t_2$ *is true iff*

$$
\begin{aligned}
&(t_1 = \mathtt{upd}(\delta_1,\alpha) \wedge t_2 = \mathtt{upd}(\delta_2,\alpha)) \\
\vee\ &(t_1 = x \wedge t_2 = x) \vee (t_1 = \star \wedge t_2 = \star) \vee (t_1 = \underline{n} \wedge t_2 = \underline{n}) \\
\vee\ &(t_1 = \lambda x.a \wedge t_2 = \lambda x.b \wedge a \sim_{\delta_1,\delta_2,\alpha} b) \\
\vee\ &(t_1 = (a_1\ b_1) \wedge t_2 = (a_2\ b_2) \wedge a_1 \sim_{\delta_1,\delta_2,\alpha} a_2 \wedge b_1 \sim_{\delta_1,\delta_2,\alpha} b_2) \\
\vee\ &\ldots
\end{aligned}
$$

Most cases are omitted in this definition as they similar to the ones presented above. Note however that crucially terms of the form δ *or* $\boldsymbol{\nu}x.t$ *are not related, and that those are the only expressions not related, thereby ruling out names except when occurring inside* upd *through the first clause.*

As discussed above, a key property of this relation is then (see `differ⇓` in terms6.lagda for details):

▶ **Lemma 16.** *If* $t_1 \sim_{\delta_1,\delta_2,\alpha} t_2$, $\mathtt{choice?}(w_1,\delta_1) = \mathtt{choice?}(w_2,\delta_2)$, $t_1 \Downarrow^{w_1}_{w_1'} t_1'$, $\mathtt{namefree}(\alpha)$, $\mathtt{comp}(\delta_1,w_1,r)$, *and* $\mathtt{comp}(\delta_2,w_2,r)$, *then there exist* w_2' *and* t_2' *such that* $t_2 \Downarrow^{w_2}_{w_2'} t_2'$, $t_1' \sim_{\delta_1,\delta_2,\alpha} t_2'$, *and* $\mathtt{choice?}(w_1',\delta_1) = \mathtt{choice?}(w_2',\delta_2)$.

which we prove by induction on the computation $t_1 \Downarrow^{w_1}_{w_1'} t_1'$.

We therefore obtain that there exist t_2' and w_4' such that $F(\mathtt{upd}(\delta',\alpha)) \Downarrow^{w_3'}_{w_4'} t_2'$, $\underline{n} \sim_{\delta,\delta',\alpha} t_2'$ and $\mathtt{choice?}(w_4',\delta') = \mathtt{choice?}(w_4,\delta) = \underline{k}$. Furthermore, by definition of the similarity relation, $t_2' = \underline{n}$. We obtain that $F(\mathtt{upd}(\delta',\alpha));!\delta'{+}\underline{1} \Downarrow^{w_3'}_{w_4'} \underline{n};!\delta'{+}\underline{1}$ and so $F(\mathtt{upd}(\delta',\alpha));!\delta'{+}\underline{1} \Downarrow^{w_3'}_{w_4'} !\delta'{+}\underline{1}$. Because $\mathtt{choice?}(w_4',\delta') = \underline{k}$, we finally obtain $F(\mathtt{upd}(\delta',\alpha));!\delta' + \underline{1} \Downarrow^{w_3'}_{w_4'} \underline{k{+}1}$, which concludes the proof of (B), and therefore that $\mathtt{mod}(F,\alpha) \in \mathsf{Nat}$.

4.4 The Modulus is the Highest Number

We now prove that $\mathtt{mod}(F,\alpha)$ returns the highest number that α is applied to in the computation it performs (see `steps-sat-isHighestℕ` in continuity3.lagda for details):

▶ **Theorem 17** (The Modulus is the Highest Number). *If* $\mathtt{mod}(F,\alpha) \Downarrow^{w}_{w'} \underline{n}$ *such that* $\mathtt{mod}(F,\alpha)$ *generates a fresh name* δ *and* $\mathtt{choice?}(w',\delta) = \underline{i}$*, then for any world* w_0 *occurring along this computation, it must be that* $\mathtt{choice?}(w_0,\delta) = \underline{j}$ *such that* $j \leq i$*.*

As shown above, we know that for any world w_1 there exist $w_2 \in \mathcal{W}$ and $k \in \mathbb{N}$ such that $\mathtt{mod}(F,\alpha) \Downarrow^{w_1}_{w_2} \underline{k{+}1}$. As in Sec. 4.3, we start computing $\mathtt{mod}(F,\alpha)$ w.r.t. the current world w_1, and generate a fresh name $\delta := \nu\mathcal{C}(w_1)$, and deduce that

$$F(\mathtt{upd}(\delta,\alpha));!\delta + \underline{1} \Downarrow^{w_1''}_{w_2} \underline{k{+}1} \tag{5}$$

where $w_1' := \mathtt{start}\nu\mathcal{C}(w_1, r)$ and $w_1'' := \mathtt{update}(w_1', \delta, \underline{0})$. Furthermore, by Ass_2, there must be a $n \in \mathbb{N}$ such that $\mathtt{choice?}(w_2,\delta) = \underline{n}$.

We now want to show that if $n < m$, for some $m \in \mathbb{N}$ (which we will instantiate with $k{+}1$), then it must also be that for any world w along the computation in Eq. (5), if $\mathtt{choice?}(w,\delta) = \underline{i}$ then $i < m$. This is not true about any computation, but it is true about the above because upd only makes a choice if that choice is higher than the "current" one. To capture this, we define the property $\mathtt{Upd}_{\delta,\alpha}(t)$, which captures that the only place where δ occurs in t is wrapped inside $\mathtt{upd}(\delta,\alpha)$. That is, $\mathtt{Upd}_{\delta,\alpha}(t)$ is true iff $t \sim_{\delta,\delta,\alpha} t$. We can then prove the following result by induction on the computation (see continuity3.lagda):

▶ **Lemma 18.** *Let* α *be a closed name-free term, and* t *be a term such that* $\mathtt{Upd}_{\delta,\alpha}(t)$ *and* $t \Downarrow^{w_1}_{w_2} u$*, and let* $\mathtt{choice?}(w_2,\delta) = \underline{n}$*, such that* $n < m$*, then for any world* w *along the computation* $t \Downarrow^{w_1}_{w_2} u$ *if* $\mathtt{choice?}(w,\delta) = \underline{i}$ *then* $i < m$*.*

Applying this result to $F(\mathtt{upd}(\delta,\alpha));!\delta + \underline{1} \Downarrow^{w_1''}_{w_2} \underline{k{+}1}$ and instantiating m with $k{+}1$, we obtain that for any world w along that computation if $\mathtt{choice?}(w,\delta) = \underline{i}$ then $i < k{+}1$.

4.5 The Modulus is the Modulus

We now prove the crux of continuity, namely that F returns the same number on functions that agree up to $\mathtt{mod}(F,\alpha)$ (see `eqfg` in continuity6.lagda for details):

▶ **Theorem 19** (The Modulus is the Modulus). *If* $w \vDash \alpha \equiv \beta \in \mathcal{B}_n$ *then* $w \vDash F(\alpha) \equiv F(\beta) \in \mathsf{Nat}$*.*

First, we prove that $w \vDash F(\alpha) \equiv F(\mathtt{upd}(\delta,\alpha)) \in \mathsf{Nat}$, which follows from the semantics of $\mathbf{\Pi}$ and Nat presented in Fig. 2, and in particular the fact that $w \vDash \alpha \equiv \mathtt{upd}(\delta,\alpha) \in \mathcal{B}$. It is therefore enough to prove that $F(\mathtt{upd}(\delta,\alpha))$ and $F(\beta)$ are equal in Nat. Relating $F(\mathtt{upd}(\delta,\alpha))$ and $F(\beta)$ instead of $F(\alpha)$ and $F(\beta)$ allows getting access to the values that α gets applied to in the computation $F(\alpha)$ as they are recorded using the choice name δ. We can then use these values to prove that $F(\mathtt{upd}(\delta,\alpha))$ and $F(\beta)$ behave similarly up to applications of α in the first computation, which are applications of β in the second, and that these applications reduce to the same numbers because the arguments, recorded using δ, are less than $\mathtt{mod}(F,\alpha)$.

However, even though $\mathtt{upd}(\delta,\alpha)$ and α are equal in $\mathcal{B}$, they behave slightly differently computationally as $\mathtt{upd}(\delta,\alpha)$ turns the call-by-name computations $\alpha(t)$ into call-by-value computations by first reducing t into an expression of the form $\underline{i}$. By typing, we know that $F(\mathtt{upd}(\delta,\alpha))$ and $F(\beta)$ compute to numbers, and to relate the two computations to prove that they compute to the same number, we first apply a similar transformation to $F(\beta)$. Let cbv be defined as follows:

$$\mathtt{cbv}(f) := \lambda x.\mathtt{let}\ y = x\ \mathtt{in}\ f(y).$$

It is straightforward to derive that $w \vDash F(\beta) \equiv F(\mathtt{cbv}(\beta)) \in \mathsf{Nat}$ from the semantics of $\mathbf{\Pi}$ and Nat presented in Fig. 2. It is therefore enough to prove that $F(\mathtt{upd}(\delta, \alpha))$ and $F(\mathtt{cbv}(\beta))$ are equal in Nat.

Because $F(\mathtt{upd}(\delta, \alpha)) \Downarrow^{w}_{w'} \underline{n}$, by Lem. 18 for any world w_0 along this computation if $\mathtt{choice?}(w_0, \delta) = \underline{i}$ then $i < k{+}1$, where $k{+}1$ is the number computed by $\mathtt{mod}(F, \alpha)$.

We now prove that $F(\mathtt{upd}(\delta, \alpha))$ and $F(\mathtt{cbv}(\beta))$ both compute to $\underline{n}$ through another simulation proof that relies on the following relation (see `updRel` in continuity4.lagda for details):

▶ **Definition 20.** *The similarity relation $t_1 \approx_{\delta,\alpha,\beta} t_2$ is true iff*

$$
\begin{aligned}
&(t_1 = \mathtt{upd}(\delta, \alpha) \wedge t_2 = \mathtt{cbv}(\beta)) \\
\vee\ &(t_1 = x \wedge t_2 = x) \vee (t_1 = \star \wedge t_2 = \star) \vee (t_1 = \underline{n} \wedge t_2 = \underline{n}) \\
\vee\ &(t_1 = \lambda x.a \wedge t_2 = \lambda x.b \wedge a \approx_{\delta,\alpha,\beta} b) \\
\vee\ &(t_1 = (a_1\ b_1) \wedge t_2 = (a_2\ b_2) \wedge a_1 \approx_{\delta,\alpha,\beta} a_2 \wedge b_1 \approx_{\delta,\alpha,\beta} b_2) \\
\vee\ &\ldots
\end{aligned}
$$

Most cases are omitted in this definition as they similar to the ones presented above. Note however that crucially terms of the form δ or $\boldsymbol{\nu} x.t$ are not related, and that those are the only expressions not related, thereby ruling out names except when occurring inside `upd` *through the first clause.*

A key property of this relation is as follows, which captures the fact that $t_1 \approx_{\delta,\alpha,\beta} t_2$ is preserved by computations, and which we prove by induction on the computation (see `steps-updRel` in continuity5.lagda for details):

▶ **Lemma 21.** *If $t_1 \approx_{\delta,\alpha,\beta} t_2$, α and β agree up to k, $t_1 \Downarrow^{w}_{w'} t_1'$ and for any world w_0 along this computation if $\mathtt{choice?}(w_0, \delta) = \underline{i}$ then $i < k{+}1$, then $t_2 \Downarrow^{w}_{w} t_2'$ such that $t_1' \approx_{\delta,\alpha,\beta} t_2'$.*

Therefore, because $F(\mathtt{upd}(\delta, \alpha)) \approx_{\delta,\alpha,\beta} F(\mathtt{cbv}(\beta))$ (as F is name-free) and $F(\mathtt{upd}(\delta, \alpha))$ computes to $\underline{n}$, it must be that $F(\mathtt{cbv}(\beta))$ also computes to $\underline{n}$, which concludes our proof of Thm. 13.

5 Conclusion and Related Works

We have shown in this paper how to validate a continuity principle for a subset of the $\mathrm{TT}^{\square}_{\mathcal{C}}$ family of type theories, such that the modulus of continuity of functions is internalized, i.e., computed using an expression of the underlying computation system. In particular, we have used stateful computations, and have discussed some of the challenges arising from such impure computations. As mentioned in the introduction, and as recalled below, this is not the first proof of continuity, however to the best of our knowledge, this is the first proof of an "internal" validity proof of continuity that relies on stateful computations. Furthermore, the proof presented above relies on an "internal" notion of probing through the use of stateful computations internal to the computation language of the type theory, while approaches such as [8, 9, 34] rely on a meta-theoretic (or "external") notion of probing.

Troelstra proved in [29, p.158] that every closed term $t \in \mathbb{N}^{\mathcal{B}}$ of N-HA$^{\omega}$ has a provable modulus of continuity in N-HA$^{\omega}$ – see also [4] for similar proofs of the consistency of continuity with various constructive theories.

Coquand and Jaber [8, 9] proved the *uniform* continuity of a Martin-Löf-like intensional type theory using forcing. Their method consists in adding a generic element `f` as a constant to the language that stands for a Cohen real of type $2^{\mathbb{N}}$, and defining the forcing conditions

as approximations of `f`. They then define a suitable *computability* predicate that expresses when a term is a computable term of some type up to approximations given by the forcing conditions. The key steps are to (1) first prove that `f` is computable and then (2) prove that well-typed terms are computable, from which they derive uniform continuity: the uniform modulus of continuity is given by the approximations.

Similarly, Escardó and Xu [34] proved that the definable functionals of Gödel's system T [14] are uniformly continuous on the Cantor space $\mathcal{C}$ (without assuming classical logic or the Fan Theorem). For that, they developed the C-Space category, which internalizes continuity, and has a *Fan functional* which computes the modulus of uniform continuity of functions in $\mathcal{C} \to \mathbb{N}$. Relating C-Space and the standard set-theoretical model of system T, they show that all System T functions on the Cantor space are uniformly continuous. Furthermore, using this model, they show how to extract computational content from proofs in HA^{ω} extended with a uniform continuity axiom, which is realized by the Fan functional.

In [13], Escardó proves that all System T functions are continuous on the Baire space and uniformly continuous on the Cantor space without using forcing. Instead, he provides an alternative interpretation of system T, where a number is interpreted by a *dialogue tree*, which captures the computation of a function w.r.t. an oracle of type $\mathcal{B}$. Escardó first proves that such computations are continuous, and then by defining a suitable relation between the standard interpretation and the alternative one, that relates the interpretations of all system T terms, derives that for all system T functions on the Baire space are continuous.

In [24, 25], the authors proved that Brouwer's continuity principle is consistent with Nuprl [7, 2] by realizing the modulus of continuity of functions on the Baire space also using Longley's method [20], but using exceptions instead of references. The realizer there is more complicated than the one presented in this paper as it involves an effectful computation that repeatedly checks whether a given number is at least as high as the modulus of continuity, and increasing that number until the modulus of continuity is reached. We do not require such a loop, and can directly extract the modulus of continuity of a function.

In [3] the authors prove that all BTT [22] functions are continuous by generalizing the method used in [13]. Their model is built in three steps as follows: an axiom model/translation adds an oracle to the theory at hand; a branching model/translation interprets types as intensional D-algebras, i.e., as types equipped with pythias; and an algebraic parametricity model/translation that relates the two previous translations by relating the calls to the pythia to the oracle. Their models allows deriving that all functions are continuous, but does not allow "internalizing" the continuity principle, which is the goal of this paper.

References

1 Agda wiki. URL: `http://wiki.portal.chalmers.se/agda/pmwiki.php`.

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. *J. Applied Logic*, 4(4):428–469, 2006. `doi:10.1016/j.jal.2005.10.005`.

3 Martin Baillon, Assia Mahboubi, and Pierre-Marie Pédrot. Gardening with the pythia A model of continuity in a dependent setting. In Florin Manea and Alex Simpson, editors, *CSL*, volume 216 of *LIPIcs*, pages 5:1–5:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPIcs.CSL.2022.5`.

4 Michael J. Beeson. *Foundations of Constructive Mathematics*. Springer, 1985.

5 Douglas Bridges and Fred Richman. *Varieties of Constructive Mathematics*. London Mathematical Society Lecture Notes Series. Cambridge University Press, 1987. `doi:10.1017/CBO9780511565663`.

6 Liron Cohen and Vincent Rahli. Constructing unprejudiced extensional type theories with choices via modalities. In Amy P. Felty, editor, *FSCD*, volume 228 of *LIPIcs*, pages 10:1–10:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.FSCD.2022.10.

7 Robert L. Constable, Stuart F. Allen, Mark Bromley, Rance Cleaveland, J. F. Cremer, Robert W. Harper, Douglas J. Howe, Todd B. Knoblock, Nax P. Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing mathematics with the Nuprl proof development system*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.

8 Thierry Coquand and Guilhem Jaber. A note on forcing and type theory. *Fundam. Inform.*, 100(1-4):43–52, 2010. doi:10.3233/FI-2010-262.

9 Thierry Coquand and Guilhem Jaber. A computational interpretation of forcing in type theory. In *Epistemology versus Ontology*, volume 27 of *Logic, Epistemology, and the Unity of Science*, pages 203–213. Springer, 2012. doi:10.1007/978-94-007-4435-6_10.

10 M. J. Cresswell and G. E. Hughes. *A New Introduction to Modal Logic*. Routledge, 1996.

11 Michael A. E. Dummett. *Elements of Intuitionism*. Clarendon Press, second edition, 2000.

12 Martín H. Escardó and Chuangjie Xu. The inconsistency of a Brouwerian continuity principle with the Curry-Howard interpretation. In *TLCA 2015*, volume 38 of *LIPIcs*, pages 153–164. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.TLCA.2015.153.

13 Martín Hötzel Escardó. Continuity of Gödel's system T definable functionals via effectful forcing. *Electr. Notes Theor. Comput. Sci.*, 298:119–141, 2013. doi:10.1016/j.entcs.2013.09.010.

14 Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, 1989.

15 Stephen C. Kleene and Richard E. Vesley. *The Foundations of Intuitionistic Mathematics, especially in relation to recursive functions*. North-Holland Publishing Company, 1965.

16 Georg Kreisel. On weak completeness of intuitionistic predicate logic. *J. Symb. Log.*, 27(2):139–158, 1962. doi:10.2307/2964110.

17 Georg Kreisel and Anne S. Troelstra. Formal systems for some branches of intuitionistic analysis. *Annals of Mathematical Logic*, 1(3):229–387, 1970. doi:10.1016/0003-4843(70)90001-X.

18 Saul A. Kripke. Semantical analysis of modal logic i. normal propositional calculi. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, 9(5-6):67–96, 1963. doi:10.1002/malq.19630090502.

19 Saul A. Kripke. Semantical analysis of intuitionistic logic i. In J.N. Crossley and M.A.E. Dummett, editors, *Formal Systems and Recursive Functions*, volume 40 of *Studies in Logic and the Foundations of Mathematics*, pages 92–130. Elsevier, 1965. doi:10.1016/S0049-237X(08)71685-9.

20 John Longley. When is a functional program not a functional program? In *ICFP'99*, pages 1–7. ACM, 1999. doi:10.1145/317636.317775.

21 Joan R. Moschovakis. An intuitionistic theory of lawlike, choice and lawless sequences. In *Logic Colloquium'90: ASL Summer Meeting in Helsinki*, pages 191–209. Association for Symbolic Logic, 1993.

22 Pierre-Marie Pédrot and Nicolas Tabareau. An effectful way to eliminate addiction to dependence. In *LICS 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005113.

23 Andrew M Pitts. Nominal sets: Names and symmetry in computer science, volume 57 of cambridge tracts in theoretical computer science, 2013.

24 Vincent Rahli and Mark Bickford. A nominal exploration of intuitionism. In Jeremy Avigad and Adam Chlipala, editors, *CPP 2016*, pages 130–141. ACM, 2016. Extended version: http://www.nuprl.org/html/Nuprl2Coq/continuity-long.pdf. doi:10.1145/2854065.2854077.

25 Vincent Rahli and Mark Bickford. Validating brouwer's continuity principle for numbers using named exceptions. *Mathematical Structures in Computer Science*, pages 1–49, 2017. doi:10.1017/S0960129517000172.

26 Anne S. Troelstra. *Choice sequences: a chapter of intuitionistic mathematics*. Clarendon Press Oxford, 1977.

27 Anne S. Troelstra. Choice sequences and informal rigour. *Synthese*, 62(2):217–227, 1985.

28 Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics An Introduction*, volume 121 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1988.

29 A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. New York, Springer, 1973.

30 A.S. Troelstra. A note on non-extensional operations in connection with continuity and recursiveness. *Indagationes Mathematicae*, 39(5):455–462, 1977. doi:10.1016/1385-7258(77)90060-9.

31 Mark van Atten and Dirk van Dalen. Arguments for the continuity principle. *Bulletin of Symbolic Logic*, 8(3):329–347, 2002. URL: http://www.math.ucla.edu/~asl/bsl/0803/0803-001.ps, doi:10.2178/bsl/1182353892.

32 Wim Veldman. Understanding and using Brouwer's continuity principle. In *Reuniting the Antipodes — Constructive and Nonstandard Views of the Continuum*, volume 306 of *Synthese Library*, pages 285–302. Springer Netherlands, 2001. doi:10.1007/978-94-015-9757-9_24.

33 Chuangjie Xu. *A continuous computational interpretation of type theories*. PhD thesis, University of Birmingham, UK, 2015. URL: http://etheses.bham.ac.uk/5967/.

34 Chuangjie Xu and Martín Hötzel Escardó. A constructive model of uniform continuity. In *TLCA 2013*, volume 7941 of *LNCS*, pages 236–249. Springer, 2013. doi:10.1007/978-3-642-38946-7_18.

Non-Uniform Complexity via Non-Wellfounded Proofs

Gianluca Curzi
University of Birmingham, UK

Anupam Das
University of Birmingham, UK

Abstract

Cyclic and non-wellfounded proofs are now increasingly employed to establish metalogical results in a variety of settings, in particular for type systems with forms of (co)induction. Under the Curry-Howard correspondence, a cyclic proof can be seen as a typing derivation "with loops", closer to low-level machine models, and so comprise a highly expressive computational model that nonetheless enjoys excellent metalogical properties.

In recent work, we showed how the cyclic proof setting can be further employed to model computational complexity, yielding characterisations of the polynomial time and elementary computable functions. These characterisations are "implicit", inspired by Bellantoni and Cook's famous algebra of safe recursion, but exhibit greater expressivity thanks to the looping capacity of cyclic proofs.

In this work we investigate the capacity for *non-wellfounded* proofs, where finite presentability is relaxed, to model non-uniformity in complexity theory. In particular, we present a characterisation of the class $\mathbf{FP}/poly$ of functions computed by polynomial-size circuits. While relating non-wellfoundedness to non-uniformity is a natural idea, the precise amount of irregularity, informally speaking, required to capture $\mathbf{FP}/poly$ is given by proof-level conditions novel to cyclic proof theory. Along the way, we formalise some (presumably) folklore techniques for characterising non-uniform classes in relativised function algebras with appropriate oracles.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Proof theory

Keywords and phrases Cyclic proofs, non-wellfounded proof-theory, non-uniform complexity, polynomial time, safe recursion, implicit complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.16

Related Version *Full Version*: `https://arxiv.org/abs/2211.16104`

Funding Both authors are supported by a UKRI Future Leaders Fellowship, *Structure vs. Invariants in Proofs*, project reference MR/S035540/1.

Acknowledgements We thank the anonymous reviewers for their helpful comments and suggestions.

1 Introduction

Non-wellfounded proof theory is the study of possibly infinite (but finitely branching) proofs, where appropriate global correctness criteria guarantee logical consistency. This area originates (in its modern guise) in the context of the modal μ-calculus [31, 16], serving as an alternative framework to manipulate least and greatest fixed points, and hence to model inductive and coinductive reasoning. Since then, non-wellfounded proofs have been widely investigated in many respects, such as predicate logic [8, 6], algebras [14, 15], arithmetic [32, 5, 12], proofs-as-programs interpretations [2, 17, 11, 24, 13], and continuous cut-elimination [30, 18]. Special attention in these works is drawn to *cyclic* (or *regular*) proofs, i.e. non-wellfounded proofs with only finitely many distinct subproofs, comprising a natural notion of finite presentability in terms of (possibly cyclic) directed graphs.

The *Curry-Howard* reading of non-wellfounded proofs has revealed a deep connection between proof-theoretic properties and computational behaviours [11, 24, 13]. On the one hand, the typical correctness conditions ensuring consistency, called *progressing* (or *validity*) criteria, correspond to totality: functions computed by progressing proofs are always well-defined on all arguments. On the other hand, regularity has a natural counterpart in the notion of *uniformity*: circular proofs can be properly regarded as programs, i.e. as finite sets of machine instructions, thus having a "computable" behaviour.

In a recent work [10], the authors extended these connections between non-wellfounded proof theory and computation to the realm of *computational complexity*. We introduced the proof systems CB and CNB capturing, respectively, the class of functions computable in polynomial time (**FP**) and the elementary functions (**FELEMENTARY**). These proof systems are defined by identifying global conditions on circular progressing proofs motivated by ideas from *Implicit Computational Complexity* (ICC).

ICC, broadly construed, is the study of machine-free (and often bound-free) characterisations of complexity classes. One of the seminal works in the area is Bellantoni and Cook's function algebra B for **FP** based on *safe recursion* [4]. The prevailing idea behind safe recursion (and its predecessor, *ramified recursion* [26]) is to partition function arguments into "safe" and "normal", namely writing $f(x_1, \dots, x_m; y_1, \dots, y_n)$ when f takes m normal inputs $\vec{x}$ and n safe inputs $\vec{y}$. In functions of B, the recursive parameters are always normal arguments, while recursive calls can only appear in safe position; hence, no recursive call can be used as recursive parameters of other previously defined functions. Our system CB morally represents a cyclic proof theoretic formulation of B.

To establish the characterisation result for CB we developed a novel function algebra for **FP**, called $\mathsf{B}^{\subset}$. Roughly, the latter extends B with a more expressive recursion mechanism on a special well-founded preorder, "$\subset$", based on permutation of prefixes of normal arguments, and whose definition requires relativisation of the algebra to admit *oracles*. The characterisation theorem is then obtained by a "sandwich" technique, where the function algebras B and $\mathsf{B}^{\subset}$ serve, respectively, as lower and upper bounds for CB.

In this paper we investigate the computational interpretation of more general *non-wellfounded* proofs, where finite presentability is relaxed in order to model *non-uniform complexity*. In particular we consider the class $\mathbf{FP}/\mathit{poly}$ of functions computable in polynomial time by Turing machines with access to *polynomial advice*. Equivalently, $\mathbf{FP}/\mathit{poly}$ is the class of functions computed by families of polynomial-size circuits. Note, in particular, that $\mathbf{FP}/\mathit{poly}$ includes *undecidable problems*, and so cannot be characterised by purely cyclic proof systems or usual function algebras, which typically have only computable functions.

We define the system nuB ("non-uniform B"), allowing a form of non-wellfoundedness somewhere between arbitrary non-wellfounded proofs and full regularity, and show that nuB duly characterises $\mathbf{FP}/\mathit{poly}$. The characterisation theorem for nuB relies on an adaption of the aforementioned sandwich technique for CB to the current setting. This requires a relativisation of both B and $\mathsf{B}^{\subset}$ to a set of oracles, which we call $\mathbb{R}$, deciding properties of string length. As a byproduct of our proof method we also obtain new relativised function algebras for $\mathbf{FP}/\mathit{poly}$ based on safe recursion, $\mathsf{B}(\mathbb{R})$ and $\mathsf{B}^{\subset}(\mathbb{R})$; these are folklore-style results that, as far as we know, have not yet appeared in the literature.

The overall structure of our result relies on a "grand tour" of inclusions, summarised as:

$$\mathbf{FP}/\mathit{poly} \overset{\text{P.27}}{\subseteq} \mathsf{B}(\mathbb{R}_{1;0}) \overset{\text{P.32}}{\subseteq} \mathsf{CB}(\mathbb{R}_{1;0}) \overset{\text{P.33}}{\subseteq} \mathsf{nuB} \overset{\text{T.36}}{\subseteq} \mathsf{CB}(\mathbb{R}_{1;1}) \overset{\text{L.43}}{\subseteq} \mathsf{B}^{\subset}(\mathbb{R}_{1;1}) \overset{\text{P.42}}{\subseteq} \mathbf{FP}(\mathbb{R}) \overset{\text{P.26}}{\subseteq} \mathbf{FP}/\mathit{poly}$$

While this may seem like a long route to take, the structure of our argument is designed so that each of the above inclusions are relatively simple to establish and, as we said, yields several intermediate characterisions of $\mathbf{FP}/\mathit{poly}$ of self-contained interest.

Related work. Characterisations of non-uniform complexity classes in the style of ICC have been considered in the context of the λ-calculus [27] and variants of linear logic [29]. The former captures the class $\mathbf{P}/poly$, i.e., the languages decided by families of polynomial circuits, while the latter also captures $\mathbf{L}/poly$, i.e., the languages decided by families of polynomial size branching programs (i.e. decision trees with sharing). However this is the first work (as far as we know) that attempts to relate non-wellfoundedness in proof theory to non-uniformity in complexity theory.

The relativised proof systems and function algebras presented in this paper only query "bits of real numbers". Proof systems based on linear logic and implementing polytime computation over actual binary streams have been considered, e.g., in [20], which provide an ICC-like characterisation of Ko's class of polynomial time computable functions over real numbers [23].

Outline of the paper. This paper is structured as follows. In Section 2 we recall some preliminaries on non-uniform and implicit complexity, in particular a proof theoretic formulation of the algebra B. In Section 3 we recall the circular system CB from [10], and introduce our new system nuB. In Section 4 we take an interlude to present some *relativised* characterisations of $\mathbf{FP}/poly$, both in the machine setting and the implicit setting, that will later serve use in our grand tour of inclusions. In Section 5 we employ those characterisations to establish the lower bound for nuB, and in Section 6 we recast nuB as a sort of relativised circular system. Finally in Section 7 we adapt results from [10] translating circular proofs to an appropriate function algebra to the relativised setting, thereby achieving the upper bound for nuB.

2 Preliminaries on computational complexity and safe recursion

Throughout this work we only consider (partial) functions on *natural numbers*. We write $|x|$ for the length of the binary representation of a number x, and for lists of arguments $\vec{x} = x_1, \dots, x_n$ we write $|\vec{x}|$ for the list $|x_1|, \dots, |x_n|$.

2.1 Non-uniform complexity classes

$\mathbf{FP}$ is the class of (total) functions computable in polynomial time on a Turing machine. The "non-uniform" class $\mathbf{FP}/poly$ is an extension of $\mathbf{FP}$ that intuitively has access to a polynomial amount of "advice", determined only by the *length* of the input. Formally:

▶ **Definition 1** (Non-uniform polynomial time)**.** $\mathbf{FP}/poly$ is the class of functions $f(\vec{x})$ for which there are strings $\alpha_{\vec{n}} \in \{0,1\}^*$, of size polynomial in $\vec{n}$, and some $f'(x, \vec{x}) \in \mathbf{FP}$ with:

- $|\alpha_{\vec{n}}|$ is polynomial in $\vec{n}$.
- $f(\vec{x}) = f'(\alpha_{|\vec{x}|}, \vec{x})$.

The strings $\{\alpha_{\vec{n}}\}_{\vec{n}}$ represent the *polynomial advice* given to a polynomial-time computation, here $f'(x, \vec{x})$. Note that $f(\vec{x})$ only "receives advice' depending on the lengths of its inputs, $\vec{x}$.

Note, in particular, that $\mathbf{FP}/poly$ admits undecidable problems. E.g. the function $f(x) = 1$ just if $|x|$ is the code of a halting Turing machine (and 0 otherwise) is in $\mathbf{FP}/poly$. Indeed, the point of the class $\mathbf{FP}/poly$ is to rather characterise a more non-uniform notion of computation. In particular, the following is well-known (see, e.g., [1, Theorem 6.11]):

▶ **Proposition 2.** *$f(\vec{x}) \in \mathbf{FP}/poly$ iff there are polynomial-size circuits computing $f(\vec{x})$.*

2.2 The Bellantoni-Cook algebra

A *two-sorted* function is a function $f(\vec{x};\vec{y})$ whose arguments have been delimited into "normal" ones ($\vec{x}$, left of ";"), and "safe" ones ($\vec{y}$, right of ";").

The two-sorted algebra B was introduced in [4] and is defined as follows:

▶ **Definition 3** (Bellantoni-Cook)**.** B is the smallest class of two-sorted functions containing,

- $0(;) := 0$
- $\mathsf{s}_0(;x) := 2x$
- $\mathsf{s}_1(;x) := 2x+1$
- $\pi^{m;n}_{j;}(x_0,\ldots,x_{m-1};y_0,\ldots,y_{n-1}) := x_j$, whenever $j < m$.
- $\pi^{m;n}_{;j}(x_0,\ldots,x_{m-1};y_0,\ldots,y_{n-1}) := y_j$, whenever $j < n$.
- $\mathsf{p}(;x) = \lfloor \frac{x}{2} \rfloor$
- $\mathsf{cond}(;w,x,y,z) := \begin{cases} x & w = 0 \\ y & w = 0 \mod 2, w \neq 0 \\ z & w = 1 \mod 2 \end{cases}$

and closed under:

- (Safe composition)
  - if $g(\vec{x};) \in \mathsf{B}$ and $h(\vec{x},x;\vec{y}) \in \mathsf{B}$ then also $f(\vec{x};\vec{y}) \in \mathsf{B}$ where $f(\vec{x};\vec{y}) := h(\vec{x},g(\vec{x};);\vec{y})$.
  - if $g(\vec{x};\vec{y}) \in \mathsf{B}$ and $h(\vec{x};\vec{y},y) \in \mathsf{B}$ then also $f(\vec{x};\vec{y}) \in \mathsf{B}$ where $f(\vec{x};\vec{y}) := h(\vec{x};\vec{y},g(\vec{x};\vec{y}))$
- (Safe recursion on notation) if $g(\vec{x};\vec{y}) \in \mathsf{B}$ and $h_0(x,\vec{x};\vec{y},y), h_1(x,\vec{x};\vec{y},y) \in \mathsf{B}$ then also $f(x,\vec{x};\vec{y}) \in \mathsf{B}$ where:

$$\begin{aligned} f(0,\vec{x};\vec{y}) &:= g(\vec{x};\vec{y}) \\ f(\mathsf{s}_0 x,\vec{x};\vec{y}) &:= h_0(x,\vec{x};\vec{y},f(x,\vec{x};\vec{y})) \quad x \neq 0 \\ f(\mathsf{s}_1 x,\vec{x};\vec{y}) &:= h_1(x,\vec{x};\vec{y},f(x,\vec{x};\vec{y})) \end{aligned}$$

Safe composition ensures that safe arguments may never appear in a normal position. Note that, in the recursion scheme, the recursion parameter is always a normal argument, whereas recursive calls must appear in safe position. Along with the constraints on safe composition, this ensures that the position of a recursive call is never the recursion parameter of another recursion. This seemingly modest constraint duly restricts computation to polynomial time, yielding Bellantoni and Cook's main result:

▶ **Theorem 4** ([4])**.** *$f(\vec{x}) \in \mathbf{FP}$ if and only if $f(\vec{x};) \in \mathsf{B}$.*

2.3 A proof-theoretic presentation of Bellantoni-Cook

We shall work with a formulation of B as a $S4$-style type system in sequent-calculus style, where modalities are used to distinguish the two sorts (similarly to [21]).

We consider *types* (or *formulas*) N ("safe") and $\Box N$ ("normal") which intuitively vary over the natural numbers. We write A, B, etc. to vary over types. A *sequent* is an expression $\Gamma \Rightarrow A$, where Γ is a list of types (called the *context* or *antecedent*) and A is a type (called the *succedent*). For a list of types $\Gamma = \overbrace{N,\ldots,N}^{k}$, we write $\Box\Gamma$ for $\overbrace{\Box N,\ldots,\Box N}^{k}$.

▶ **Definition 5.** A B-*derivation* is a (finite) derivation built from the rules in Figure 1.

$$\mathsf{id}\ \frac{}{N \Rightarrow N} \qquad \mathsf{cut}_N\ \frac{\Gamma \Rightarrow N \quad \Gamma, N \Rightarrow B}{\Gamma \Rightarrow B} \qquad \mathsf{cut}_\Box\ \frac{\Gamma \Rightarrow \Box N \quad \Box N, \Gamma \Rightarrow B}{\Gamma \Rightarrow B}$$

$$\mathsf{w}_N\ \frac{\Gamma \Rightarrow B}{\Gamma, N \Rightarrow B} \qquad \mathsf{w}_\Box\ \frac{\Gamma \Rightarrow B}{\Box N, \Gamma \Rightarrow B} \qquad \mathsf{e}\ \frac{\Gamma, A, B, \Gamma' \Rightarrow C}{\Gamma, B, A, \Gamma' \Rightarrow C} \qquad \Box_l\ \frac{\Gamma, N \Rightarrow A}{\Box N, \Gamma \Rightarrow A} \qquad \Box_r\ \frac{\Box\Gamma \Rightarrow N}{\Box\Gamma \Rightarrow \Box N}$$

$$0\ \frac{}{\Rightarrow N} \qquad 1\ \frac{}{\Rightarrow N} \qquad \mathsf{s}_0\ \frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A} \qquad \mathsf{s}_1\ \frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A} \qquad \mathsf{srec}\ \frac{\Gamma \Rightarrow N \quad \Box N, \Gamma, N \Rightarrow N \quad \Box N, \Gamma, N \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

$$\mathsf{cond}_N\ \frac{\Gamma \Rightarrow N \quad \Gamma, N \Rightarrow N \quad \Gamma, N \Rightarrow N}{\Gamma, N \Rightarrow N} \qquad \mathsf{cond}_\Box\ \frac{\Gamma \Rightarrow N \quad \Box N, \Gamma \Rightarrow N \quad \Box N, \Gamma \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

$$|\mathsf{cond}|_N\ \frac{\Gamma \Rightarrow N \quad \Gamma, N \Rightarrow N}{\Gamma, N \Rightarrow N} \qquad |\mathsf{cond}|_\Box\ \frac{\Gamma \Rightarrow N \quad \Box N, \Gamma \Rightarrow N}{\Box N, \Gamma \Rightarrow N}$$

Figure 1 B as a sequent-style type system.

$$\begin{aligned}
f_{\mathsf{id}}(;y) &:= y \\
f_{\mathsf{cut}_N(\mathcal{D}_0,\mathcal{D}_1)}(\vec{x};\vec{y}) &:= f_{\mathcal{D}_1}(\vec{x};\vec{y}, f_{\mathcal{D}_0}(\vec{x};\vec{y})) \\
f_{\mathsf{cut}_\Box(\mathcal{D}_0,\mathcal{D}_1)}(\vec{x};\vec{y}) &:= f_{\mathcal{D}_1}(f_{\mathcal{D}_0}(\vec{x};\vec{y}),\vec{x};\vec{y}) \\
f_{\mathsf{w}_N(\mathcal{D}_0)}(\vec{x};\vec{y},y) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{\mathsf{w}_\Box(\mathcal{D}_0)}(x,\vec{x};\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{\mathsf{e}_N(\mathcal{D}_0)}(\vec{x};\vec{y},y,y',\vec{y}') &:= f_{\mathcal{D}_0}(\vec{x};\vec{y},y',y,\vec{y}') \\
f_{\mathsf{e}_\Box(\mathcal{D}_0)}(\vec{x},x,x',\vec{x}';\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x},x',x,\vec{x}';\vec{y}) \\
f_{\Box_l(\mathcal{D}_0)}(x,\vec{x};\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y},x) \\
f_{\Box_r(\mathcal{D}_0)}(\vec{x};) &:= f_{\mathcal{D}_0}(\vec{x};) \\
f_i(;) &:= i \\
f_{\mathsf{s}_i(\mathcal{D}_0)}(\vec{x};\vec{y}) &:= \mathsf{s}_i(;f_{\mathcal{D}_0}(\vec{x};\vec{y}))
\end{aligned}$$

$$\begin{aligned}
f_{\mathsf{srec}(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(0,\vec{x};\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{\mathsf{srec}(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(\mathsf{s}_i x,\vec{x};\vec{y}) &:= f_{\mathcal{D}_{i+1}}(x,\vec{x};\vec{y}, \\
&\qquad f_{\mathsf{srec}(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(x,\vec{x};\vec{y})) \\
f_{\mathsf{cond}_N(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(\vec{x};\vec{y},0) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{\mathsf{cond}_N(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(\vec{x};\vec{y},\mathsf{s}_i y) &:= f_{\mathcal{D}_{i+1}}(\vec{x};\vec{y},y) \\
f_{\mathsf{cond}_\Box(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(0,\vec{x};\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{\mathsf{cond}_\Box(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_2)}(\mathsf{s}_i x,\vec{x};\vec{y}) &:= f_{\mathcal{D}_{i+1}}(x,\vec{x};\vec{y}) \\
f_{|\mathsf{cond}|_N(\mathcal{D}_0,\mathcal{D}_1)}(\vec{x};\vec{y},0) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{|\mathsf{cond}|_N(\mathcal{D}_0,\mathcal{D}_1)}(\vec{x};\vec{y},\mathsf{s}_i y) &:= f_{\mathcal{D}_1}(\vec{x};\vec{y},y) \\
f_{|\mathsf{cond}|_\Box(\mathcal{D}_0,\mathcal{D}_1)}(0,\vec{x};\vec{y}) &:= f_{\mathcal{D}_0}(\vec{x};\vec{y}) \\
f_{|\mathsf{cond}|_\Box(\mathcal{D}_0,\mathcal{D}_1)}(\mathsf{s}_i x,\vec{x};\vec{y}) &:= f_{\mathcal{D}_1}(x,\vec{x};\vec{y})
\end{aligned}$$

Figure 2 Semantics of system B, where $i \in \{0,1\}$ and $\mathsf{s}_i x \neq 0$ and $\mathsf{s}_i y \neq 0$.

The colouring of type occurrences in Figure 1 may be ignored for now, they will become relevant in the next section. We may write $\mathcal{D} = \mathsf{r}(\mathcal{D}_1, \dots, \mathcal{D}_n)$ (for $n \leq 3$) if r is the bottom-most inference step of a derivation $\mathcal{D}$ whose immediate subderivations are, respectively, $\mathcal{D}_1, \dots, \mathcal{D}_n$. As done in [10], we shall assume w.l.o.g. that sequents have shape $\Box N, \dots, \Box N, N, \dots, N \Rightarrow A$, i.e. in the left-hand side all $\Box N$ occurrences are placed before all N occurrences.

We construe the system of B-derivations as a class of two-sorted functions by identifying each rule instance as an operation on two-sorted functions as follows:

▶ **Definition 6** (Semantics of B). Given a B-derivation $\mathcal{D}$ of $\overbrace{\Box N, \dots, \Box N}^{m}, \overbrace{N, \dots, N}^{n} \Rightarrow A$ we define a two-sorted function $f_{\mathcal{D}}(x_1, \dots, x_m; y_1, \dots, y_n)$ in Figure 2 by induction on the structure of $\mathcal{D}$ (all rules as typeset in Figure 1).

This formal semantics exposes how B-derivations and functions in the algebra B relate. The rule srec in Figure 1 corresponds to safe recursion, and safe composition along safe parameters is expressed by cut_N. Note, however, that the interpretation of $\mathsf{cut}_\Box$ in Figure 2

apparently does not satifsfy the required constraint on safe composition of a function g along a normal parameter of a function h, which forbids the presence of safe parameters in g. However, this admission turns out to be harmless, and we are able to obtain the following result that justifies the overloading of the notation "B":

▶ **Proposition 7** ([10]). *$f(\vec{x};\vec{y}) \in \mathsf{B}$ iff there is a B-derivation $\mathcal{D}$ for which $f_{\mathcal{D}}(\vec{x};\vec{y}) = f(\vec{x};\vec{y})$.*

▶ Remark 8 (Bootstrapping). Note that the rules 1, $|\mathsf{cond}|_N$ and $|\mathsf{cond}|_\square$ are semantically redundant, being derivable from the others by: $f_1 = f_{\mathsf{s}_1(0)}$, $f_{|\mathsf{cond}|_N(\mathcal{D}_0,\mathcal{D}_1)} = f_{\mathsf{cond}_N(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_1)}$, and $f_{|\mathsf{cond}|_\square(\mathcal{D}_0,\mathcal{D}_1)} = f_{\mathsf{cond}_\square(\mathcal{D}_0,\mathcal{D}_1,\mathcal{D}_1)}$. Indeed, our original presentation of the system in [10] did not include these rules, but we have "bootstrapped" our system here in order to facilitate the definitions of our restricted "non-wellfounded" systems later for characterising $\mathbf{FP}/poly$, in particular in Section 3.

3 Non-wellfounded systems based on Bellantoni-Cook

In this section we recall a "coinductive" version of B that was recently introduced in our earlier work [10], and go on to introduce the new system nuB of this work. In particular we shall give global criteria that control the computational strength of non-wellfounded typing derivations. Throughout this section we shall work with the system $\mathsf{B}^- := \mathsf{B} - \{\mathsf{srec}\}$.

▶ **Definition 9** (Coderivations). A (B^--)*coderivation* $\mathcal{D}$ is a possibly infinite rooted tree generated by the rules of B^-. Formally, we identify $\mathcal{D}$ with a (labelled) prefix-closed subset of $\{0,1,2\}^*$ (i.e. a ternary tree). Each node is labelled by an inference step from B^- such that, whenever $\nu \in \mathcal{D}$ is labelled by a step $\dfrac{S_1 \quad \cdots \quad S_n}{S}$, for $n \leq 3$, ν has n children in $\mathcal{D}$ labelled by steps with conclusions $S_1, \ldots, S_n$ respectively. Sub-coderivations of a coderivation $\mathcal{D}$ rooted at position $\nu \in \{0,1,2\}^*$ are denoted $\mathcal{D}_\nu$, so that $\mathcal{D}_\varepsilon = \mathcal{D}$.

Examples of coderivations can be found in Figure 3 (some of them are from [10]), whose computational meaning is discussed in Example 13, and employ the following conventions:

▶ **Convention 10** (Representing coderivations). Henceforth, we may mark steps by $\bullet$ (or similar) in a coderivation to indicate roots of identical sub-coderivations. Moreover, to avoid ambiguities and to ease parsing of (co)derivations, we shall often underline principal formulas of a rule instance in a given coderivation and omit instances of structural rules e_N, $\mathsf{e}_\square$, w_N and $\mathsf{w}_\square$, absorbing them into other steps (typically cuts) when it causes no confusion. Finally, when the sub-coderivations $\mathcal{D}_0$ and $\mathcal{D}_1$ above the second and the third premise of the conditional rule (from left) are similar, we may compress them into a single "parametrised" sub-coderivation $\mathcal{D}_i$ (with $i = 0, 1$).

As discussed in [13, 11, 24], coderivations can be identified with Kleene-Herbrand-Gödel style equational programs, in general computing partial recursive functionals (see, e.g., [22, §63] for further details). We shall specialise this idea to our two-sorted setting.

▶ **Definition 11** (Semantics of coderivations). To each B^--coderivation $\mathcal{D}$ we associate a two-sorted Kleene-Herbrand-Gödel partial function $f_{\mathcal{D}}$ obtained by construing the semantics of Definition 6 as a (possibly infinite) equational program. Given a two-sorted function $f(\vec{x};\vec{y})$, we say that f is *defined* by a B^--coderivation $\mathcal{D}$ if $f_{\mathcal{D}}(\vec{x};\vec{y}) = f(\vec{x};\vec{y})$.

▶ Remark 12. The notion of *computation* for equational programs is given by (finitary) reasoning in equational logic (see, e.g., [22, §63]): for numerals $\vec{m}, \vec{n}$, we have that $f_{\mathcal{D}}(\vec{m};\vec{n})$ is well-defined and returns some numeral k just if the equation $f_{\mathcal{D}}(\vec{m};\vec{n}) = k$ can be (finitely)

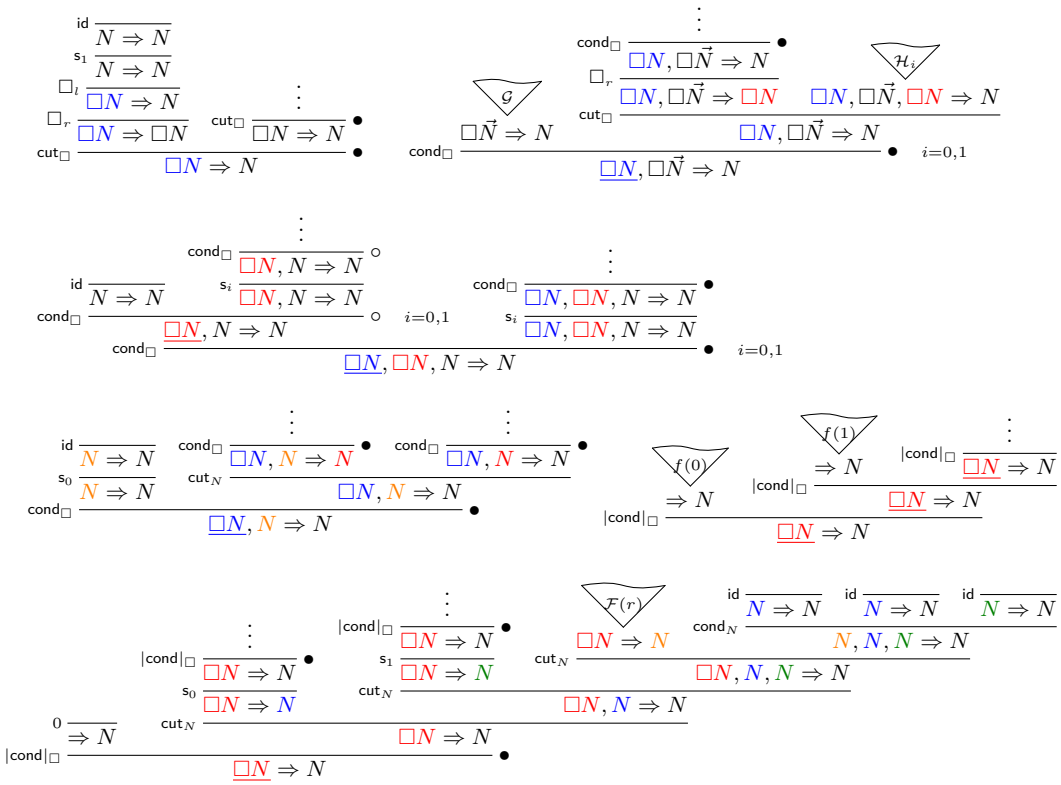


Figure 3 Examples of coderivations: $\mathcal{I}$ (top left), $\mathcal{R}$ (top right), $\mathcal{C}$ (second line), $\mathcal{E}$ (third line, left), $\mathcal{F}(f)$ with $f : \mathbb{N} \to \mathbb{N}$ (third line, right), $\mathcal{A}(r)$ with $r : \mathbb{N} \to \{0, 1\}$ (bottom).

derived in equational logic (with basic numerical axioms) over the equational program for $\mathcal{D}$. Implicit here is the fact that the semantics of B^--coderivations yield *coherent* equational programs: whenever $f_{\mathcal{D}}(\vec{m}; \vec{n}) = k$ and $f_{\mathcal{D}}(\vec{m}; \vec{n}) = k'$ are derivable then $k = k'$ [13, 11].

▶ **Example 13.** By purely equational reasoning, we can simplify the Kleene-Gödel-Herbrand style semantics in Definition 11 of the coderivations in Figure 3 to get the equational programs in Figure 4: $f_{\mathcal{I}}$ represents a function that is always undefined, as its equational program keeps increasing the length of the input; $f_{\mathcal{R}}$ is an instance of a *non-safe* recursion scheme (on notation), as the recursive call appears in normal position; $f_{\mathcal{C}}$ computes concatenation of the binary representation of three natural numbers; $f_{\mathcal{E}}$ has exponential growth rate (as long as $y \neq 0$), since $f_{\mathcal{E}}(x; y) = 2^{2^{|x|}} \cdot |y|$; the (infinite) equational program for $f_{\mathcal{F}(f)}$ computes $f(|x|)$ by simply exhausting the values of $|x|$; finally, $f_{\mathcal{A}(r)}$ on input x returns the binary string $r(0) \cdot r(1) \cdot \cdots \cdot r(|x| - 1)$ if $x > 0$, and 0 otherwise.

The above examples illustrate several recursion theoretic features of B^--coderivations that we shall seek to control in the remainder of this section:

- **(I)** *non-totality* (e.g., the coderivation $\mathcal{I}$);
- **(II)** *non-computability* (e.g., the coderivation $\mathcal{F}(f)$, with f non-computable);
- **(III)** *non-safety* (e.g., the coderivation $\mathcal{R}$), despite the presence of modalities implementing the normal/safe distinction of function arguments;
- **(IV)** *nested recursion* (e.g., the coderivation $\mathcal{E}$).

$$
\begin{aligned}
f_{\mathcal{I}}(x;) &= f_{\mathcal{I}}(\mathsf{s}_1 x;) & f_{\mathcal{E}}(0;y) &= \mathsf{s}_0(;y)\\
f_{\mathcal{R}}(0,\vec{x};) &= f_{\mathcal{G}}(\vec{x};) & f_{\mathcal{E}}(\mathsf{s}_i x;y) &= f_{\mathcal{E}}(x;f_{\mathcal{E}}(x;y))\\
f_{\mathcal{R}}(\mathsf{s}_i x,\vec{x};) &= f_{\mathcal{H}_i}(x,\vec{x},f_{\mathcal{R}}(x,\vec{x};);) & \{f_{\mathcal{F}(f)}(x;) &= f(|x|)\}_{|x|\in\mathbb{N}}\\
f_{\mathcal{C}}(0,0;z) &= z & f_{\mathcal{A}(r)}(0;) &= 0\\
f_{\mathcal{C}}(0,\mathsf{s}_i y;z) &= \mathsf{s}_i f_{\mathcal{C}}(0,y;z) & f_{\mathcal{A}(r)}(\mathsf{s}_i x;) &= \begin{cases}\mathsf{s}_0 f_{\mathcal{A}(r)}(x;) & \text{if } f_{\mathcal{F}(r)}(x;)=0\\ \mathsf{s}_1 f_{\mathcal{A}(r)}(x;) & \text{otherwise}\end{cases}\\
f_{\mathcal{C}}(\mathsf{s}_i x,y;z) &= \mathsf{s}_i f_{\mathcal{C}}(x,y;z)
\end{aligned}
$$

Figure 4 Equational programs derived from the coderivations in Figure 3, where $i \in \{0,1\}$.

To address (I) we shall adapt to our setting a well-known "totality criterion" from non-wellfounded proof theory (similar to those in [13, 11, 24]). First we need to recall some standard structural proof theoretic notions:

▶ **Definition 14** (Ancestry)**.** Fix a coderivation $\mathcal{D}$. We say that a type occurrence A is an *immediate ancestor* of a type occurrence B in $\mathcal{D}$ if they are types in a premiss and conclusion (respectively) of an inference step and, as typeset in Figure 1, have the same colour. If A and B are in some Γ or Γ', then furthermore they must be in the same position in the list.

For a definition of immediate ancestry avoiding colours, we point the reader to standard proof theory references, e.g. [9, Sec. 1.2.3]. Being a binary relation, immediate ancestry forms a directed graph upon which our totality criterion is built:

▶ **Definition 15** (Progressing coderivations)**.** Fix a coderivation $\mathcal{D}$. A *thread* is a maximal path in the graph of immediate ancestry. We say that a (infinite) thread is *progressing* if it is eventually constant $\Box N$ and infinitely often principal for a $\mathsf{cond}_\Box$ rule or a $|\mathsf{cond}|_\Box$ rule. A coderivation is *progressing* if each of its infinite branches has a progressing thread.

In [10] we showed that the progressing criterion is indeed sufficient (but obviously not necessary) to guarantee that the partial function computed by a coderivation is, in fact, total (see also [24, 11, 13]):

▶ **Proposition 16** (Progressing implies totality, [10])**.** *If $\mathcal{D}$ is progressing, then $f_{\mathcal{D}}$ is total.*

The argument for this proposition is by contradiction: assuming non-totality, construct an infinite "non-total branch", whence a contradiction to well-orderedness of $\mathbb{N}$ is implied by a progressing thread along it. We shall use similar argument later in the proof of Lemma 37.

▶ **Example 17.** In Figure 3, $\mathcal{I}$ has precisely one infinite branch (that loops on $\bullet$) which contains no instances of $\mathsf{cond}_\Box$ or $|\mathsf{cond}|_\Box$ at all, so $\mathcal{I}$ is not progressing. On the other hand, $\mathcal{C}$ has two simple loops, one on $\bullet$ and the other one on $\circ$. For any infinite branch B we have two cases: if B crosses the bottommost conditional infinitely many times, it contains a progressing blue thread; otherwise, B crosses the topmost conditional infinitely many times, so that it contains a progressing red thread. Therefore, $\mathcal{C}$ is progressing. By applying the same reasoning, we conclude that $\mathcal{E}$, $\mathcal{F}(f)$, $\mathcal{A}(r)$, and $\mathcal{R}$ are progressing (if $\mathcal{G}$ and $\mathcal{H}_i$ are).

To address (III)-(IV) we recall the following properties of coderivations from [10]:

▶ **Definition 18** (Safety, left-leaning)**.** We say that a coderivation $\mathcal{D}$ is *safe* if each branch crosses only finitely many $\mathsf{cut}_\Box$-steps, and *left-leaning* if each branch goes right at a cut_N-step only finitely often.

▶ **Example 19.** In Figure 3, the only non-safe coderivations are $\mathcal{R}$ and $\mathcal{I}$, as the branches looping on $\bullet$ contain infinitely many $\mathsf{cut}_\square$. $\mathcal{E}$ is the only non-left-leaning coderivation, as it has a branch looping at $\bullet$ that crosses infinitely many times the rightmost premise of a cut_N.

Finally, concerning (II), recall that the aim of this work is to characterise non-uniform classes, which may contain non-computable predicates and functions. To this end we introduce a generalisation of the notion of "regularity", typically corresponding to computability (e.g. in [11, 13, 24]), that is commonplace in cyclic proof theory:

▶ **Definition 20** (Generalised regularity)**.** Let $\mathsf{R} \subseteq \mathsf{B}^-$. A B^--coderivation $\mathcal{D}$ is R-*regular* if it has only finitely many distinct sub-coderivations containing rules among R. If $\mathsf{R} = \mathsf{B}^-$, i.e. it has only finitely many distinct sub-coderivations, then we say that $\mathcal{D}$ is *regular* (or *circular*).

Note that, while usual derivations may be naturally written as finite trees or dags, regular coderivations may be naturally written as finite directed (possibly cyclic) graphs. Also, from a regular coderivation $\mathcal{D}$ we obtain a *finite* equational program for $f_\mathcal{D}$. In particular, while there are continuum many (non-wellfounded) coderivations, there are only countably many regular ones.

▶ **Example 21.** In Figure 3, $\mathcal{F}(f)$ and $\mathcal{A}(r)$ are the only non-regular coderivations (as long as $\mathcal{G}$, $\mathcal{H}_i$ are regular). Also, $\mathcal{A}(r)$ is R-regular for any $\mathsf{R} \subseteq \mathsf{B}^- - \{0, 1, |\mathsf{cond}|_\square\}$, since $r(i)$ is computed by just a 0 or 1 step when $r : \mathbb{N} \to \{0, 1\}$.

We are now ready to present the non-wellfounded proof systems that will be considered in this paper:

▶ **Definition 22** (CB and nuB)**.** CB is the class of regular progressing safe and left-leaning B^--coderivations. nuB is the class of $\{\mathsf{cond}_\square, \mathsf{cond}_N, \mathsf{s}_0, \mathsf{s}_1, \mathsf{id}\}$-regular progressing safe and left-leaning B^--coderivations. A two-sorted function $f(\vec{x}; \vec{y})$ is CB-*definable* (resp. nuB-*definable*) if there is a coderivation $\mathcal{D} \in \mathsf{CB}$ (resp. $\mathcal{D} \in \mathsf{nuB}$) such that $f_\mathcal{D}(\vec{x}; \vec{y}) = f(\vec{x}; \vec{y})$.

Recalling Examples 17, 19 and 21, $\mathcal{C}$ is the only coderivation in CB, while $\mathcal{A}(r)$ is an example of coderivation in nuB for any $r : \mathbb{N} \to \{0, 1\}$. The system CB was already introduced in [10], where we showed that $\mathsf{CB} = \mathbf{FP}$ (among other results), whereas nuB (read "non-uniform B") is new. The main result of this paper is to show that nuB admits just the right amount of non-wellfoundedness to duly characterise the analogous non-uniform class:

▶ **Theorem 23.** $\mathsf{nuB} = \mathbf{FP}/\mathit{poly}$

The rest of this work is devoted to the proof of this result. In particular, the two directions of the equality are given by Corollary 35 and Corollary 44.

▶ Remark 24 (On proof checking). Let us point out that all conditions on coderivations we have considered so far are *decidable* on regular coderivations. In particular, progressiveness may be decided by reduction to universality of Büchi automata. In the presence of safety, however, it turns out that proof checking becomes easier: checking whether a regular coderivation is in CB is actually decidable in $\mathbf{NL}$ [10, Cor. 32]. Of course, as nuB coderivations are not finitely presented (indeed like $\mathbf{FP}/\mathit{poly}$ programs), such decidability issues are no longer relevant.

4 On relativised characterisations of $\mathbf{FP}/\mathit{poly}$

In this section we consider recursion theoretic characterisations of $\mathbf{FP}/\mathit{poly}$ via relativised function algebras. This will serve not only as a "warm up" to motivate our main characterisation, but will also provide several of the intermediate results necessary to that end. The results of this section are based on textbook techniques and are (presumably) folklore.

4.1 Non-uniformity via resource-bounded oracle machines

A *relation* is a function $r(\vec{x})$ such that we always have $r(\vec{x}) \in \{0,1\}$.

▶ **Definition 25** (Relativised complexity classes)**.** Let R be a set of relations. The class $\mathbf{FP}(R)$ consists of just the functions computable in polynomial time by a Turing machine with access to an oracle for each $r \in R$.

For instance, using this notion of relativised computation, we can define the levels of the functional polynomial hierarchy **FPH** by $\square_1^p := \mathbf{FP}$, $\square_2^p := \mathbf{FP}(\mathbf{NP})$, $\square_3^p := \mathbf{FP}(\Sigma_2^p)$, etc.

Let us write $\mathbb{R} := \{r : \mathbb{N}^k \to \{0,1\} \mid |\vec{x}| = |\vec{y}| \implies r(\vec{x}) = r(\vec{y})\}$. Note that the notation $\mathbb{R}$ is suggestive here, since its elements are essentially maps from lengths/positions to Booleans, and so may be identified with Boolean streams.

▶ **Proposition 26.** $\mathbf{FP}/poly = \mathbf{FP}(\mathbb{R})$.

Proof sketch. For the left-right inclusion, let $p(n)$ be a polynomial and $\mathbf{C} = (C_n)_{n<\omega}$ be a circuit family with each C_n taking n Boolean inputs and having size $< p(n)$. We need to show that the language computed by $\mathbf{C}$ is also computed in $\mathbf{FP}(\mathbb{R})$. Let $c \in \mathbb{R}$ be the function that, on inputs x, y returns the $|y|^{\text{th}}$ bit of $C_{|x|}$. Using this oracle we can compute $C_{|x|}$ by polynomially queries to c, and this may be evaluated as usual using a polynomial-time evaluator in $\mathbf{FP}$.

For the right-left inclusion, notice that a polynomial-time machine can only make polynomially many calls to oracles with inputs of only polynomial size. Thus, if $f \in \mathbf{FP}(\mathbb{R})$ then there is some p_f with $f \in \mathbf{FP}(\mathbb{R}^{<p_f})$, where $\mathbb{R}^{<p_f}$ is the restriction of each $r \in \mathbb{R}$ to only its first $p_f(|\vec{x}|)$ many bits. Now, since f can only call a fixed number of oracles from $\mathbb{R}$, we can collect these finitely many polynomial-length prefixes into a single advice string for computation in $\mathbf{FP}/poly$. ◀

4.2 A relativised Bellantoni-Cook characterisation of $\mathrm{FP}/poly$

We shall employ the following writing conventions for the remainder of this work. For a set of (single-sorted) functions F, let us write:

- $F_{1;0}$ for the set of two-sorted functions $f(\vec{x};)$ for each $f(\vec{x}) \in F$;
- $F_{1;1}$ for the set of two-sorted functions $f(\vec{x};\vec{y})$ for each $f(\vec{x},\vec{y}) \in F$.

Given a set F of two-sorted functions, the algebra $\mathsf{B}(F)$ is defined just like B but with additional initial (two-sorted) functions F. Note that, since functions of $\mathsf{B}(F)$ are given by finite programs, they can only depend on finitely many members of F.

▶ **Proposition 27.** $\mathbf{FP}/poly \subseteq \mathsf{B}(\mathbb{R}_{1;0})$

One natural way to prove this result would be to go via $\mathbf{FP}(\mathbb{R})$, in light of Proposition 26. Indeed Bellantoni established foundational results relating $\mathbf{FP}(R)$ and versions of $\mathsf{B}(R)$, for R a set of relations, in [3], but unfortunately the sorting of the corresponding arguments is subtle and does not immediately give the result we are after. For this reason we give a direct proof, that nonetheless inlines some ideas from [3].

Proof of Proposition 27. Let $\mathbf{C} = (C_n)_{n<\omega}$ be a circuit family with each C_n taking n inputs and having size $< p(n)$, for some (monotone) polynomial p. We need to show that the language computed by $\mathbf{C}$ is also computed in $\mathsf{B}(\mathbb{R}_{1;0})$.

First, let $\mathrm{Eval}(x,y)$ evaluate the circuit described by x on the input y. Since $\mathrm{Eval} \in \mathbf{FP}$, we have as standard (e.g. by [4, Lemma 3.2]) a function $\mathrm{Eval}(m;x,y) \in \mathsf{B}$ and a monotone polynomial q such that $|m| \geq q(|x|,|y|) \implies \mathrm{Eval}(m;x,y) = \mathrm{Eval}(x,y)$. Now,

in particular, if x is the description of some C_n and $n = |y|$, then also $|x| \leq p(|y|)$, and so $|m| \geq q(p(|y|), |y|) \implies \mathrm{Eval}(m; x, y) = \mathrm{Eval}(x, y)$. Finally, denoting $\overbrace{\mathsf{s}_1 \dots \mathsf{s}_1}^{n} 0$ by 1^n, this means that we have $\mathrm{Eval}(y; x) := \mathrm{Eval}(1^{q(p(|y|),|y|)}; x, y) \in \mathsf{B}$, that in particular evaluates, when x describes $C_{|y|}$, the circuit $C_{|y|}$ on input y.

Now, let $c \in \mathbb{R}_{1;0}$ with $c(y, z;) = |z|^{\text{th}}$ bit of $C_{|y|}$. We show that the function $C(y, z;) = c(y, 0;) \cdot c(y, 1;) \cdot \dots \cdot c(y, 1^{|z|-1};)$ is in $\mathsf{B}(c)$ by the following instance of safe recursion:

$$\begin{aligned} C(y, 0;) &= 0 \\ C(y, \mathsf{s}_i z;) &= \mathsf{cond}(; c(y, z;), \mathsf{s}_0(; C(y, z;)), \mathsf{s}_1(; C(y, z;))) \end{aligned}$$

So we have that $C(y;) := C(y, 1^{p(|y|)};)$ computes the description of $C_{|y|}$. Now we can decide whether y is accepted by $C_{|y|}$ simply by calling the function $\mathrm{Eval}(y; C(y;)) \in \mathsf{B}(c)$. ◀

It turns out that we also have the converse inclusion too. This will be subsumed by our later results but we include it here for the sake of completeness. The key is to establish a general form of Bellantoni and Cook's polymax bounding lemma to account for modulus of continuity as well as growth:

▶ **Lemma 28** (Relational bounding lemma for B)**.** *Let R be a set of two-sorted relations, and suppose $f(R)(\vec{x}; \vec{y}) \in \mathsf{B}(R)$. Then there is a polynomial p_f such that, setting $m_f(\vec{m}, \vec{n}) := p_f(\vec{m}) + \max \vec{n}$, we have:*

- *(Polynomial modulus of growth) $|f(R)(\vec{x}; \vec{y})| < m_f(|\vec{x}|, |\vec{y}|)$*
- *(Polynomial modulus of continuity) $f(R)(\vec{x}; \vec{y}) = f(\lambda|\vec{u}|, |\vec{v}| < m_f(|\vec{x}|, |\vec{y}|).r(\vec{u}; \vec{v}))_{r \in R}(\vec{x}; \vec{y})$*

Using this we may establish:

▶ **Proposition 29** (E.g. see [3])**.** *Let R be a set of relations.* $\mathsf{B}(R_{1;1}) \subseteq \mathbf{FP}(R)$.

We shall not actually need this result directly in this work, rather recovering (a version of) it from a more refined grand tour of inclusions. However this does lead to the first "implicit" characterisation of $\mathbf{FP}/poly$ of this work:

▶ **Corollary 30.** $\mathsf{B}(\mathbb{R}_{1;0}) = \mathbf{FP}/poly$

5 $\mathbf{FP}/poly \subseteq$ nuB via relativised circular systems

In this section we establish one direction of Theorem 23. In particular, by the end of this section, we will have established the following inclusions,

$$\mathbf{FP}/poly \subseteq \mathsf{B}(\mathbb{R}_{1;0}) \subseteq \mathsf{CB}(\mathbb{R}_{1;0}) \subseteq \mathsf{nuB}$$

where $\mathsf{CB}(F)$ is an extension of CB by new initial sequents for two-sorted functions in F.

5.1 Relativised simulation of B in CB

We shall consider "relativised" versions of CB, that may include new initial sequents. Formally:

▶ **Definition 31.** Let F be a set of two-sorted functions. A $\mathsf{B}^-(F)$-coderivation is just a usual B^--coderivation that may use initial sequents of the form $f \dfrac{}{\Box N^{n_i}, N^{m_i} \Rightarrow N}$, when $f \in F$ takes n_i normal and m_i safe inputs. We write $\mathsf{CB}(F)$ for the set of CB-coderivations allowing initial sequents from F. The semantics of such coderivations and the notion of $\mathsf{CB}(F)$-definability are as expected, with $f_{\mathcal{D}(F)}$ denoting the induced interpretation of $\mathcal{D}(F) \in \mathsf{CB}(F)$.

Note, again, that since $\mathsf{CB}(F)$ coderivations are regular, they only depend on finitely many members of F. By a modular extension of the result that $\mathsf{B} \subseteq \mathsf{CB}$ from [10], we obtain:

▶ **Proposition 32.** *Let F be a set of two-sorted functions.* $\mathsf{B}(F) \subseteq \mathsf{CB}(F)$.

The proof is simply by structural induction on the definition of a $\mathsf{B}(F)$ function, where the recursion cases are handled by circularity as in [10]. In particular, if f is defined by safe recursion on notation from g, h_0, h_1 then the corresponding CB-coderivation is given by:

$$\dfrac{\Gamma \Rightarrow N \quad \dfrac{\vdots}{\Box N, \Gamma \Rightarrow N}\,\mathsf{cond}_\Box \quad \dfrac{\Box N, \Gamma, N \Rightarrow N}{\Box N, \Gamma \Rightarrow N}\,\mathsf{cut}_N \; h_i}{\Box N, \Gamma \Rightarrow N}\,\mathsf{cond}_\Box \; g \qquad i = 0, 1$$

The only new cases in the induction are for an initial function from F, which is simply translated into the appropriate initial sequent.

5.2 Simulating $\mathbb{R}_{1;0}$ oracles in nuB

In this subsection we shall establish:

▶ **Proposition 33.** $\mathsf{CB}(\mathbb{R}_{1;0}) \subseteq \mathsf{nuB}$.

By definition of nuB and CB, it suffices to only consider the new initial sequents from $\mathbb{R}_{1;0}$. For this we simply appeal to the following lemma:

▶ **Lemma 34.** *For each $r(\vec{x};) \in \mathbb{R}_{1;0}$, there is a nuB-coderivation defining it, in particular using only the rules $0, 1, |\mathsf{cond}|_\Box$.*

Proof. We proceed by induction on the length of $\vec{x}$. When the list is empty, then $r(;)$ is just a Boolean, in which case we can derive it with just the 0 or 1 step. Now, for $r(x, \vec{x};)$ we have:

$$\dfrac{\Box\vec{N} \Rightarrow N \quad \dfrac{\Box\vec{N} \Rightarrow N \qquad \vdots}{\underline{\Box N}, \Box\vec{N} \Rightarrow N}\,|\mathsf{cond}|_\Box \; \mathrm{IH}(r_1)}{\underline{\Box N}, \Box\vec{N} \Rightarrow N}\,|\mathsf{cond}|_\Box \; \mathrm{IH}(r_0)$$

where $r_i(\vec{x})$ is the function $r(1^i, \vec{x})$ and the coderivations marked $\mathrm{IH}(r_i)$ are obtained by the inductive hypothesis for r_i. ◀

Note that, by putting together Proposition 27, Proposition 32 (setting $F = \mathbb{R}_{1;0}$) and Proposition 33, we now have one half of our main result:

▶ **Corollary 35.** $\mathbf{FP}/poly \subseteq \mathsf{nuB}$

6 nuB as relativised regular coderivations

To facilitate the other direction of Theorem 23, let us first address a form of converse to Proposition 33 above, that duly embeds nuB into a relativised circular system, which we shall rely on in the next section:

▶ **Theorem 36.** $\mathsf{nuB} \subseteq \mathsf{CB}(\mathbb{R}_{1;1})$.

Before giving the proof, we need to first establish some intermediate results:

▶ **Lemma 37.** *If $\mathcal{D}$ is progressing and $\{\mathsf{s}_0, \mathsf{s}_1, \mathsf{id}\}$-free then $f_\mathcal{D}$ is a relation, i.e. $f_\mathcal{D}(\vec{x};\vec{y}) \leq 1$.*

Proof sketch. We proceed by contradiction, always assuming Proposition 16, that progressing coderivations compute total functions.

If $f_\mathcal{D}(\vec{x};\vec{y}) > 1$ then we argue that there is an immediate sub-coderivation $\mathcal{D}'$ of $\mathcal{D}$ and arguments $\vec{x}', \vec{y}'$ such that $f_{\mathcal{D}'}(\vec{x}';\vec{y}') > 1$. Some of the critical cases are:

- If $\mathcal{D} = \mathsf{cut}_N(\mathcal{D}_0, \mathcal{D}_1)$ then $f_\mathcal{D}(\vec{x};\vec{y}) = f_{\mathcal{D}_1}(\vec{x};\vec{y}, f_{\mathcal{D}_0}(\vec{x};\vec{y}))$, and we set $\mathcal{D}' := \mathcal{D}_1$, $\vec{x}' := \vec{x}$ and $\vec{y}' := \vec{y}, f_{\mathcal{D}_0}(\vec{x};\vec{y})$ (since $f_{\mathcal{D}_0}(\vec{x};\vec{y})$ is well-defined). The case for $\mathsf{cut}_\Box$ is similar.
- If $\mathcal{D} = \mathsf{cond}_\Box(\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2)$ then $\vec{x} = x, \vec{z}$ with $f_\mathcal{D}(0, \vec{z};\vec{y}) = \mathcal{D}_0(\vec{z};\vec{y})$ and $f_\mathcal{D}(\mathsf{s}_i x', \vec{z};\vec{y}) = f_{\mathcal{D}_{i+1}}(x', \vec{z};\vec{y})$. If $x = 0$ we set $\mathcal{D}' := \mathcal{D}_0$, $\vec{x}' := \vec{z}$, and $\vec{y}' := \vec{y}$. If $x = \mathsf{s}_i x'$ then we set $\mathcal{D}' := \mathcal{D}_{i+1}$, $\vec{x}' := x', \vec{z}$, and $\vec{y}' := \vec{y}$. The cases for cond_N, $|\mathsf{cond}|_N$, and $|\mathsf{cond}|_\Box$ are similar.
- In all other cases $\mathcal{D}$ ends with a unary rule so that $\mathcal{D}'$ is the only immediate sub-coderivation, and $\vec{x}', \vec{y}'$ are determined by the semantics of the rule (cf. Figure 2).

Note that, in the absence of $\mathsf{s}_0, \mathsf{s}_1$, we indeed have that $f'(\vec{x}';\vec{y}') = f(\vec{x};\vec{y}) > 1$ so we may continually apply this process to build up a branch $B = (\mathcal{D} = \mathcal{D}^{(0)}, \mathcal{D}^{(1)}, \mathcal{D}^{(2)}, \dots)$ and arguments $(\vec{x};\vec{y}) = (\vec{x}^{(0)};\vec{y}^{(0)}), (\vec{x}^{(1)};\vec{y}^{(1)}), (\vec{x}^{(2)};\vec{y}^{(2)}), \dots$ such that $f_{\mathcal{D}^{(k)}}(\vec{x}^{(k)};\vec{y}^{(k)}) = f_\mathcal{D}(\vec{x};\vec{y}) > 1$. Observe that B cannot end at an id step, by assumption that $\mathcal{D}$ is id-free. Also, if B ends at a 0 or 1 step we have by construction that $f_\mathcal{D}(\vec{x};\vec{y}) \in \{0, 1\}$, a contradiction. Thus B must be infinite. Since $\mathcal{D}$ is progressing there is a progressing thread along B, say $(\Box N^i)_{i \geq k}$, where each $\Box N^i$ is an occurrence of $\Box N$. Let us examine the values, say x^i, assigned to each $\Box N^i$. Notice that:

- by inspection of the rules and their interpretations from Definition 11, we have that $x^{i+1} \leq x^i$; and,
- if $\Box N^i$ is principal for a $\mathsf{cond}_\Box$ or a $|\mathsf{cond}|_\Box$ step then $x^{i+1} < x^i$.

If follows that $(x^i)_{i \geq k}$ is a non-increasing sequence of natural numbers that does not converge, contradicting the well-ordering property of $\mathbb{N}$. ◀

▶ **Lemma 38.** *If $\mathcal{D}$ is $\{\mathsf{cond}_\Box, \mathsf{cond}_N, \mathsf{id}\}$-free, $|\vec{x}| = |\vec{x}'|$ and $|\vec{y}| = |\vec{y}'|$, then $f_\mathcal{D}(\vec{x};\vec{y}) = f_\mathcal{D}(\vec{x}';\vec{y}')$, whenever $f(\vec{x};\vec{y})$ is well-defined.*

Proof sketch. Being given by an equational program, we have that $f_\mathcal{D}(\vec{x};\vec{y}) = m$ has a (finite) equational derivation for some $m \in \mathbb{N}$, by assumption that it is well-defined (cf. Remark 12). Replacing $\vec{x}, \vec{y}$ by $\vec{x}', \vec{y}'$ in this derivation yields $f_\mathcal{D}(\vec{x}';\vec{y}') = m$ too. The only critical cases are the steps $|\mathsf{cond}|_N$, $|\mathsf{cond}|_\Box$, whose semantics only depend on the length of their arguments. ◀

Putting the two above Lemmata together we have:

▶ **Proposition 39.** *If $\mathcal{D}$ is progressing and $\{\mathsf{cond}_\Box, \mathsf{cond}_N, \mathsf{s}_0, \mathsf{s}_1, \mathsf{id}\}$-free, then $f_\mathcal{D} \in \mathbb{R}_{1;1}$.*

Now we can prove Theorem 36:

Proof sketch. Let $\mathcal{D}$ be a nuB-coderivation and let V be the set of minimal nodes ν such that $\mathcal{D}_\nu$ is $\{\mathsf{cond}_\Box, \mathsf{cond}_N, \mathsf{s}_0, \mathsf{s}_1, \mathsf{id}\}$-free, and so by Proposition 39 we have that each $f_{\mathcal{D}_\nu} \in \mathbb{R}_{1;1}$.

Now, let $\mathcal{D}^V$ be obtained from $\mathcal{D}$ by simply deleting each sub-coderivation $\mathcal{D}_\nu$, for $\nu \in V$, and construing each of their conclusions as new initial sequents. By definition of nuB, note that $\mathcal{D}^V$ is now a coderivation in $\mathsf{CB}(f_{\mathcal{D}_\nu})_{\nu \in V} \subseteq \mathsf{CB}(\mathbb{R}_{1;1})$ and we are done. ◀

7 nuB $\subseteq$ FP/*poly*: a relativised algebra subsuming circular typing

The final part of our chain of inclusions requires us to translate (relativised) circular coderivations into an appropriate function algebra. The idea is that, in the presence of safety, one can reduce circularity to a form of recursion on "permutations of prefixes" that nonetheless remains feasible. This was (one of) the main result(s) of [10] and, fortunately, we are able to import those results accounting only for additional initial relations.

7.1 Safe recursion on permutations of prefixes

Let us write $\vec{x} \subseteq \vec{y}$ if $\vec{x}$ is a permutation of prefixes of $\vec{y}$, i.e. $\vec{x} = x_0, \dots, x_{n-1}$ and $\vec{y} = y_0, \dots, y_{n-1}$ and there is a permutation $\pi : [n] \to [n]$ s.t. each x_i is a prefix of $y_{\pi i}$. We shall write $\vec{x} \subset \vec{y}$ if for at least one $i < n$ we have that x_i is a strict prefix of $y_{\pi i}$. Note in particular that $\subset$ is a well-founded pre-order so admits an induction and recursion principles.

To formulate recursion over well-founded relations it is convenient to employ (two-sorted) oracles as placeholders for recursive calls. Due to the necessary constraints on composition, we shall formally distinguish these oracles (metavariables, a, b, etc.) from additional initial functions (metavariables f, g etc. until now).

▶ **Definition 40.** Let F be a set of two-sorted functions. The algebra $\mathsf{B}^{\subset}(F, \vec{a})$ is the smallest class of two-sorted functions containing,

- all the initial functions of B;
- each (two-sorted) function a_i among $\vec{a}$;
- each (two-sorted) function $f \in F$;

and closed under:

- (Relativised safe composition)
  - if $g(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(F, \vec{a})$ and $h(\vec{x};\vec{y}, y) \in \mathsf{B}^{\subset}(F, \vec{a})$ then $f(\vec{x};\vec{y}) := h(\vec{x};\vec{y}, g(\vec{x};\vec{y})) \in \mathsf{B}^{\subset}(F, \vec{a})$;
  - if $g(\vec{x};) \in \mathsf{B}^{\subset}(F)$ and $h(\vec{x}, x;\vec{y}) \in \mathsf{B}^{\subset}(F, \vec{a})$ then $f(\vec{x};\vec{y}) := h(\vec{x}, g(\vec{x};);\vec{y}) \in \mathsf{B}^{\subset}(F, \vec{a})$;
- (Safe recursion on $\subset$)
  if $h(a)(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(F, a, \vec{a})$ then $f(\vec{x};\vec{y}) := h(\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}.f(\vec{u};\vec{v}))(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(F, \vec{a})$.

To be clear, the "guarded" abstraction notation above is formally defined as

$$(\lambda\vec{u} \subset \vec{x}, \lambda\vec{v} \subseteq \vec{y}.f(\vec{u};\vec{v}))\,(\vec{u}';\vec{v}') \;:=\; \begin{cases} f(\vec{u}';\vec{v}') & \vec{u}' \subset \vec{x}, \vec{v}' \subseteq \vec{y} \\ 0 & \text{otherwise} \end{cases}$$

$\mathsf{B}^{\subset}(\varnothing, \vec{a})$ is the same as the notion $\mathsf{B}^{\subset}(\vec{a})$ from [10]. Note in particular the distinction between F and $\vec{a}$ in the safe composition scheme: when composing along a normal parameter (second line), the function $g(\vec{x};)$ must not contain any oracles among $\vec{a}$.

Adapting the *Bounding Lemma* from [10, Lemma 38] to account for further initial relations gives:

▶ **Lemma 41** (Relational Bounding lemma)**.** *Let R be a set of two-sorted relations and $f(R)(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(R)$. There is a polynomial $p_f(\vec{n})$ such that, writing $m_f(\vec{x},\vec{y}) = p_f(|\vec{x}|) + \max|\vec{y}|$, we have:*

- $|f(R)(\vec{x};\vec{y})| < m_f(\vec{x},\vec{y})$
- $f(R)(\vec{x};\vec{y}) = f(\lambda|\vec{u}_r|, |\vec{v}_r| < m_f(\vec{x},\vec{y}).r(\vec{u}_r;\vec{v}_r))_{r\in R}(\vec{x};\vec{y})$

The first point is common to implicit complexity, being essentially Bellantoni and Cook's "polymax bounding lemma" from [4]. The second point expresses a dual property: while the first bounds the modulus of *growth*, the second bounds the modulus of *continuity*.

Here it is important the the new initial functions are relations, or at least that they have constant/limited growth rate. In fact, for the proof, cf. [10], one needs a more complicated statement accounting for growth properties of the intermediate oracles $\vec{a}$ used for recursion, even though we only ultimately need the statement above for our purposes, once all such oracles $\vec{a}$ are "discharged".

Using the Bounding Lemma we have from [10] (again accounting for further initial relations) the main characterisation result for $\mathsf{B}^{\subset}$:

▶ **Proposition 42** (Relativised characterisation). *For a set R of relations,* $\mathsf{B}^{\subset}(R_{1;1}) \subseteq \mathbf{FP}(R)$.

The main point for proving this result is that the graph of $\subseteq$ is relatively small, in particular for each $\vec{y}$ there are only polynomially many $\vec{x} \subseteq \vec{y}$.[1] So we can calculate a function $f(\vec{x};\vec{y}) \in \mathsf{B}^{\subset}(R)$ simply by polynomial-time induction (at the meta level) on $\subset$, storing all previous values in a lookup table. This table will have only polynomially many entries, by the previous observation about the size of the graph of $\subseteq$, and each entry will have only polynomial size by the Bounding Lemma.

7.2 $\mathsf{CB}(\mathbb{R}_{1;1}) \subseteq \mathsf{B}^{\subset}(\mathbb{R}_{1;1})$: from circular proofs to recursive functions

The point of $\mathsf{B}^{\subset}$ in [10] was to play the role of a target algebra to translate circular coderivations into. The *Translation Lemma* from that work [10, Lemma 47], accounting for further initial relations, gives:

▶ **Lemma 43** (Relativised translation). *Let R be a set of relations.* $\mathsf{CB}(R_{1;1}) \subseteq \mathsf{B}^{\subset}(R_{1;1})$.

Note that we specialise the statement above only to sets of relations to avoid size issues potentially caused be new initial funtions of arbitrary growth rate. In fact, this proof requires closure of $\mathsf{B}^{\subset}(F,\vec{a})$ under a *simultaneous* version of its recursion scheme, upon which a careful translation from circular coderivations in "cycle normal form" (see, e.g., [7, Definition 6.2.1]) to an equational specification can be duly resolved in $\mathsf{B}^{\subset}$.

Now by setting $R = \mathbb{R}$, we have the following consequence of Proposition 42:

▶ **Corollary 44.** $\mathsf{nuB} \subseteq \mathbf{FP}/\mathit{poly}$

Proof. We have $\mathsf{nuB} \subseteq \mathsf{CB}(\mathbb{R}_{1;1})$ by Theorem 36, $\mathsf{CB}(\mathbb{R}_{1;1}) \subseteq \mathsf{B}^{\subset}(\mathbb{R}_{1;1})$ by Lemma 43, $\mathsf{B}^{\subset}(\mathbb{R}_{1;1}) \subseteq \mathbf{FP}(\mathbb{R})$ by Proposition 42, and finally $\mathbf{FP}(\mathbb{R}) \subseteq \mathbf{FP}/\mathit{poly}$ by Proposition 26. ◀

Along with Corollary 35, we have now established both directions of our main result Theorem 23 that $\mathsf{nuB} = \mathbf{FP}/\mathit{poly}$, hence completing the proof.

8 Conclusions

In this work we presented the two-sorted non-wellfounded proof system nuB and proved that it characterises the complexity class $\mathbf{FP}/\mathit{poly}$. Our results build on previous work [10], where we defined the cyclic proof systems CB and CNB capturing, respectively, $\mathbf{FP}$ and

[1] Of course, this polynomial depends on the length of $\vec{y}$, but for a given function of the algebra this is some global constant.

FELEMENTARY [10]. The system nuB is obtained from CB by associating non-uniformity in computation to a form of non-wellfoundedness in proof theory. To establish the characterisation theorems, we also formalised some (presumably) folklore results on relativised function algebras for $\mathbf{FP}/poly$.

For future research, the first author is investigating non-wellfounded approaches to $\mathbf{FP}/poly$ in the setting of linear logic [19]. In particular, we are studying a non-wellfounded version of Mazza's Parsimonious Logic [28], a variant of linear logic where the exponential modality ! satisfies Milner's law ($!A \simeq !A \otimes A$). This provides a natural computational interpretation of formulas $!A$ as types of streams on A. Mazza showed in [29] that Parsimonious Logic can be used to capture $\mathbf{P}/poly$ using *wellfounded* proofs that are essentially *infinitely branching*. We conjecture that a similar characterisation can be obtained in a non-wellfounded (and finitely branching) setting, using ideas from this work.

Another direction is to explore applications of the results of this paper to probabilistic complexity. In particular, we aim to study fragments of nuB modelling the class **BPP** (bounded-error probabilistic polynomial time), essentially by leveraging on well-known derandomisation methods showing the inclusion of **BPP** in $\mathbf{FP}/poly$, and hence in $\mathbf{FP}(\mathbb{R})$ (see Proposition 26). A challenging aspect of this task is to obtain characterisation results that are entirely in the style of ICC, since **BPP** is defined by explicit (error) bounds, as observed in [25]. We suspect that nuB represents the right framework for investigating fully implicit characterisations of this class, where additional proof-theoretic conditions can be introduced to restrict computationally the access to oracles and, consequently, to model bounded-error probabilistic computation.

As [10] also established a system CNB for **FELEMENTARY**, it would be pertinent to ask whether ideas in this work can be applied to CNB to characterise $\mathbf{FELEMENTARY}/poly$, i.e., the class of functions computable in elementary time by a Turing machine with access to a polynomial advice. Unfortunately, the modulus of continuity established for CNB in [10] is super-polynomial (indeed elementary), meaning that the same technique, a priori, would not restrict computation to only polynomial advice. Consideration of this issue is left to future research.

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

2 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 für Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.42.

3 Stephen Bellantoni. Predicative recursion and the polytime hierarchy. In Peter Clote and Jeffrey B. Remmel, editors, *Feasible Mathematics II*, pages 15–29, Boston, MA, 1995. Birkhäuser Boston.

4 Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the polytime functions (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 283–293, New York, NY, USA, 1992. Association for Computing Machinery. doi:10.1145/129712.129740.

5 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.

6 Stefano Berardi and Makoto Tatsuta. Classical system of Martin-Lof's inductive definitions is not equivalent to cyclic proofs. *Log. Methods Comput. Sci.*, 15(3), 2019. doi:10.23638/LMCS-15(3:10)2019.

7 James Brotherston. *Sequent calculus proof systems for inductive definitions*. PhD thesis, University of Edinburgh, 2006. PhD thesis.

8 James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, 2011.

9 Samuel R. Buss. Chapter i – an introduction to proof theory. In Samuel R. Buss, editor, *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, pages 1–78. Elsevier, 1998. doi:10.1016/S0049-237X(98)80016-5.

10 Gianluca Curzi and Anupam Das. Cyclic implicit complexity. *CoRR*, abs/2110.01114, 2021. To appear in proceedings of *LICS 2022*. arXiv:2110.01114.

11 Anupam Das. A circular version of Gödel's T and its abstraction complexity. *CoRR*, abs/2012.14421, 2020. arXiv:2012.14421.

12 Anupam Das. On the logical complexity of cyclic arithmetic. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:1)2020.

13 Anupam Das. On the logical strength of confluence and normalisation for cyclic proofs. In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 29:1–29:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSCD.2021.29.

14 Anupam Das and Damien Pous. A cut-free cyclic proof system for Kleene algebra. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 261–277. Springer, 2017.

15 Anupam Das and Damien Pous. Non-Wellfounded Proof Theory For (Kleene+Action) (Algebras+Lattices). In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2018.19.

16 Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ-calculus. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 273–284. Springer, 2006.

17 Abhishek De and Alexis Saurin. Infinets: The parallel syntax for non-wellfounded proof-theory. In Serenella Cerrito and Andrei Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods – 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings*, volume 11714 of *Lecture Notes in Computer Science*, pages 297–316. Springer, 2019. doi:10.1007/978-3-030-29026-9_17.

18 Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In *Computer Science Logic 2013 (CSL 2013)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.

19 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.

20 Emmanuel Hainry, Damiano Mazza, and Romain Péchoux. Polynomial time over the reals with parsimony. In Keisuke Nakano and Konstantinos Sagonas, editors, *Functional and Logic Programming – 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings*, volume 12073 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2020. doi:10.1007/978-3-030-59025-3_4.

21 Martin Hofmann. A mixed modal/linear lambda calculus with applications to Bellantoni-Cook safe recursion. In Mogens Nielsen and Wolfgang Thomas, editors, *Computer Science Logic, 11th International Workshop, CSL '97, Annual Conference of the EACSL, Aarhus, Denmark, August 23-29, 1997, Selected Papers*, volume 1414 of *Lecture Notes in Computer Science*, pages 275–294. Springer, 1997. doi:10.1007/BFb0028020.

22 Stephen Cole Kleene. *Introduction to Metamathematics*. Bubliotheca Mathematica. Wolters-Noordhoff Publishing, 7 edition, 1971.

23 Ker-I Ko. *Complexity Theory of Real Functions*. Birkhauser Boston Inc., USA, 1991.

24 Denis Kuperberg, Laureline Pinault, and Damien Pous. Cyclic proofs, system T, and the power of contraction. *Proc. ACM Program. Lang.*, 5(POPL):1–28, 2021. doi:10.1145/3434282.

25 Ugo Dal Lago and Paolo Parisen Toldin. A higher-order characterization of probabilistic polynomial time. *Inf. Comput.*, 241:114–141, 2015. doi:10.1016/j.ic.2014.10.009.

26 Daniel Leivant. A foundational delineation of computational feasiblity. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 2–11. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151625.

27 Damiano Mazza. Non-uniform polytime computation in the infinitary affine lambda-calculus. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 305–317. Springer, 2014. doi:10.1007/978-3-662-43951-7_26.

28 Damiano Mazza. Simple parsimonious types and logarithmic space. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 24–40. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.24.

29 Damiano Mazza and Kazushige Terui. Parsimonious types and non-uniform computation. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 350–361. Springer, 2015. doi:10.1007/978-3-662-47666-6_28.

30 Grigori E Mints. Finite investigations of transfinite derivations. *Journal of Soviet Mathematics*, 10(4):548–596, 1978.

31 Damian Niwiński and Igor Walukiewicz. Games for the μ-calculus. *Theoretical Computer Science*, 163(1-2):99–116, 1996.

32 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.

Open Higher-Order Logic

Ugo Dal Lago
Department of Computer Science and Engineering, University of Bologna, Italy

Francesco Gavazzo
Department of Computer Science, University of Pisa, Italy

Alexis Ghyselen
Department of Computer Science and Engineering, University of Bologna, Italy

Abstract

We introduce a variation on Barthe et al.'s higher-order logic in which formulas are interpreted as predicates over *open* rather than *closed* objects. This way, concepts which have an intrinsically functional nature, like continuity, differentiability, or monotonicity, can be expressed and reasoned about in a very natural way, following the structure of the underlying program. We give open higher-order logic in distinct flavors, and in particular in its *relational* and *local* versions, the latter being tailored for situations in which properties hold only in *part* of the underlying function's domain of definition.

2012 ACM Subject Classification Theory of computation → Higher order logic; Theory of computation → Logic and verification

Keywords and phrases Formal Verification, Relational Logic, First-Order Properties

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.17

Related Version *Full Version*: `https://arxiv.org/abs/2211.06671`

Funding This work was funded by the ERC CoG "DIAPASoN" GA 818616.

1 Introduction

Reasoning about functional programs poses a whole series of challenges due, in particular, to the presence of higher-order constructions. A class of methodologies particularly suitable for compositional reasoning on such programs is that of *type systems*, which can be seen as lightweight formal methods for the verification of relatively simple properties, mainly safety ones. In the last thirty years, it has become apparent that properties beyond mere safety are actually amenable to be verified through types, e.g, termination [6, 9], complexity bounds [13, 7], and noninterference [18]. In all these cases, types serve as expressions meant to abstractly describe the input and output interfaces to functions. Various levels of abstractions in turn give rise to distinct levels of expressiveness, and to type inference problems of varying degrees of difficulty.

A somehow different, although related, approach is the one of program logics, in which types are replaced or complemented by formulas written in a logical language. This approach, pioneered by Floyd and Hoare in the context of first-order imperative languages [10, 12], is nowadays common also in the realm of higher-order programming languages [4], where it stands out for its expressive power. Indeed, relative completeness results [5], which hold in many contexts within the realm program logics, are rarer in type systems.

A simple, yet powerful, form of program logic among those capable of dealing with higher-order programs is *higher-order logic*, as formulated by Aguirre et al. in a series of recent works [2, 11, 1, 19], most of which focusing on relational reasoning about higher-order programs. One common trait between the many introduced dialects of higher-order logic is the fact that object programs are taken to be terms of a simply-typed λ-calculus, while

properties are expressed in predicate logic. In the words of Aguirre et al. [2], higher-order logic *can be understood as an attempt to internalize the versatility of relational logical relations in a syntactic framework*. Indeed, rules of higher-order logic are built in such a way that syntax-directed reasoning about programs can be done following the rules provided by logical relations [17, 21]: any basic property at ground types is generalized at higher types by stipulating, conceptually, that related functions should map related inputs to related outputs.

As observed in several works [3, 15, 22], however, ordinary logical relations cannot easily deal with properties which are higher-order extensions of *first-order*, rather than ground, properties. This includes continuity, differentiability, or monotonicity properties. Indeed, what would be the base case in the (recursive) definition of the corresponding logical relation? Properties like continuity hold for functions and are meaningless if formulated for, say, real numbers. As we show in Section 2 below, expectedly, similar difficulties show up when dealing with the same kind of properties in higher-order logic. Reasoning is indeed possible, but becomes cumbersome and difficult to be carried out compositionally, i.e. in a syntax-directed way. Going back to logical relations, however, there is a way out, which consists in switching to an *open* version of logical relations in which the base case is indeed the one of first-order types. This way, one is allowed to start from, say, continuity and generalize it to higher-order types naturally.

In this paper, we show that open logical relations can themselves be given a formal counterpart in the realm of higher-order logic. We do so by introducing a variation on Aguirre et al.'s higher-order logic, called *open higher-order logic*. The salient feature of this new system, compared to those from the literature, is the fact that the mathematical objects that proofs implicitly deal with, namely higher-order functions, are assumed to in turn depend on a sequence of ground global parameter $\Theta = \boldsymbol{x}_1 : \mathtt{B}_1, \ldots, \boldsymbol{x}_n : \mathtt{B}_n$, hence the attribute "open". This way, a predicate P about objects of type τ is taken as a subset of $[\![\tau]\!]^{[\![\Theta]\!]}$, rather than just $[\![\tau]\!]$.

Technically, the contributions of this paper are threefold:

- We define four concrete logical systems, all built around the aforementioned ideas, and capable of dealing with formulas, programs, relations between programs (see Section 4 for those three systems), and local reasoning (Section 5), respectively.
- For each of the presented systems, we give an interpretation of formulas and sequents into an underlying semantic universe.
- We provide a series of examples, dealing with properties like continuity and differentiability, showing how they can be handled in the logical systems. These are spread out in Section 4 and Section 5.

2 Reasoning about First-Order Properties, Compositionally

Before moving to the technical development of open higher-order logic, we informally describe the kind of problems such a logic is meant to solve. We do so by means of an example that higher-order logic is *in principle* capable of dealing with, although doing that compositionally is highly nontrivial: continuity in the presence of higher-order functions.

Let us consider a simply typed λ-calculus extended with a base type `Real` for real numbers, as well as with constants and symbols for functions of first-order type, not necessarily interpreted as continuous functions. It is clear that a term t of type $\mathtt{Real} \to \mathtt{Real}$, in a situation like the one just mentioned, computes a function whose continuity properties depend on how it is constructed and on which constants occur within it. In UHOL [11], the unary variant of higher order logic (HOL), the fact that such a term t actually computes

a continuous function can be expressed as the formula $\mathcal{C}(t)$, where $\mathcal{C}$ is a unary predicate on terms of type $\texttt{Real} \to \texttt{Real}$. Clearly, this predicate's properties, at least some of them, should be captured by way of logical formulas, which become axioms in the underlying formal system. We would thus have, e.g., an axiom about stability of continuity by composition, namely the following formula:

$$\forall p, q : \texttt{Real} \to \texttt{Real}, (\mathcal{C}(p) \wedge \mathcal{C}(q)) \Rightarrow \mathcal{C}(\lambda x.q\ (p\ x))$$

Now, let t be the term $\lambda x.h\ (g\ (f\ x))$, where f, g, h are all constants of type $\texttt{Real} \to \texttt{Real}$ interpreted as continuous functions. We would like to prove within UHOL, possibly in a compositional way, that t – itself a term of type $\texttt{Real} \to \texttt{Real}$ – is continuous too. By design, UHOL's rules – and thus UHOL's proofs – follow the term structure target formulas refer to; and in the case of λ-abstractions, the corresponding rule can be read as follows: "if, whenever the argument x satisfies a precondition ϕ, the body u satisfies the postcondition ψ, then $\lambda x.u$ satisfies the formula $\forall x.\phi \Rightarrow \psi$." When it comes to $\mathcal{C}(t)$, this means we need a postcondition satisfied by the term $h\ (g\ (f\ x))$ *for every* x. But continuity on x cannot be defined looking only at (the semantics of) $h\ (g\ (f\ x))$, namely at a single real number, and thus it is not clear how one should proceed.

Open higher-order logic (OHOL), instead, considers open objects as first-class citizens, without altering the rest of the framework in any other way, so still decomposing terms in the style of logical relations. In other words, in open higher-order logic we can reason on the open term $h\ (g\ (f\ x))$ seeing it *as a function* of type $\texttt{Real} \to \texttt{Real}$. Formally, in this case, we consider open terms for the context $\Theta = x : \texttt{Real}$. The proof can then proceed as follows:

$$\frac{\Gamma \mid \Psi \vdash^{\Theta} f : \texttt{Real} \to \texttt{Real}\ \{\forall y : \texttt{Real}, \mathcal{C}(y) \Rightarrow \mathcal{C}(\boldsymbol{r}\ y)\} \qquad \Gamma \mid \Psi \vdash^{\Theta} (g\ (h\ x)) : \texttt{Real}\ \{\mathcal{C}(\boldsymbol{r})\}}{\Gamma \mid \Psi \vdash^{\Theta} f\ (g\ (h\ x)) : \texttt{Real}\ \{\mathcal{C}(\boldsymbol{r})\}}$$

with the following tree for the second premise.

$$\frac{\Gamma \mid \Psi \vdash^{\Theta} g : \texttt{Real} \to \texttt{Real}\ \{\forall y : \texttt{Real}, \mathcal{C}(y) \Rightarrow \mathcal{C}(\boldsymbol{r}\ y)\} \qquad \Gamma \mid \Psi \vdash^{\Theta} (h\ x) : \texttt{Real}\ \{\mathcal{C}(\boldsymbol{r})\}}{\Gamma \mid \Psi \vdash^{\Theta} (g\ (h\ x)) : \texttt{Real}\ \{\mathcal{C}(\boldsymbol{r})\}}$$

The application rule that we use for this proof is basically an elimination of the implication constructor for the target formula. The derivation for the functions f and g would then be obtained using an axiom assessing that continuity composes, whereas the continuity of $h\ x$ would then follow by hypothesis, as we assumed that all three functions were continuous and thus, in particular, we have $\mathcal{C}(h\ x)$.

The above example relies on continuity to show a limitation of (U)HOL. Such a limitation, however, is not specific to continuity, and it virtually affects any first order property. In this paper, we shall study another interesting example involving a first-order property, this time at a relational level: correctness of a state-of-the-art algorithm for automatic differentiation. In fact, an algorithm for forward mode differentiation of simply-typed terms has been recently proved correct by way of open logical relations [3]. Formalizing the aforementioned correctness proof in higher-order logic, however, would pose problems of exactly the same kind as those we described above, since derivability and (automatic) differentiation only make sense when spelled out on functions. In Section 4.2 , we show how to solve this problem by a relational version of OHOL capable of dealing both with (open) terms t *and* with their derivatives $D(t)$.

$$t, u ::= x \mid \underline{c} \mid \underline{f} \mid \lambda y.t \mid t\ u \mid \langle t, u\rangle \mid \pi_1(t) \mid \pi_2(t) \qquad \tau, \sigma ::= \mathtt{B} \mid \tau \times \sigma \mid \tau \to \sigma$$

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{c : \mathtt{B} \in \mathcal{C}}{\Gamma \vdash \underline{c} : \mathtt{B}} \qquad \frac{f : \tilde{\mathtt{B}} \to \mathtt{B} \in \mathcal{F}}{\Gamma \vdash \underline{f} : \tilde{\mathtt{B}} \to \mathtt{B}} \qquad \frac{\Gamma, x : \tau \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \tau \to \sigma}$$

$$\frac{\Gamma \vdash t : \tau \to \sigma \qquad \Gamma \vdash u : \tau}{\Gamma \vdash t\ u : \sigma} \qquad \frac{\Gamma \vdash t : \tau_1 \qquad \Gamma \vdash u : \tau_2}{\Gamma \vdash \langle t, u\rangle : \tau_1 \times \tau_2} \qquad \frac{\Gamma \vdash t : \tau_1 \times \tau_2}{\Gamma \vdash \pi_i(t) : \tau_i}$$

Figure 1 Syntax and Static Semantics for $\Lambda_{\mathcal{C},\mathcal{F}}$.

Suppose given for each base type $\mathtt{B}$ an interpretation $[\![\mathtt{B}]\!]$ (for example, the type for reals could be given as an interpretation the actual set of real numbers). Suppose also that for each constant $c : \mathtt{B} \in \mathcal{C}$, we have an interpretation $[\![c]\!] \in [\![\mathtt{B}]\!]$ and for each function $f : \tilde{\mathtt{B}} \to \mathtt{B} \in \mathcal{F}$, we have an interpretation $[\![f]\!] : [\![\mathtt{B}]\!]^{[\![\tilde{\mathtt{B}}]\!]}$.

Then, the interpretation of types is defined by an object in the category of **Set**:

$$\tau_1 \times \tau_2 = [\![\tau_1]\!] \times [\![\tau_2]\!] \qquad \tau_1 \to \tau_2 = [\![\tau_2]\!]^{[\![\tau_1]\!]}$$

And, a typing derivation $\Gamma \vdash t : \tau$ is interpreted as a map $[\![\Gamma \vdash t : \tau]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$, this is defined by induction on the type derivation with:

$$[\![\Gamma, x : \tau \vdash x : \tau]\!](\tilde{y}, y) = y \qquad [\![\Gamma \vdash \underline{c} : \mathtt{B}]\!](\tilde{y}) = [\![c]\!] \qquad [\![\Gamma \vdash \underline{f} : \tilde{\mathtt{B}} \to \mathtt{B}]\!](\tilde{y}) = [\![f]\!]$$

$$[\![\Gamma \vdash \lambda x.t : \tau_1 \to \tau_2]\!](\tilde{y}) = \mathtt{fun}\ (y : [\![\tau_1]\!]) \mapsto [\![\Gamma, x : \tau_1 \vdash t : \tau_2]\!](\tilde{y}, y)$$

$$[\![\Gamma \vdash t_1\ t_2 : \tau_2]\!](\tilde{y}) = ([\![\Gamma \vdash t_1 : \tau_1 \to \tau_2]\!](\tilde{y}))([\![\Gamma \vdash t_2 : \tau_1]\!](\tilde{y}))$$

$$[\![\Gamma \vdash \langle t_1, t_2\rangle : \tau_1 \times \tau_2]\!](\tilde{y}) = ([\![\Gamma \vdash t_1 : \tau_1]\!](\tilde{y}), [\![\Gamma \vdash t_2 : \tau_2]\!](\tilde{y}))$$

$$[\![\Gamma \vdash \pi_i(t) : \tau_i]\!](\tilde{y}) = \pi_i([\![\Gamma \vdash t : \tau_1 \times \tau_2]\!](\tilde{y}))$$

Figure 2 Denotational Semantics for $\Lambda_{\mathcal{C},\mathcal{F}}$.

3 Preliminaries

The (higher-order) logic we deal with in this work is, in its bare essence, a formal framework to prove properties about higher-order programs. Consequently, a precise definition of such a logic requires a formal definition of what a program is. Here, we consider a λ-calculus with base types, constants, and functions [3]. We write $\mathcal{C}$ and $\mathcal{F}$ for the collections of constants and symbols for first-order functions upon which terms are defined. The syntax and statics of the resulting calculus, that we denote by $\Lambda_{\mathcal{C},\mathcal{F}}$, are given in Figure 1. The metavariable $\mathtt{B}$ ranges over *base types* (we assume a fixed collection of base types as given), such as real numbers or booleans. Moreover, we assume each constant $c \in \mathcal{C}$ to inhabit a base type $\mathtt{B}$ (notation $c : \mathtt{B} \in \mathcal{C}$) and each function $f \in \mathcal{F}$ to inhabit a function type $\mathtt{B}_1 \times \cdots \times \mathtt{B}_m \to \mathtt{B}$ (notation $f : \mathtt{B}_1 \times \cdots \times \mathtt{B}_m \to \mathtt{B} \in \mathcal{F}$). We oftentimes employ the notation $\tilde{\mathtt{B}}$ to denote a tuple $\mathtt{B}_1 \times \cdots \times \mathtt{B}_m$ when m is clear from the context. This notation generalizes to other objects, e.g. terms, types, and typing contexts.

Finally, we endow $\Lambda_{\mathcal{C},\mathcal{F}}$ with a standard set-theoretic denotational semantics in the usual way. Accordingly, each type τ is interpreted as a set $[\![\tau]\!]$, and any derivable typing judgment $\Gamma \vdash t : \tau$ is interpreted as a function $[\![\Gamma \vdash t : \tau]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$, where $[\![\Gamma]\!] = \prod_{y:\sigma \in \Gamma} [\![\sigma]\!]$.

$$\frac{}{\Gamma \mid \Psi, \phi \vdash \phi} \qquad \frac{\Gamma \vdash t_i : \tau \qquad t_1 =_{(\to)} t_2}{\Gamma \mid \Psi \vdash (t_1 = t_2)} \qquad \frac{\Gamma \mid \Psi \vdash \phi[t_1/y] \qquad \Gamma \mid \Psi \vdash (t_1 = t_2)}{\Gamma \mid \Psi \vdash \phi[t_2/y]}$$

$$\frac{\Gamma, y : \tau \mid \Psi \vdash \phi}{\Gamma \mid \Psi \vdash \forall y : \tau.\phi} \qquad \frac{\Gamma \mid \Psi \vdash \forall y : \tau.\phi \qquad \Gamma \vdash t : \tau}{\Gamma \mid \Psi \vdash \phi[t/y]}$$

Figure 3 Selected Rules of HOL.

3.1 Higher-Order Logic

Having defined $\Lambda_{\mathcal{C},\mathcal{F}}$, we now recall basic definitions of *Higher-Order Logic* [11], the logic we will build upon.

▶ **Definition 1.** *The syntax of HOL formulas is given by the following grammar:*

$$\phi ::= P(t_1, \dots, t_m) \mid \top \mid \bot \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \Rightarrow \phi \mid \forall y : \tau.\phi \mid \exists y : \tau.\phi.$$

HOL formulas are defined starting from a given collection of atomic predicates on $\Lambda_{\mathcal{C},\mathcal{F}}$ terms, from which more complex formulas are then constructed relying on connectives. Each predicate P comes with an *arity* $(\tau_1, \dots, \tau_m)$ stating that in an atomic formula $P(t_1, \dots, t_m)$ each term t_i must have type τ_i. Moreover, notice that variables introduced by quantifiers are (typed) term variables, meaning that they can occur in terms themselves occurring in atomic formulas. This intuitive description is formalized by means of well-typing judgments of the form $\Gamma \vdash \phi$ whose defining rules are as expected. Due to space constraints, we omit the formal definition of such rules and address the reader to one of the many papers on the subject [2, 11]. As it is customary, we also assume an equality predicate to be available for *all* types.

HOL inherits the set-theoretical semantics of $\Lambda_{\mathcal{C},\mathcal{F}}$: given an interpretation $[\![P]\!] \subseteq [\![\tau_1]\!] \times \cdots \times [\![\tau_m]\!]$ for each predicate P of arity $(\tau_1, \dots, \tau_m)$, well-typed assertions $\Gamma \vdash \phi$ are interpreted as subsets $[\![\Gamma \vdash \phi]\!] \subseteq [\![\Gamma]\!]$. Intuitively, this semantics is defined for non-atomic formulas by the standard interpretation of boolean constructors and quantification, whereas we define the interpretation $[\![P(t_1, \dots, t_m)]\!]$ of an atomic formula $P(t_1, \dots, t_m)$ as the set of elements $\gamma \in [\![\Gamma]\!]$ such that $([\![\Gamma \vdash t_i : \tau_i]\!](\gamma))_{1 \leq i \leq m} \in [\![P]\!]$.

The real power of HOL is not its set-theoretic semantics, but its proof theory. In fact, HOL comes with a proof system that we recall here in a sequent calculus-style. Such a system employs judgments of the form $\Gamma \mid \Psi \vdash \phi$, where Ψ is a set of well-typed assertions, and ϕ is a well-typed assertion. A valid judgment $\Gamma \mid \Psi \vdash \phi$ attests that in the typing context Γ and under the assumptions in Ψ, ϕ is true. Accordingly, predicates P must come with axioms defining their (logical) meaning, as there is no rule for arbitrary predicates. We recall some of the most important rules in Figure 3, other rules being the usual inference rules for logical constructors (we write $=_{(\to)}$ for the smallest equivalence relation including $\to$, the reduction relation on $\Lambda_{\mathcal{C},\mathcal{F}}$, which can be defined as expected). Note that in addition to the logical rules, there are rules specific to the equality predicate allowing, in particular, term substitutions.

3.2 Unary HOL

In practical examples, especially in presence of unary predicates, (the proof system of) HOL may be difficult to use, its rules being ultimately formula-directed. In those cases, it is desirable to have a system (whose rules are) directed by the structure of the terms involved.

$$\frac{\Gamma, y : \tau \mid \Psi \vdash \phi[y/\boldsymbol{r}]}{\Gamma, y : \tau \mid \Psi \vdash y : \tau \ \{\phi\}} \qquad \frac{\Gamma \mid \Psi \vdash \phi[\underline{c}/\boldsymbol{r}]}{\Gamma \mid \Psi \vdash \underline{c} : \mathtt{B} \ \{\phi\}}$$

$$\frac{\Gamma \mid \Psi \vdash \phi[\underline{f}/\boldsymbol{r}]}{\Gamma \mid \Psi \vdash \underline{f} : \tilde{\mathtt{B}} \to \mathtt{B} \ \{\phi\}} \qquad \frac{\Gamma, y : \tau \mid \Psi, \psi \vdash t : \sigma \ \{\phi\}}{\Gamma \mid \Psi \vdash \lambda y.t : \tau \to \sigma \ \{\forall y.\psi \Rightarrow \phi[\boldsymbol{r}\ y/\boldsymbol{r}]\}}$$

$$\frac{\Gamma \mid \Psi \vdash t : \tau \to \sigma \ \{\forall y.\psi[y/\boldsymbol{r}] \Rightarrow \phi[\boldsymbol{r}\ y/\boldsymbol{r}]\} \qquad \Gamma \mid \Psi \vdash u : \tau \ \{\psi\}}{\Gamma \mid \Psi \vdash t\ u : \sigma \ \{\phi[u/y]\}}$$

$$\frac{\Gamma \mid \Psi \vdash t_i : \tau_i \ \{\phi_i\} \qquad \Gamma \mid \Psi \vdash \forall y, z.\phi_1[y/\boldsymbol{r}] \wedge \phi_2[z/\boldsymbol{r}] \Rightarrow \phi[\langle y, z\rangle/\boldsymbol{r}]}{\Gamma \mid \Psi \vdash \langle t_1, t_2\rangle : \tau_1 \times \tau_2 \ \{\phi\}}$$

$$\frac{\Gamma \mid \Psi \vdash t : \tau_1 \times \tau_2 \ \{\phi[\pi_i(\boldsymbol{r})/\boldsymbol{r}]\}}{\Gamma \mid \Psi \vdash \pi_i(t) : \tau_i \ \{\phi\}} \qquad \frac{\Gamma \mid \Psi \vdash t : \tau \ \{\psi\} \qquad \Gamma \mid \Psi \vdash \psi[t/\boldsymbol{r}] \Rightarrow \phi[t/\boldsymbol{r}]}{\Gamma \mid \Psi \vdash t : \tau \ \{\phi\}}$$

Figure 4 Rules of UHOL.

Moving from this observation, Aguirre et al. [11, 2] have developed term-directed proof systems for relational higher-order logics. We recall how such systems work, focusing on the unary case only.

▶ **Definition 2.** Unary *higher-order logic (UHOL) is an inference system based on judgments of the form* $\Gamma \mid \Psi \vdash t : \tau \ \{\phi\}$*, where* Γ *and* t *are as usual,* Ψ *is a set of HOL formulas, and* ϕ *is a HOL formula with a distinguished free variable* $\boldsymbol{r}$ *not appearing in* Γ *(i.e.* $\boldsymbol{r}$ *acts as a placeholder for* t *in the* target formula ϕ*). The defining rules of UHOL are given in Figure 4.*

A UHOL judgment $\Gamma \mid \Psi \vdash t : \tau \ \{\phi\}$ attests that t has type τ in the typing context Γ, and that the formula $\phi[t/\boldsymbol{r}]$ is true under the assumptions in Ψ. To prove such judgments, we rely on term-directed rules. In particular, the first three rules in Figure 4 state that for base terms, satisfiability of the target formula must be verified in the HOL judgment system. Other rules reshape the target formula according to the structure of t. For example, in the λ-abstraction rule the target formula expresses that if an argument satisfies a precondition ψ, then the application satisfies a postcondition ϕ. This can be verified assuming ψ and trying to prove the target formula ϕ for the body of the λ-abstraction. Finally, the last rule, that does not depend on the structure of t, allows us to use HOL judgments to weaken target formulas.

Aguirre et al.'s results [11, 2] show that HOL and UHOL are equivalent – i.e. $\Gamma \mid \Psi \vdash t : \sigma \ \{\phi\}$ if and only if $\Gamma \mid \Psi \vdash \phi[t/\boldsymbol{r}]$ – and that they are both sound in regard to the previously introduced set-theoretic semantics: if $\Gamma \mid \Psi \vdash \phi$ is valid, then $[\![\Gamma \vdash (\bigwedge_{\psi \in \Psi} \psi)]\!] \subseteq [\![\Gamma \vdash \phi]\!]$. But even if equivalent from a semantic and provability perspective, they are not so if one looks at proof effectiveness and automation.

In the HOL system, it is difficult to know when to use a term substitution to simplify a term, as well as when to apply the cut rule. Consequently, the system turns out to work well on judgments speaking about simple enough terms, but it may be difficult to use when more complex terms are involved. In UHOL, instead, the last rule of Figure 4 – which is crucial to guarantee expressiveness of the logic – turns out to be problematic for automation, and it is thus desirable to avoid its usage as much as possible. This makes the proof system of UHOL effective when applied to formulas with possibly complex terms, but *only* if there is little or no need to adjust the target formula, i.e. to use the last rule in Figure 4. Additionally, UHOL relies on HOL to prove properties of base terms, and thus it inherits the same weaknesses of the latter. Finally, the system is designed to work on terms and formulas whose shape

matches the semantic behavior of the terms they refer to. In absence of such a correspondence, proofs become remarkably difficult. This is indeed problematic, as there are many natural examples lacking such a correspondence and for which, consequently, UHOL is ineffective. A typical pattern of such a behavior is, for instance, the one we described in Section 2: when a formula for a λ-abstraction focuses on the whole function – and not on the actual application of this function to an argument – the rule for λ-abstraction cannot be used.

The just described scenario, namely the one in which one aims to prove *first-order* properties of programs, highlights an important weakness of both HOL and UHOL: none of the them can easily – not to say compositionally – cope with first-order properties of programs. The main contribution of the present work, namely the definition of Open HOL, provides a way to go beyond the aforementioned weakness, this way achieving effective and compositional reasoning about first-order program properties.

4 Open Higher-Order Logic

In this section, we introduce *Open* HOL (OHOL, for short), a higher-order logic designed to natively deal with *first-order properties* of programs. To do so, we follow the idea behind open logical relations [3] where, in the base case of the (recursive) definition of a logical relation, *closed* terms of a ground type $\mathtt{B}$ – which, semantically, correspond to elements in $[\![\mathtt{B}]\!]$ – are replaced by *open* terms having free variables in a fixed typing context Θ – so that the semantics of such terms is given by *functions* from $[\![\Theta]\!]$ to $[\![\mathtt{B}]\!]$, which are then required to satisfy the first-order property of interest.

Let us fix a typing context $\Theta = \boldsymbol{x}_1 : \mathtt{B}_1, \dots, \boldsymbol{x}_n : \mathtt{B}_n$. For the sake of clarity, we consider variables $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ as disjoint from the usual term variables, which we denote by $y, y_1, \dots$. OHOL's formulas are defined parametrically with respect to Θ (meaning that each typing context Θ gives a OHOL's formulas' grammar) by the following grammar:

$$\phi ::= P^{\Theta}(t_1, \dots, t_m) \mid \top \mid \bot \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \Rightarrow \phi \mid \forall y : \tau.\phi \mid \exists y : \tau.\phi.$$

Notice that there is only one difference between HOL's and OHOL's formulas, namely atomic predicates. In OHOL, in fact, atomic predicates P^{Θ} are parametrized by the typing context Θ, with the intended meaning that arguments of P^{Θ} may have free variables in Θ. That is, if a predicate P^{Θ} has arity $(\tau_1, \dots, \tau_n)$, then in an atomic formula $P^{\Theta}(t_1, \dots, t_m)$ we require each term t_i to have type τ_i in the typing context Θ. Formally, this implies that to derive well-typed OHOL's judgments $\Gamma \vdash^{\Theta} \phi$, we shall rely on the following rule for atomic predicates:

$$\frac{P^{\Theta} \text{ has arity } (\tau_1, \dots, \tau_m) \qquad \forall 1 \leq i \leq m,\ \Gamma, \Theta \vdash t_i : \tau_i}{\Gamma \vdash^{\Theta} P^{\Theta}(t_1, \dots, t_m)}$$

Let us now see how the admittedly minor differences between the defining grammars of HOL and OHOL formulas (and, consequently, in their judgment rules) impact on the logics themselves. We begin with set-theoretic semantics, where we see that OHOL indeed interprets a term t of type τ with free variables in Θ as a function from $[\![\Theta]\!]$ to $[\![\tau]\!]$. More generally, we interpret well-typed assertions $\Gamma \vdash^{\Theta} \phi$ as subsets $[\![\Gamma \vdash^{\Theta} \phi]\!] \subseteq [\![\Gamma^{\Theta}]\!]$ with $[\![\Gamma^{\Theta}]\!] \cong \prod_{y:\tau \in \Gamma} [\![\tau]\!]^{[\![\Theta]\!]}$. For readability, it is convenient to introduce the following notation: given $f \in B^{A \times C}$ and $g \in A^C$, we define $f \star g \in B^C$ by $(f \star g)(x) = f(g(x), x)$.

▶ **Definition 3.** *Given an interpretation* $[\![P^{\Theta}]\!] \subseteq \prod_{1 \leq i \leq m} [\![\tau_i]\!]^{[\![\Theta]\!]}$ *of predicates* P^{Θ} *of arity* $(\tau_1, \dots, \tau_m)$*, the semantics* $[\![\Gamma \vdash^{\Theta} P^{\Theta}(t_1, \dots, t_m)]\!] \subseteq [\![\Gamma]\!]^{[\![\Theta]\!]}$ *of the judgment* $\Gamma \vdash^{\Theta} P^{\Theta}(t_1, \dots, t_m)$ *is defined as follows:*

$$[\![\Gamma \vdash^{\Theta} P^{\Theta}(t_1, \ldots, t_m)]\!] = \{\tilde{y} \in [\![\Gamma^{\Theta}]\!] \mid \prod_{1 \leq i \leq m} ([\![\Gamma, \Theta \vdash t_i : \tau_i]\!] \star \tilde{y}) \in [\![P^{\Theta}]\!]\}.$$

We define the semantics $[\![\Gamma \vdash^{\Theta} \phi]\!] \subseteq [\![\Gamma]\!]^{[\![\Theta]\!]}$ *of a well-typed judgment* $\Gamma \vdash^{\Theta} \phi$ *by inductively extending* $[\![\Gamma \vdash^{\Theta} P^{\Theta}(t_1, \ldots, t_m)]\!]$ *in the usual way. For instance:*

$$[\![\Gamma \vdash^{\Theta} \phi \vee \psi]\!] = [\![\Gamma \vdash^{\Theta} \phi]\!] \cup [\![\Gamma \vdash^{\Theta} \psi]\!];$$
$$[\![\Gamma \vdash^{\Theta} \forall y : \tau.\phi]\!] = \{\tilde{y} \in [\![\Gamma^{\Theta}]\!] \mid \forall y \in [\![\tau]\!]^{[\![\Theta]\!]}, (\tilde{y}, y) \in [\![\Gamma, y : \tau \vdash^{\Theta} \phi]\!]\}.$$

Notice that even if OHOL's formulas are essentially those of HOL, Definition 3 interprets terms and variables of type τ as function from $[\![\Theta]\!]$ to $[\![\tau]\!]$. In particular, notice that in the formula $\forall y : \tau.\phi$ the variable y has type τ in the underlying context, but it is interpreted as a function in $[\![\tau]\!]^{[\![\Theta]\!]}$.

Having defined the set-theoretic semantics of OHOL, we move to its proof system. We consider judgments of the form $\Gamma \mid \Psi \vdash^{\Theta} \phi$ and adjust the judgment system of HOL (Figure 3) by adding the typing context Θ to all premises of a type derivation of $\Lambda_{\mathcal{C},\mathcal{F}}$ terms. For example the rule

$$\frac{\Gamma \mid \Psi \vdash \forall y : \tau.\phi \quad \Gamma \vdash t : \tau}{\Gamma \mid \Psi \vdash \phi[t/y]} \quad \text{is replaced by} \quad \frac{\Gamma \mid \Psi \vdash^{\Theta} \forall y : \tau.\phi \quad \Gamma, \Theta \vdash t : \tau}{\Gamma \mid \Psi \vdash^{\Theta} \phi[t/y]}$$

▶ **Proposition 4.** *OHOL is sound with respect to the set-theoretic semantics. That is, if* $\Gamma \mid \Psi \vdash^{\Theta} \phi$ *is derivable, then* $[\![\Gamma \vdash^{\Theta} \bigwedge_{\psi \in \Psi} \psi]\!] \subseteq [\![\Gamma \vdash^{\Theta} \phi]\!]$.

Proof. By induction on $\Gamma \mid \Psi \vdash^{\Theta} \phi$ proceeding as in the corresponding proof for HOL. ◀

4.1 Unary OHOL

The OHOL proof system, being ultimately defined upon the one for HOL, shares (some) strengths and weaknesses with the latter. In particular, as for HOL, the formula-directed nature of OHOL rules makes OHOL difficult to use in presence of complex terms. To overcome this problem, we proceed as in the design of UHOL and introduce a new judgment system, which we dub *Unary Open HOL* (UOHOL, for short), whose proof rules are term-directed. Formally, UOHOL has judgments of the form $\Gamma \mid \Psi \vdash^{\Theta} t : \tau \ \{\phi\}$, where Ψ is a set of OHOL formulas, ϕ is a OHOL formula with a distinguished variable $\boldsymbol{r}$, and t is a $\Lambda_{\mathcal{C},\mathcal{F}}$ term. A judgment $\Gamma \mid \Psi \vdash^{\Theta} t : \tau \ \{\phi\}$ states that $\Gamma, \Theta \vdash t : \tau$ and that, under the assumptions in Ψ, $\phi[t/\boldsymbol{r}]$ is satisfied. UOHOL rules consist of the previously mentioned rules of Figure 4, with one additional rule:

$$\frac{(\boldsymbol{x}_i : \mathtt{B}_i) \in \Theta \qquad \Gamma \mid \Psi \vdash^{\Theta} \phi[\boldsymbol{x}_i/\boldsymbol{r}]}{\Gamma \mid \Psi \vdash^{\Theta} \boldsymbol{x}_i : \mathtt{B}_i \ \{\phi\}}$$

Such a rule corresponds to the axiom rule in Figure 4, but for variables in Θ. Finally, following the case of HOL and UHOL, we prove equivalence of OHOL and UOHOL.

▶ **Proposition 5.** *UOHOL and OHOL are equivalent, i.e.* $\Gamma \mid \Psi \vdash^{\Theta} t : \tau \ \{\phi\}$ *if and only if* $\Gamma \mid \Psi \vdash^{\Theta} \phi[t/\boldsymbol{r}]$.

Compared to UHOL, we see that UOHOL allows for more satisfactory proofs of higher-order properties, as mentioned in Section 2. To see that, let ϕ be the following OHOL formula: $\forall y : \tau \to \sigma, \forall z : \tau, P^{\Theta}(y\ z)$. It is easy to see that ϕ could be written as a HOL formula too: for, let us consider the formula ψ below, with T being product of types in Θ:

$$\forall y : T \to (\tau \to \sigma), \forall z : (T \to \tau), P(\lambda x.(y\ x)\ (z\ x)).$$

Compared to ψ, the formula ϕ is considerably simpler, and thus it facilitates derivation in OHOL or UOHOL. Moreover, it is precisely the simpler (logical) structure of ϕ that allows us to deal with first-order properties smoothly, in contrast to the previously mentioned UHOL's weaknesses. This is the main point of OHOL and UOHOL. We do not claim any increase of expressiveness over the standard HOL. What we claim, instead, is that OHOL and UOHOL constitute a way to simplify the process of proving some formulas of HOL that cannot be readily proved valid in HOL or UHOL.

4.2 OHOL with Multiple Domains and ROHOL

The discussion at the end of Section 3 on the drawbacks and benefits of HOL and UHOL remains relevant for OHOL and UOHOL. Moreover, even if OHOL solves some problems suffered by HOL when applied to first-order properties, it does *not* solve *all* of them. In particular, OHOL focuses on a single domain – the typing context Θ – that cannot be changed in a formula. Consequently, all terms in a given formula are relative to the *same* domain Θ. As we are going to see, however, it is sometimes useful to allow different terms to be associated to different typing contexts within the same formula. Here, we present a simple generalization of OHOL designed to tackle this problem, showing also its usefulness for the relational version of UOHOL, that we briefly sketch below.

4.2.1 Handling Multiples Domains

Let us begin with OHOL with multiple domains of definition. Due to space constraints, the formal description of the logic has to be omitted (but see [8] for details). Nonetheless, we can still outline its main features here, although at an informal level only. In OHOL with multiple domains of definition, we associate to each variable and term a unique typing context, but contrary to OHOL, we allow different terms to be associated with *distinct* typing contexts. In particular, we assign to variables not only types but also typing contexts, which now play the role of kinds. Consequently, we have typed variables $y : \tau :: \Theta$ stating that y has type τ, and τ has kind Θ. Semantically, we let kinds specify the domains of the semantic interpretation of variables: given typing contexts Θ_1 and Θ_2, the variables $y_1 : \tau_1 :: \Theta_1$ and $y_2 : \tau_2 :: \Theta_2$ are interpreted as functions in $[\![\tau_1]\!]^{[\![\Theta_1]\!]}$ and $[\![\tau_2]\!]^{[\![\Theta_2]\!]}$, respectively (recall that in OHOL each variable $y : \tau$ is interpreted as a function $[\![\tau]\!]^{[\![\Theta]\!]}$, with Θ *fixed*).

The semantic interpretation for formulas using kinds is a rather straightforward generalization of the semantics in Definition 3: we only need to modify the notion of arity for predicates, which are now of the form $(\tau_1 :: \Theta_1, \tau_2 :: \Theta_2, \dots, \tau_m :: \Theta_m)$, with the intended meaning that type τ_i has kind Θ_i. Therefore, a predicate of arity $(\tau_1 :: \Theta_1, \tau_2 :: \Theta_2, \dots, \tau_m :: \Theta_m)$ is now interpreted as a subset of $\prod_{1 \leq i \leq m} [\![\tau_i]\!]^{[\![\Theta_i]\!]}$.

As for the logical rules, the only relevant point is to specify a kind for each term and for each variable, and to impose that all variables in a term have the same kind. For example, we have the following rule for the introduction and elimination of the $\forall$ constructor:

$$\frac{\Gamma, y : \tau :: \Theta \mid \Psi \vdash \phi}{\Gamma \mid \Psi \vdash \forall y : \tau :: \Theta.\phi} \qquad \frac{\Gamma \mid \Psi \vdash \forall y : \tau :: \Theta.\phi \qquad \Delta \vdash t : \tau \qquad \forall (z : \sigma) \in \Delta, (z : \sigma :: \Theta) \in \Gamma}{\Gamma \mid \Psi \vdash \phi[t/y]}$$

4.2.2 Going Relationally

Even if theoretically fine, having access to different kinds is not that useful for UOHOL. Indeed, in a UOHOL proof, the focus is on a *single* term (as well as its subterms) only, so that if a term has kind Θ, then we do not gain much from using OHOL with multiple

domains of definition. The potential of such a logic, instead, becomes evident when we move to *relational* reasoning. Aguirre et al. [2, 11] have introduced RHOL, a relational version of UHOL talking about *pairs* of terms, rather than a single term in isolation. Because of the similarities between UHOL and UOHOL, the design of ROHOL from UOHOL can be done by mimicking the original construction by Aguirre et al. A RHOL judgment has the shape $\Gamma \mid \Psi \vdash t_1 : \tau_1 \sim t_2 : \tau_2 \ \{\phi\}$, where ϕ contains two special variables $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ acting as placeholders for t_1 and t_2, respectively. The intuitive meaning is that t_1 and t_2 have types τ_1 and t_2, respectively, in the typing context Γ, and that $\phi[t_1/\boldsymbol{r}_1][t_2/\boldsymbol{r}_2]$ holds under the assumptions in Ψ.

As an example, a RHOL rule for λ-abstraction is

$$\frac{\Gamma, y_1 : \sigma_1, y_2 : \sigma_2 \mid \Psi, \psi \vdash t_1 : \tau_1 \sim t_2 : \tau_2 \ \{\phi\}}{\Gamma \mid \Psi \vdash \lambda y_1.t_1 : \sigma_1 \to \tau_1 \sim \lambda y_2.t_2 : \sigma_2 \to \tau_2 \ \{\forall y_1, y_2.\psi \Rightarrow \phi[\boldsymbol{r}_1 \ y_1/\boldsymbol{r}_1][\boldsymbol{r}_2 \ y_2/\boldsymbol{r}_2]\}}$$

Such a rule can be seen as two instances of the rule for λ-abstractions from UHOL merged together. In a similar fashion, we can extend such a rule to our open setting with a fixed context Θ. More interesting is the case of multiple domains of definition, where we can consider the first term having kind Θ_1 and the second one having kind Θ_2. This way, we obtain the following rule in what we may call ROHOL with multiple domains of definition:

$$\frac{\Gamma, y_1 : \sigma_1 :: \Theta_1, y_2 : \sigma_2 :: \Theta_2 \mid \Psi, \psi \vdash t_1 : \tau_1 \sim t_2 : \tau_2 \ \{\phi\}}{\Gamma \mid \Psi \vdash \lambda y_1.t_1 : \sigma_1 \to \tau_1 \sim \lambda y_2.t_2 : \sigma_2 \to \tau_2 \ \{\forall y_1 : \sigma_1 :: \Theta_1, y_2 : \sigma_2 :: \Theta_2.\psi \Rightarrow \phi[\boldsymbol{r}_1 \ y_1/\boldsymbol{r}_1][\boldsymbol{r}_2 \ y_2/\boldsymbol{r}_2]\}}$$

Such a rule also allows us to see the benefits of having different kinds, as we are now allowed to relate functions with different initial domains. As before, space constraints force us to omit formal details (for which we refer to the long version of this paper [8]). Instead, we witness how our logic work by means of a nontrivial example.

▶ **Example 6** (Automatic Differentiation)**.** Let us consider the problem of formally proving correctness of forward mode automatic differentiation (AD) [20, 14] as treated by means of open logical relations [3]. In a nutshell, AD can be presented as a mapping D on terms and types such that if $\Gamma \vdash t : \tau$ then $D(\Gamma) \vdash D(t) : D(\tau)$, with $D(t)$ computing the derivative of t. We will not detail the transformation here (a complete description can be found in [3, 14], as well as in the long version of this paper [8]). Proving the correctness of AD amounts to showing that for any first order term $\Theta \vdash t : \mathtt{Real}$, where $\Theta = \boldsymbol{x}_1 : \mathtt{Real}, \ldots, \boldsymbol{x}_n : \mathtt{Real}$, the (semantics of the) term $D(\Theta) \vdash D(t) : \mathtt{Real} \times \mathtt{Real}$ – with $D(\Theta) = d\boldsymbol{x}_1 : \mathtt{Real} \times \mathtt{Real}, \ldots, d\boldsymbol{x}_n : \mathtt{Real} \times \mathtt{Real}$ – is the derivative of (the semantics of) t. Correctness proofs have beed recently given, both semantically [14] and syntactically [3]. Here, we sketch how the syntactic proof by Barthe et al. [3] can be embedded in RHOL. First, given a proof of $\Gamma, \Theta \vdash t : \tau$, we obtain the judgment

$$\Gamma :: \Theta, D(\Gamma) :: D(\Theta) \mid \Psi \vdash t : \tau \sim D(t) : D(\tau) \ \{\boldsymbol{r}_1 \ \mathcal{R}^{\Theta}_{\tau} \ \boldsymbol{r}_2\}$$

where Ψ is a set of assertions relating elements of Γ and $D(\Gamma)$ and $\boldsymbol{r}_1 \ \mathcal{R}^{\Theta}_{\tau} \ \boldsymbol{r}_2$ is a formula corresponding to the logical relation by Barthe et al.[3]. The key point to stress is that in our (ROHOL) proof the flow of the derivation follows naturally from the definition of an open logical relation, as elegantly showed by the case of λ-abstractions. For a λ-abstraction, in fact, we have $D(\lambda y : \sigma.t) = \lambda dy : D(\sigma).D(t)$ and $D(\sigma \to \tau) = D(\sigma) \to D(\tau)$. Consequently, we have to derive the following judgment (assuming $\Gamma \vdash \lambda y.t : \sigma \to \tau$)

$$\Gamma :: \Theta, D(\Gamma) :: D(\Theta) \mid \Psi \vdash \lambda y.t : \sigma \to \tau \sim \lambda dy.D(t) : D(\sigma) \to D(\tau) \ \{\boldsymbol{r}_1 \ \mathcal{R}^{\Theta}_{\sigma \to \tau} \ \boldsymbol{r}_2\}.$$

The logical relation $t_1 \ \mathcal{R}^{\Theta}_{\sigma\to\tau}\ t_2$ for arrow types in this case is defined by

$$\forall y_1 : \sigma :: \Theta . \forall y_2 : D(\sigma) :: D(\Theta) . (y_1 \ \mathcal{R}^{\Theta}_{\sigma}\ y_2) \Rightarrow (t_1\ y_1)\ \mathcal{R}^{\Theta}_{\tau}\ (t_2\ y_2)$$

and this is exactly the shape of the λ-abstraction rule in ROHOL. Thus, we can apply the rule:

$$\frac{\Gamma :: \Theta, y : \sigma :: \Theta, D(\Gamma) :: D(\Theta), dy : D(\sigma) :: D(\Theta) \mid \Psi, (y_1 \ \mathcal{R}^{\Theta}_{\sigma}\ y_2) \vdash t : \tau \sim D(t) : D(\tau)\ \{\boldsymbol{r}_1\ \mathcal{R}^{\Theta}_{\tau}\ \boldsymbol{r}_2\}}{\Gamma :: \Theta, D(\Gamma) :: D(\Theta) \mid \Psi \vdash \lambda y.t : \sigma \to \tau \sim \lambda dy.D(t) : D(\sigma) \to D(\tau)\ \{\boldsymbol{r}_1\ \mathcal{R}^{\Theta}_{\sigma\to\tau}\ \boldsymbol{r}_2\}}$$

and conclude the proof by induction.

5 Local Open Higher-Order Logic

In this section, we go beyond OHOL and introduce a *local* version of OHOL allowing formulas to account for the domain of definition of functions. The motivation behind the introduction of such a logic is ultimately found in the interaction between the openness of OHOL – whereby terms of a ground type are actually regarded as first-order functions – and constructs of the language, whose correct semantic behavior relies on having actual terms of ground type – and not first-order functions – as arguments. To clarify, let us consider *the* main construct badly interacting with the current formulation of OHOL: the conditional. Let us consider the term construct $\mathsf{if}\ t\ \mathsf{then}\ u_1\ \mathsf{else}\ u_0$. The standard UHOL rule for this conditional would be:

$$\frac{\Gamma \vdash t : \mathtt{Bool} \qquad \Gamma \mid \Psi, t = \mathtt{tt} \vdash u_1 : \tau\ \{\phi\} \qquad \Gamma \mid \Psi, t = \mathtt{ff} \vdash u_0 : \tau\ \{\phi\}}{\Gamma \mid \Psi \vdash \mathsf{if}\ t\ \mathsf{then}\ u_1\ \mathsf{else}\ u_0 : \tau\ \{\phi\}}$$

expressing that this conditional satisfies a formula ϕ when both branches satisfy ϕ with the obvious assumption that in the first branch t is true and in the second branch t is false.

If we naively generalize this rule to UOHOL, we see that $t =^{\Theta} \mathtt{tt}$ does not describe equality between the boolean *values* t and $\mathtt{tt}$: instead, it gives equality between the boolean *function* t from $[\![\Theta]\!]$ to $[\![\mathtt{Bool}]\!]$ and the boolean *constant function* $\mathtt{tt}$ from $[\![\Theta]\!]$ to $[\![\mathtt{Bool}]\!]$. Consequently, the resulting rule would be unsound, as in the first branch we cannot assume t to be constantly equal to true (there are subsets of Θ in which t is true and others in which t is false).

To overcome this problem, we extend OHOL to take into account restricted (sub)domains of $[\![\Theta]\!]$. We do so by defining a new logic, *Local OHOL* (LOHOL, for short). Compared to OHOL, the main novelty of LOHOL is the introduction of formulas of the form $[S]\phi$, where S represents a subset of $[\![\Theta]\!]$, expressing that ϕ is a formula on functions with domain of definition S, instead of the whole $[\![\Theta]\!]$.

▶ **Definition 7.** *LOHOL's formulas are defined by the following grammar (we define disjunction, implication, and existential quantification by way of De Morgan's laws).*

$$\phi ::= P^{\Theta}(S_1, \dots, S_k; t_1, \dots, t_m) \mid \bot \mid \neg\phi \mid \phi \wedge \phi \mid \forall y : \tau.\phi \mid [S]\phi \mid \forall X.\phi$$
$$S ::= X \mid S \cup S \mid S \cap S \mid \emptyset \mid S^{\complement} \mid \{\Theta \mid P^{S}(t_1, \dots, t_m)\}.$$

Here, X is a set variable, $P^S(t_1, \dots, t_m)$ a predicate on terms that can use variables in Θ, and $P^{\Theta}(S_1, \dots, S_k; t_1, \dots, t_m)$ a predicate on arbitrary numbers of sets and terms.

We denote by $(k; \tau_1, \dots, \tau_m)$ the arity of a predicate P^{Θ}, with k the number of sets and τ_i type of t_i. This generic definition allows us to have predicates on terms only – as in OHOL – as well as predicates on sets only – such as the subset inclusion predicate. Finally,

$$\frac{}{X, \tilde{X} \mid \Gamma \vdash^{\Theta} X} \qquad \frac{(\Gamma, \Theta \vdash t_i : \tau_i)_{1 \le i \le m} \qquad P^S \text{ has arity } (\tau_1, \ldots, \tau_m)}{\tilde{X} \mid \Gamma \vdash^{\Theta} \{\Theta \mid P^S(t_1, \ldots, t_m)\}}$$

$$\frac{(\Gamma, \Theta \vdash t_i : \tau_i)_{1 \le i \le m} \qquad (\tilde{X} \mid \Gamma \vdash^{\Theta} S_j)_{1 \le j \le k} \qquad P^{\Theta} \text{ has arity } (k; \tau_1, \ldots, \tau_m)}{\tilde{X} \mid \Gamma \vdash^{\Theta} P^{\Theta}(S_1, \ldots, S_k; t_1, \ldots, t_m)}$$

$$\frac{\tilde{X} \mid \Gamma, y : \tau \vdash^{\Theta} \phi}{\tilde{X} \mid \Gamma \vdash^{\Theta} \forall y : \tau.\phi} \qquad \frac{X, \tilde{X} \mid \Gamma \vdash^{\Theta} \phi}{\tilde{X} \mid \Gamma \vdash^{\Theta} \forall X.\phi} \qquad \frac{\tilde{X} \mid \Gamma \vdash^{\Theta} S_2 \qquad \tilde{X} \mid \Gamma \vdash^{\Theta} \phi}{\tilde{X} \mid \Gamma \vdash^{\Theta} [S_2]\phi}$$

Figure 5 Well-Typing Judgments.

$\{\Theta \mid P^S(t_1, \ldots, t_m)\}$ is called the comprehension formula, as it gives a simple comprehension schema. For instance, $[\ \{\boldsymbol{x} : \texttt{Real} \mid \boldsymbol{x} \le 2\}]$ restricts the domain of definition from $\mathbb{R}$ to $]-\infty, 2]$.

Let us now move to LOHOL's semantics, beginning with the semantic of well-typed formulas. First, we define the well-typed judgments $\tilde{X} \mid \Gamma \vdash^{\Theta} S$ and $\tilde{X} \mid \Gamma \vdash^{\Theta} \phi$, where $\tilde{X}$ denotes a sequence of set variables occurring free in S and and ϕ. The most important rules are given in Figure 5. Such rules should be clear. In particular, the first rule, similar to an axiom rule, states that in a judgment of the form $\tilde{X} \mid \Gamma \vdash^{\Theta} S$ we cannot use set variables outside those in $\tilde{X}$. Notice also that in the last rule, the second premise shows that restricting an already restricted domain amounts to intersecting the domains themselves.

Next, we define the denotational semantics $[\![\tilde{X} \mid \Gamma \vdash^{\Theta} S]\!] : \widetilde{2^{[\![\Theta]\!]}} \times [\![\Gamma^{\Theta}]\!] \to 2^{[\![\Theta]\!]}$ of a well-typed set $\tilde{X} \mid \Gamma \vdash^{\Theta} S$, as mapping subsets of $[\![\Theta]\!]$ (one for each set variable in $\tilde{X}$) and functions in $[\![\tau]\!]^{[\![\Theta]\!]}$ to subsets of $[\![\Theta]\!]$. And finally, we define the semantics $[\![\tilde{X} \mid \Gamma \vdash^{\Theta} \phi]\!]_S \subseteq \widetilde{2^{[\![\Theta]\!]}} \times [\![\Gamma^{\Theta}]\!]$ of a well-typed assertion $\tilde{X} \mid \Gamma \vdash^{\Theta} \phi$ parameterized by a well-typed set $\tilde{X} \mid \Gamma \vdash^{\Theta} S$ as giving the possible values for set variables in $\tilde{X}$ and variables in Γ for which the formula ϕ, on the restricted domain represented by S, is true. The use of this parameter S is useful for the inductive definition, but in practice for a well-typed assertion $\tilde{X} \mid \Gamma \vdash^{\Theta} \phi$, its semantics is defined as $[\![\tilde{X} \mid \Gamma \vdash^{\Theta} \phi]\!]_{\emptyset^{\complement}}$, taking as parameter the set representing $[\![\Theta]\!]$.

▶ **Definition 8.** *Let* $[\![P^S]\!] \subseteq [\![\tau_1]\!] \times \cdots \times [\![\tau_m]\!]$ *and* $[\![P^{\Theta}]\!] \subseteq 2^{[\![\Theta]\!]^k} \times \coprod_{S \subseteq [\![\Theta]\!]} \prod_{1 \le i \le m} [\![\tau_i]\!]^S$ *be interpretations of predicates* P^S *of arity* $(\tau_1, \ldots, \tau_m)$ *and* P^{Θ} *of arity* $(k; \tau_1, \ldots, \tau_m)$*, respectively. Then, the semantics of well-typed judgments and of well-typed assertions is inductively defined thus:*

$$[\![X, \tilde{X} \mid \Gamma \vdash^{\Theta} X]\!]((X, \tilde{X}), \tilde{y}) = X$$

$$[\![\tilde{X} \mid \Gamma \vdash^{\Theta} \{\Theta \mid P^S(t_1, \ldots, t_m)\}]\!](\tilde{X}, \tilde{y}) = \{\tilde{x} \in [\![\Theta]\!] \mid [\![P^S]\!] \ni ([\![\Gamma, \Theta \vdash t_i : \tau_i]\!](\tilde{y}(\tilde{x}), \tilde{x}))_{1 \le i \le m}\}$$

$$[\![\tilde{X} \mid \Gamma \vdash^{\Theta} S_1 \cup S_2]\!](\tilde{X}, \tilde{y}) = [\![\tilde{X} \mid \Gamma \vdash^{\Theta} S_1]\!](\tilde{X}, \tilde{y}) \cup [\![\tilde{X} \mid \Gamma \vdash^{\Theta} S_2]\!](\tilde{X}, \tilde{y})$$

$$[\![\tilde{X} \mid \Gamma \vdash^{\Theta} P^{\Theta}(S_1, \ldots, S_k; t_1, \ldots, t_m)]\!]_S =$$
$$\{(\tilde{X}, \tilde{y}) \mid [\![P^{\Theta}]\!] \ni (([\![\tilde{X} \mid \Gamma \vdash^{\Theta} S_i]\!](\tilde{X}, \tilde{y}))_{1 \le i \le k}, ((\![\Gamma, \Theta \vdash t_i : \tau_i]\!] \star \tilde{y})_{|[\![\tilde{X} \mid \Gamma \vdash^{\Theta} S]\!](\tilde{X}, \tilde{y})})_{1 \le i \le m})\}$$

$$[\![\tilde{X} \mid \Gamma \vdash^{\Theta} \forall y : \tau.\phi]\!]_S = \{(\tilde{X}, \tilde{y}) \mid \forall y \in [\![\Theta \Rightarrow \tau]\!], (\tilde{X}, (\tilde{y}, y)) \in [\![\tilde{X} \mid \Gamma, y : \tau \vdash^{\Theta} \phi]\!]_S\}$$

$$[\![\tilde{X} \mid \Gamma \vdash^{\Theta} \forall X.\phi]\!]_S = \{(\tilde{X}, \tilde{y}) \mid \forall X \in 2^{[\![\Theta]\!]}, ((X, \tilde{X}), \tilde{y}) \in [\![X, \tilde{X} \mid \Gamma \vdash^{\Theta} \phi]\!]_S\}$$

$$[\![\tilde{X} \mid \Gamma \vdash^{\Theta} [S']\phi]\!]_S = [\![\tilde{X} \mid \Gamma \vdash^{\Theta} \phi]\!]_{S \cap S'}.$$

Let us comment on some important points of Definition 8. First, the interpretation of set predicates $P^S(t_1, \ldots, t_m)$ coincides with the interpretation of HOL predicates. This way, set predicates can work on ground terms, as in $t = \texttt{tt}$. The interpretation of predicates P^{Θ}, instead, is a subset of $2^{[\![\Theta]\!]^k} \times \coprod_{S \subseteq [\![\Theta]\!]} \prod_{1 \le i \le m} [\![\tau_i]\!]^S$. The first projection $2^{[\![\Theta]\!]^k}$ represents

the k subsets of $[\![\Theta]\!]$, whereas the second projection corresponds to the type of predicates in OHOL, but for all the possible subsets of Θ. As an example, consider $\Theta = \boldsymbol{x} : \texttt{Real}$ and let $\mathcal{C}^\Theta$ be a predicate of arity $(0; \texttt{Real})$ for local continuity. Then, we would interpret $\mathcal{C}^\Theta$ as

$$[\![\mathcal{C}^\Theta]\!] = \{f : A \to \mathbb{R} \mid A \subseteq \mathbb{R} \text{ and } f \text{ is locally continuous on } A\}.$$

The most interesting case of Definition 8 is the one for LOHOL predicates. The key point of this definition, and the main difference with OHOL, is the restriction operation $([\![\Gamma, \Theta \vdash t_i : \tau_i]\!] \star \tilde{y})_{|[\![\tilde{X} \mid \Gamma \vdash^\Theta S]\!](\tilde{X}, \tilde{y})}$. The considered function $([\![\Gamma, \Theta \vdash t_i : \tau_i]\!] \star \tilde{y})$ is the same as the one of OHOL but, as expected for LOHOL, we have to restrict its domain of definition to the subset represented by S, which is given by $[\![\tilde{X} \mid \Gamma \vdash^\Theta S]\!](\tilde{X}, \tilde{y})$.[1]

Definition 8 also shows that the constructor $[S]$ is semantically relevant only for atomic predicates. Syntactically, this implies that it is always possible to commute this constructor with other connectives, hence pushing it just before atomic formulas.

▶ **Proposition 9.** *We have the following logical equivalences:*

$$[S]\top \equiv \top \qquad [S]\neg\phi \equiv \neg[S]\phi \qquad [S][T]\phi \equiv [S \cap T]\phi \qquad [\emptyset^{\mathtt{C}}]\phi \equiv \phi$$
$$[S](\phi \wedge \psi) \equiv [S]\phi \wedge [S]\psi \qquad [S]\forall y : \tau.\phi \equiv \forall y : \tau.[S]\phi \qquad [S]\forall X.\phi \equiv \forall X.[S]\phi$$

From Proposition 9 it follows that a judgment system for LOHOL needs not have a rule covering $[S]$, as it is enough to consider atomic formulas of the form $[S]P^\Theta(\dots)$.

▶ Remark 10. In the following, we will still use the notation $[S]\phi$ to improve readability. Moreover, we will tacitly assume to have defined an equivalence relation on sets that we can apply in formulas, so to identify sets (trivially) having the same semantics, such as $S \cap (T \cap U)$ and $(S \cap T) \cap U$.

We now give an inference judgment system for LOHOL. The rules are close to the ones presented in Figure 3 and are given in Figure 6. A judgment has the shape $\tilde{X} \mid \Gamma \mid \Psi \vdash^\Theta \phi$, and we ask that all formulas and sets are well-typed. The main difference with OHOL is that we have to keep track of set variables. For the last rule, we use the following definition.

▶ **Definition 11** (Functional Extension)**.** *The pointwise extension of a predicate P^S with arity $(\tau_1, \dots, \tau_m)$ is the predicate P^S_p of arity $(0; \tau_1, \dots, \tau_m)$ semantically interpreted by:*

$$[\![P^S_p]\!] = \{(f_1, \dots, f_m) \in \textstyle\prod_{1 \le i \le m} [\![\tau_i]\!]^S \mid S \subseteq [\![\Theta]\!] \wedge \forall \tilde{x} \in S.(f_1(\tilde{x}), \dots, f_m(\tilde{x})) \in [\![P^S]\!]\}$$

▶ **Example 12.** The pointwise extension $\le_p$ of $\le$ on real numbers corresponds to the usual pointwise comparison (on a possibly restricted domain of definition). Notice that the last rule of Figure 6 proves e.g. the formula $[\boldsymbol{x} + 3 \le 2](\boldsymbol{x} + 3 \le_p 2)$ showing that restricting the domain of definition to $]-\infty, -1]$ ensures that the function $x \mapsto x + 3$ is always smaller that the constant function $x \mapsto 2$ on this domain.

It is easy to show that the system defined in Figure 6 is indeed sound.

▶ **Theorem 13.** *We have $[\![\tilde{X} \mid \Gamma \vdash^\Theta \bigwedge_{\psi \in \Psi} \psi]\!]_{\emptyset^{\mathtt{C}}} \subseteq [\![\tilde{X} \mid \Gamma \vdash^\Theta \phi]\!]_{\emptyset^{\mathtt{C}}}$ for any derivable judgment $\tilde{X} \mid \Gamma \mid \Psi \vdash^\Theta \Phi$.*

[1] Notice that in a set-theoretical semantics, the notion of restriction indeed corresponds to the usual notion of function restrictions. If one wants to give a categorical meaning to this and consider other categories than **Set**, she could consider presheaves semantics for terms.

$$\frac{}{\tilde{X} \mid \Gamma \mid \Psi, \phi \vdash^{\Theta} \phi} \qquad \frac{\Gamma, \Theta \vdash t_i : \tau \qquad t_1 =_{(\to)} t_2 \qquad \tilde{X} \mid \Gamma \vdash^{\Theta} S}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} [S](t_1 = t_2)}$$

$$\frac{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} [S]\phi[t_1/y] \qquad \tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} [S](t_1 = t_2)}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} [S]\phi[t_2/y]} \qquad \frac{\tilde{X} \mid \Gamma, y : \tau \mid \Psi \vdash^{\Theta} \phi}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \forall y : \tau.\phi}$$

$$\frac{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \forall y : \tau.\phi \qquad \Gamma, \Theta \vdash^{\Theta} t : \tau}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \phi[t/y]} \qquad \frac{\tilde{X}, X \mid \Gamma \mid \Psi \vdash^{\Theta} \phi}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \forall X.\phi}$$

$$\frac{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \forall X.\phi \quad \tilde{X} \mid \Gamma \vdash^{\Theta} S}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \phi[S/X]} \qquad \frac{P_p^S \text{ pointwise extension of } P^S}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} [S \cap (\ \{\Theta \mid P^S(t_1, \ldots, t_m)\})]P_p^S(t_1, \ldots, t_m)}$$

Figure 6 Inference Judgment System for LOHOL.

$$\frac{\tilde{X} \mid \Gamma, y : \tau \mid \Psi \vdash^{\Theta} \phi[y/\boldsymbol{r}]}{\tilde{X} \mid \Gamma, y : \tau \mid \Psi \vdash^{\Theta} y : \tau\ \{\phi\}} \qquad \frac{(\boldsymbol{x}_i : \mathtt{B}_i) \in \Theta \qquad \tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \phi[\boldsymbol{x}_i/\boldsymbol{r}]}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \boldsymbol{x}_i : \mathtt{B}_i\ \{\phi\}} \qquad \frac{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \phi[\underline{c}/\boldsymbol{r}]}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \underline{c} : \mathtt{B}\ \{\phi\}}$$

$$\frac{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \phi[\underline{f}/\boldsymbol{r}]}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \underline{f} : \tilde{\mathtt{B}} \to \mathtt{B}\ \{\phi\}} \qquad \frac{\tilde{X} \mid \Gamma, y : \tau \mid \Psi, \psi \vdash^{\Theta} t : \sigma\ \{\phi\}}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \lambda y.t : \tau \to \sigma\ \{\forall y.\psi \Rightarrow \phi[\boldsymbol{r}\ y/\boldsymbol{r}]\}}$$

$$\frac{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t : \tau \to \sigma\ \{\forall y.\psi[y/\boldsymbol{r}] \Rightarrow \phi[\boldsymbol{r}\ y/\boldsymbol{r}]\} \qquad \tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} u : \tau\ \{\psi\}}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t\ u : \sigma\ \{\phi[u/y]\}}$$

$$\frac{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t_i : \tau_i\ \{\phi_i\} \qquad \tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \forall y, z.\phi_1[y/\boldsymbol{r}] \wedge \phi_2[z/\boldsymbol{r}] \Rightarrow \phi[\langle y, z\rangle/\boldsymbol{r}]}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \langle t_1, t_2\rangle : \tau_1 \times \tau_2\ \{\phi\}}$$

$$\frac{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t : \tau_1 \times \tau_2\ \{\phi[\pi_i(\boldsymbol{r})/\boldsymbol{r}]\}}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \pi_i(t) : \tau_i\ \{\phi\}} \qquad \frac{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t : \tau\ \{\psi\} \qquad \tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \psi[t/\boldsymbol{r}] \Rightarrow \phi[t/\boldsymbol{r}]}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t : \tau\ \{\phi\}}$$

$$\frac{\tilde{X}, X \mid \Gamma \mid \Psi \vdash^{\Theta} t : \tau\ \{\phi\}}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t : \tau\ \{\forall X.\phi\}} \qquad \frac{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t : \tau\ \{\forall X.\phi\} \qquad \tilde{X} \mid \Gamma \vdash^{\Theta} S}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t : \tau\ \{\phi[S/X]\}}$$

$$\frac{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t : \mathtt{Bool}\ \{\phi_t\} \qquad \begin{array}{c}\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} u_1 : \tau\ \{[\ \{\Theta \mid t = \mathtt{tt}\}]\phi_1\} \\ \tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} u_0 : \tau\ \{[\ \{\Theta \mid t = \mathtt{ff}\}]\phi_0\}\end{array}}{\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \mathsf{if}\ t\ \mathsf{then}\ u_1\ \mathsf{else}\ u_0 : \tau\ \{\phi_t[t/r] \wedge [\ \{\Theta \mid t = \mathtt{tt}\}]\phi_1 \wedge [\ \{\Theta \mid t = \mathtt{ff}\}]\phi_0\}}$$

Figure 7 Some Rules of ULOHOL.

Finally, we design a framework for unary predicates, called ULOHOL, similarly to UOHOL. Remarkably, ULOHOL allows us to give a satisfactory rule for conditionals. Most of the defining rules of ULOHOL are straightforward (but notice that, contrary to UOHOL, we have to keep track of set variables). We give some of the most relevant ones in Figure 7. Finally, we show the equivalence with the judgment system of LOHOL.

▶ **Theorem 14** (Equivalence between ULOHOL and LOHOL). *For every sequence of set variables $\tilde{X}$, context Γ, type τ, term t, set of assertions Ψ and assertion ϕ, $\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t : \tau\ \{\phi\}$ is derivable if and only if $\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} \phi[t/\boldsymbol{r}]$ is.*

Proof. The important implication is the right-to-left one. The proof proceeds by induction on $\tilde{X} \mid \Gamma \mid \Psi \vdash^{\Theta} t : \tau\ \{\phi\}$ following the usual proof in UHOL. For the last rule, the desired implication follows from $[t =_{\mathtt{Bool}} \mathsf{ff}](t =_{\Theta \to \mathtt{Bool}} \mathsf{ff})$, as pointwise equality corresponds to functional equality, and so we can substitute t by ff in the formula $[t = \mathsf{ff}]\phi_0$, which is semantically equivalent to replacing $\mathsf{if}\ t\ \mathsf{then}\ u_1\ \mathsf{else}\ u_0$ by u_0 (and similarly for ϕ_1). ◀

$$\models \forall X, Y.[X]\mathcal{O}(Y) \Leftrightarrow \mathcal{O}(Y) \qquad (ORestr)$$
$$\models \mathcal{O}(\ \{\Theta \mid (x < \underline{c}) = \mathsf{tt}\}) \qquad (ORight)$$
$$\models \mathcal{O}(\ \{\Theta \mid (x > \underline{c}) = \mathsf{tt}\}) \qquad (OLeft)$$
$$\models \forall X, Y.(\mathcal{O}(X) \wedge \mathcal{O}(Y)) \Rightarrow (\mathcal{O}(X \cup Y) \wedge \mathcal{O}(X \cap Y)) \qquad (OStab)$$
$$\models \forall X, Y, z.([X]\mathcal{C}(z) \wedge (Y \subseteq X)) \Rightarrow [Y]\mathcal{C}(z) \qquad (CSubsets)$$
$$\models \forall X, Y, z.(\mathcal{O}(X) \wedge \mathcal{O}(Y) \wedge [X]\mathcal{C}(z) \wedge [Y]\mathcal{C}(z)) \Rightarrow [X \cup Y]\mathcal{C}(z) \qquad (COpenUnion)$$

Figure 8 Defining axioms of the predicates $\mathcal{O}$ and $\mathcal{C}$.

5.1 LOHOL at Work

Having introduced L(U)OHOL formally, let us now see the benefits of its characteristic features – namely openness, locality, and a novel rule for conditionals – on a nontrivial example: local continuity of real-valued functions. More specifically, we take inspiration from the work by Mazza and Pagani [16] (where it is shown that the points of error of AD on a PCF language form a set of measure zero) and show how our system can be used to prove *local continuity* on *open sets* of $\mathbb{R}$, this way getting rid of those discontinuity points that may occur due to the presence of conditionals.

Let us consider the context $\Theta = x : \mathtt{Real}$ and let t be the term $\mathsf{if}\ (x < 3)\ \mathsf{then}\ u_1\ \mathsf{else}\ u_0$. We want to show that t is continuous on a set that excludes: *(i)* the possible points of discontinuity of u_1; *(ii)* the possible points of discontinuity of u_0; *(iii)* the point $x = 3$, for which continuity is not assured. To do that, we need predicates and axioms for open sets, including stability of continuity by union of open sets. Consequently, we introduce the predicates $\mathcal{O}(S)$, of arity $(1;\cdot)$, and $\mathcal{C}(t)$, of arity $(0;\mathtt{Real})$. Our goal is then to prove a formula of the shape $(\mathcal{O}(S) \wedge [S]\mathcal{C}(t))$, meaning that t is locally continuous on the open set $S \subseteq \mathbb{R}$. The defining axioms of $\mathcal{O}$ and $\mathcal{C}$ are given in Figure 8, where we assume common predicates for sets (such as $\subseteq$) as given.

The first axiom states that restriction has no effect on $\mathcal{O}$, as there are simply no function to restrict, whereas the remaining axioms describe standard properties of open sets and continuity. Notice that we use the formula $(x \leq 3 = \mathsf{tt})$ instead of $x \leq 3$. This is because we shall construct open sets using the boolean $\Lambda_{\mathcal{C},\mathcal{F}}$ function $<$, and thus it is natural to consider the predicate $(x \leq 3 = \mathsf{tt})$. Nonetheless, it is also possible to use the formula $x \leq 3$ as well, although we would need to add an axiom linking the function $< \in \mathcal{F}$ and the set predicate $<$ of LOHOL.

Let us now consider the formulas ϕ_i, with $i \in \{0, 1\}$, defined by

$$\phi_i = [\ \{\Theta \mid (x < 3) = \mathsf{tt}\}](\mathcal{O}(S_i) \wedge [S_i]\mathcal{C}(\boldsymbol{r}))$$

and the judgments $\cdot \mid \cdot \vdash^{\Theta} u_i : \mathtt{Real}\ \{\phi_i\}$ which, altogether, give derivations of the form

$$\frac{\cdot \mid \cdot \vdash^{\Theta} (x < 3) : \mathtt{Bool}\ \{\phi_b\} \qquad \cdot \mid \cdot \vdash^{\Theta} u_0 : \mathtt{Real}\ \{\phi_0\} \qquad \cdot \mid \cdot \vdash^{\Theta} u_1 : \mathtt{Real}\ \{\phi_1\}}{\cdot \mid \cdot \vdash^{\Theta} \mathsf{if}\ (x < 3)\ \mathsf{then}\ u_1\ \mathsf{else}\ u_0 : \mathtt{Real}\ \{\phi_b \wedge \phi_1 \wedge \phi_0\}}$$

The target formula of this proof is not very satisfactory, as it forces us to use the implication rule. This, after all, is not surprising: conditionals constitute a complex case to prove continuity, and a generic rule for conditional cannot be expected to work directly on such a complex case. We overcome this problem by deriving a new rule for the conditional taking advantages of the axiomatic theory of $\mathcal{O}$ and $\mathcal{C}$.

▶ **Proposition 15.** *The following rule is derivable.*

$$\frac{\Gamma \mid \Psi \vdash^{\Theta} b : \mathtt{Bool}\ \{\mathcal{O}(R_0) \wedge \mathcal{O}(R_1) \wedge (R_0 \subseteq \ \{\Theta \mid b = \mathtt{ff}\}) \wedge (R_1 \subseteq \ \{\Theta \mid b = \mathtt{tt}\})\} \quad (\forall i \in \{0,1\})\ \Gamma \mid \Psi \vdash^{\Theta} t_i : \mathtt{Real}\ \{(\mathcal{O}(S_i) \wedge [S_i]\mathcal{C}(\boldsymbol{r}))\}}{\Gamma \mid \Psi \vdash^{\Theta} \mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_0 : \mathtt{Real}\ \{\mathcal{O}((S_0 \cap R_0) \cup (S_1 \cap R_1)) \wedge [(S_0 \cap R_0) \cup (S_1 \cap R_1)]\mathcal{C}(\boldsymbol{r})\}}$$

Proof. First, we show $(\mathcal{O}(S_0) \wedge [S_0]\mathcal{C}(\boldsymbol{r})) \Rightarrow [\ \{\Theta \mid b = \mathtt{tt}\}](\mathcal{O}(S_0) \wedge [S_0]\mathcal{C}(\boldsymbol{r}))$, for any term b. Recall that the constructor $[S]$ on the top level should always be understood as pushed in front of atomic formulas, so that what we actually need to prove is

$$(\mathcal{O}(S_0) \wedge [S_0]\mathcal{C}(\boldsymbol{r})) \Rightarrow [\ \{\Theta \mid b = \mathtt{tt}\}]\mathcal{O}(S_0) \wedge [S_0 \cap\ \{\Theta \mid b = \mathtt{tt}\}]\mathcal{C}(\boldsymbol{r}))$$

This follows from the axioms (Orestr) and (CSubsets), using the axiom $X \cap Y \subseteq X$ (which we assume as a defining axiom of the predicate $\subseteq$). To conclude, we prove

$$(\phi_b \wedge \phi_1 \wedge \phi_0) \Rightarrow \mathcal{O}((S_0 \cap R_0) \cup (S_1 \cap R_1) \wedge [(S_0 \cap R_0) \cup (S_1 \cap R_1)]\mathcal{C}(\boldsymbol{r}),$$

where ϕ_b is the formula $\mathcal{O}(R_0) \wedge \mathcal{O}(R_1) \wedge (R_0 \subseteq \ \{\Theta \mid b = \mathtt{ff}\}) \wedge (R_1 \subseteq \ \{\Theta \mid b = \mathtt{tt}\})$. This is a consequence of the axioms in Figure 8: openness of $(S_0 \cap R_0) \cup (S_1 \cap R_1)$ comes from (Ostab), whereas continuity of $\boldsymbol{r}$ follows from (CSubsets) and (COpenUnion). ◀

Armed with Proposition 15, let us come back to our main example. Assuming an appropriate axiomatization of the relations $<, >, \leq, \geq$, we define $R_1 = \ \{\Theta \mid (x < 3) = \mathtt{tt}\}$, $R_0 = \ \{\Theta \mid (x > 3) = \mathtt{tt}\}$, and derive the judgment

$$\cdot \mid \cdot \vdash^{\Theta} (x < 3) : \mathtt{Bool}\ \{\mathcal{O}(R_0) \wedge \mathcal{O}(R_1) \wedge (R_0 \subseteq \ \{\Theta \mid (x < 3) = \mathtt{ff}\}) \wedge (R_1 \subseteq \ \{\Theta \mid (x < 3) = \mathtt{tt}\})\}.$$

Then, using the rule for conditionals, we infer the possible discontinuity points of t, as well as those of u_1 (resp. u_0) smaller (resp. greater) than 3, and the point $\boldsymbol{x} = 3$ itself.

6 Conclusion

We have introduced several systems of *open* higher-order logic, i.e. extensions of HOL capable of dealing with first-order properties natively. Even if our logics do not increase the expressive power of HOL, they considerably improve its effectiveness, allowing for compositional proofs of nontrivial program properties, such as (local) continuity and differentiability. We have tested our formalism on several nontrivial examples, obtaining compositional proofs of state-of-the-art results (such as correctness of automatic differentiation) in a formal framework.

The number of extensions of OHOL defined and the heterogeneity of the examples analyzed suggest that OHOL (and variations thereof) may play an important role in the formal analysis of higher-order programming languages. Prompted by that observation, the authors plan to investigate applications of the logical systems to programs with effects (e.g. probabilistic programs) and to intensional program analyses (such as quantitative reasoning and program complexity).

References

1 Alejandro Aguirre, Gilles Barthe, Lars Birkedal, Aleš Bizjak, Marco Gaboardi, and Deepak Garg. Relational reasoning for markov chains in a probabilistic guarded lambda calculus. In *Proc. of ESOP 2018*, volume 10801 of *LNCS*, pages 214–241. Springer, 2018. doi:10.1007/978-3-319-89884-1_8.

2 Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Pierre-Yves Strub. A relational logic for higher-order programs. *Proc. ACM Program. Lang.*, 1(ICFP):21:1–21:29, 2017. doi:10.1145/3110265.

3 Gilles Barthe, Raphaëlle Crubillé, Ugo Dal Lago, and Francesco Gavazzo. On the versatility of open logical relations: Continuity, automatic differentiation, and a containment theorem. In *Proc. of ESOP 2020*, volume 12075 of *LNCS*. Springer, 2020. doi:10.1007/978-3-030-44914-8_3.

4 Edwin Brady. Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming*, 23(5):552–593, 2013. doi:10.1017/S095679681300018X.

5 Stephen A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, 1978. doi:10.1137/0207005.

6 Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for lambda-terms. *Archiv fur mathematische Logik und Grundlagenforschung*, 19(1):139–156, 1978. doi:10.1007/BF02011875.

7 Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. In *Proc. of LICS 2011*, pages 133–142, 2011. doi:10.1109/LICS.2011.22.

8 Ugo Dal Lago, Francesco Gavazzo, and Alexis Ghyselen. Open higher-order logic (long version), 2022. arXiv:2211.06671.

9 Rowan Davies and Frank Pfenning. Intersection types and computational effects. In *Proc. of ICFP 2000*, pages 198–208, 2000. doi:10.1145/351240.351259.

10 Robert W Floyd. Assigning meanings to programs. In *Program Verification*, pages 65–81. Springer, 1993.

11 Alejandro Aguirre Galindo. *Relational Logics for Higher-Order Effectful Programs*. Ph.D Thesis, 2020.

12 C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969. doi:10.1145/363235.363259.

13 Martin Hofmann. Linear types and non-size-increasing polynomial time computation. *Information and Computation*, 183(1):57–85, 2003. doi:10.1016/S0890-5401(03)00009-9.

14 Mathieu Huot, Sam Staton, and Matthijs Vákár. Correctness of automatic differentiation via diffeologies and categorical gluing. In *Proc. of FoSSACS 2020*, volume 12077 of *LNCS*, pages 319–338. Springer, 2020.

15 Achim Jung and Jerzy Tiuryn. A new characterization of lambda definability. In *Proc. of TLCA 1993*, volume 664 of *LNCS*, pages 245–257. Springer, 1993. doi:10.1007/BFb0037110.

16 Damiano Mazza and Michele Pagani. Automatic differentiation in PCF. *Proc. ACM Program. Lang.*, 5(POPL):1–27, 2021. doi:10.1145/3434309.

17 Gordon Plotkin. Lambda-definability and logical relations. Memorandum SAI-RM-4, University of Edinburgh, 1973.

18 Francois Pottier. A simple view of type-secure information flow in the pi-calculus. In *Proc. of CSFW 2002*, pages 320–330, 2002. doi:10.1109/CSFW.2002.1021826.

19 Tetsuya Sato, Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Justin Hsu. Formal verification of higher-order probabilistic programs: Reasoning about approximation, convergence, bayesian inference, and optimization. *Proc. ACM Program. Lang.*, 3(POPL):38:1–38:30, 2019. doi:10.1145/3290351.

20 Amir Shaikhha, Andrew Fitzgibbon, Dimitrios Vytiniotis, and Simon Peyton Jones. Efficient differentiable programming in a functional array-processing language. *Proc. ACM Program. Lang.*, 3(ICFP):97:1–97:30, 2019. doi:10.1145/3341701.

21 Richard Statman. Logical relations and the typed lambda-calculus. *Information and Control*, 65(2):85–97, 1985. doi:10.1016/S0019-9958(85)80001-2.

22 Sam Staton, Frank Wood, Hongseok Yang, Chris Heunen, and Ohad Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Proc. of LICS 2016*, pages 525–534, 2016. doi:10.1145/2933575.2935313.

Frobenius Structures in Star-Autonomous Categories

Cédric de Lacroix
LIS, CNRS UMR 7020, Aix-Marseille Université, Marseille, France

Luigi Santocanale
LIS, CNRS UMR 7020, Aix-Marseille Université, Marseille, France

Abstract

It is known that the quantale of sup-preserving maps from a complete lattice to itself is a Frobenius quantale if and only if the lattice is completely distributive. Since completely distributive lattices are the nuclear objects in the autonomous category of complete lattices and sup-preserving maps, we study the above statement in a categorical setting. We introduce the notion of Frobenius structure in an arbitrary autonomous category, generalizing that of Frobenius quantale. We prove that the monoid of endomorphisms of a nuclear object has a Frobenius structure. If the environment category is star-autonomous and has epi-mono factorizations, a variant of this theorem allows to develop an abstract phase semantics and to generalise the previous statement. Conversely, we argue that, in a star-autonomous category where the monoidal unit is a dualizing object, if the monoid of endomorphisms of an object has a Frobenius structure and the monoidal unit embeds into this object as a retract, then the object is nuclear.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Constructive mathematics; Theory of computation → Proof theory

Keywords and phrases Quantale, Frobenius quantale, Girard quantale, associative algebra, star-autonomous category, nuclear object, adjoint

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.18

Related Version *Full Version*: `https://hal.archives-ouvertes.fr/hal-03739197`

Funding Work supported by the ANR project LAMBDACOMB ANR-21-CE48-0017.

1 Introduction

Context. A major motivation for giving birth to linear logic was to restore a classical negation in a constructive setting. This was achieved on the proof-theoretic side, by controlling the structural rules of weakening and contraction. As a byproduct, the logical connectives of conjunction and disjunction have been split into their multiplicative and additive versions, and the additive connectives no longer have a classical behaviour, that is, they no longer distribute over each other.

We might tackle the same problem – restoring a classical negation in a constructive setting – using algebraic and model theoretic approaches and considering variants of linear logic. A standard complete provability semantics of (possibly non-commutative) linear logic is on the class of structures known as quantales, see e.g. [40]. A quantale $(Q, *)$ is a complete lattice with a multiplication which distributes over sups in both variables (i.e., it is a semigroup in the category SLatt of complete lattices and join-preserving maps). Restoring classical negation means, in this context, considering Frobenius or Girard quantales. A quantale is Frobenius if it also comes with two suitable antitone maps ${}^{\perp}(-)$ and $(-)^{\perp}$ representing negations, and it is Girard if these two maps coincide. Among quantales, probably the most interesting are those on the set $[L, L]$ of sup-preserving endomaps of a complete lattice L, with multiplication given by composition and the ordering computed pointwise. These quantales

naturally arise in logic, computer science, and quantum mechanics. For example, for a set X and its powerset lattice $P(X)$, the quantale $[P(X), P(X)]$ is isomorphic to the quantale $P(X \times X)$ of relations on X and it is the main object of study in relation algebra [28]. Use of these quantales to model concurrency and quantum mechanics/computation abound in the literature, see e.g. [3, 4, 30]. Our interest for quantales of the form $[L, L]$ stems from generalized linear orders [19, 33, 20, 36], which are sort of fuzzy relations whose set of truth values is a quantale. These relations need to satisfy constraints which are expressed using the negation, hence the quantale must be a Frobenius quantale. While developing a theory of these structures we came across (and contributed to) the following result:

▶ **Theorem 1** (See [26, 14, 16, 35, 34]). *The quantale $[L, L]$ of sup-preserving endomaps of a complete lattice L is a Frobenius quantale if and only if L is completely distributive.*

Complete distributivity is an infinitary generalization of the usual distributive law. Basic examples of completely distributive lattices are the complete linear orders, the powerset lattices and, of course, the finite distributive lattices. As logicians, the above theorem struck us, since it shows that a model theoretic approach to restoring a classical negation fundamentally diverges from the proof-theoretic one. Indeed, the theorem has the following reading: *if a classical negation is enforced on the most standard class of models of non-commutative intuitionistic linear logic* (the sup-preserving endomaps of a complete lattice)*, then also the additive connectives of the logic* (the suprema and infima of the lattice) *have a classical behaviour* (that is, they distribute over each other).

Contribution. Due to its strength, the above statement deserves to be studied from the largest number of perspectives. We take here a categorical approach and give a proof of the statement that relies on the $*$-autonomous structure of SLatt, the category of complete lattices and sup-preserving maps. In doing so, we generalise the statement to $*$-autonomous categories. This is possible due to a fundamental characterization of complete distributivity in the categorical language, stating that *the completely distributive lattices are exactly the nuclear objects in* SLatt, see [29, 22]. We recall that an object A of a symmetric monoidal closed category $\mathcal{V} = (V, I, \otimes, \alpha, \lambda, \rho, [-,-], ev)$ is *nuclear* if the canonical map $\mathtt{mix} : A^* \otimes A \longrightarrow [A, A]$ is an isomorphism, where A^*, the dual of A, is the internal hom $[A, I]$ (see e.g. [32]).

To achieve our goal, we define Frobenius structures in a symmetric monoidal closed category by strictly mimicking the definition of (possibly unitless) Frobenius quantales given in [13]. Definitions of similar structures appear in the literature [25, 23, 39, 15] and differ from ours mainly w.r.t. the environment category and the presence of units. We consider the choice of an axiomatization only as a tool for our goal of generalizing Theorem 1 to a categorical setting. We insist, however, that the theorem is about characterizing when a canonical quantale on the tensor product $L^* \otimes L$ of an object L in SLatt has a unit. To define Frobenius structures, we use the notion of *dual pairing* in a monoidal category $\mathcal{V}$. This is a map $\epsilon : A \otimes B \longrightarrow I$ with two given universal properties. It turns out that in a $*$-autonomous category such a dual pairing exists if and only if B is isomorphic to A^*, if and only if A is isomorphic to B^*. This notion provides the framework by which to study objects that are dual to each other only up to isomorphism: for example $(A^* \otimes A, [A, A])$ is part of a dual pairing in any $*$-autonomous category and, for any complete lattice L, (L, L^{op}) is part of a dual pairing in SLatt. If $\epsilon : A \otimes B \longrightarrow I$ is a dual pairing and A is a semigroup, then A acts on B on the left and on the right. The left and right actions, noted α^ℓ_A and α^ρ_A, correspond, in the category SLatt, to the two implications of a quantale. We define generalized Frobenius quantales in arbitrary symmetric monoidal category as follows:

▶ **Definition.** A *Frobenius structure* is a tuple $(A, B, \epsilon, \mu_A, l, r)$ where (A, B, ϵ) is a dual pairing, (A, μ_A) is a semigroup, l and r are invertible maps from A to B such that

$$\epsilon \circ (A \otimes l) = \epsilon \circ (A \otimes r) \circ \sigma_{A,A} \qquad \text{and} \qquad \alpha^{\ell}_{A} \circ (A \otimes r) = \alpha^{\rho}_{A} \circ (l \otimes A)\,. \tag{1}$$

Almost by definition, in SLatt, a Frobenius structure of the form $(Q, Q^{op}, \epsilon, *, {}^{\perp}(-), (-)^{\perp})$ amounts to a Frobenius quantale, as defined in [13], that is a quantale $(Q, *)$ coming with antitone negations satisfying

$$x \leq {}^{\perp}y \text{ iff } y \leq x^{\perp} \qquad \text{and} \qquad y^{\perp}/x = y\backslash{}^{\perp}x\,.$$

We prove then the following result, generalizing the direct implication of Theorem 1.

▶ **Theorem A.** *If A is nuclear, then there is a map l such that $([A, A], [A, A]^*, ev_{[A,A],I}, \circ, l, l)$ is a Frobenius structure.*

The proof of this theorem is almost straightforward from the definition of Frobenius structure. Let us mention that Theorem A is strictly related to (and may be considered an instance of) Corollary 3.3 in [39]. The connection, however, depends on the fact that adjoints in a symmetric monoidal closed category, or dualisable objects, are exactly the nuclear objects, as argued in Section 6. The statement can be further generalised: if `mix` is not invertible but the underlying $*$-autonomous category has some nice factorization system, then the image of `mix` is the support of a Frobenius structure. In SLatt, this construction yields the Girard quantale of tight endomaps of a complete lattice studied in [13]. The generalization is a consequence of a double negation construction that we described in [13] for Frobenius quantales, and that we generalise here to a categorical setting. The statement sounds as follows:

▶ **Theorem B.** *Let $\mathcal{V}$ be a $*$-autonomous category with an epi-mono factorization system. Let (A, μ_A) be a semigroup in $\mathcal{V}$ and let $\epsilon : A \otimes B \longrightarrow I$ be a dual pairing. If $f : A \longrightarrow B$ is a map such that $\epsilon \circ (A \otimes f) = \epsilon \circ (A \otimes f) \circ \sigma_{A,A}$, then the image of f is the carrier of a Frobenius structure.*

We also demonstrate that the converse of Theorem A actually holds if we add another condition, which we identify now as a key ingredient of the proof in [34] of Theorem 1. We say that an objet A of a monoidal category is *pseudo-affine* if the tensor unit I embeds into A as a retract. For example, every complete lattice which is not a singleton is pseudo-affine in SLatt. We prove:

▶ **Theorem C.** *If A is a pseudo-affine object of a $*$-autonomous category and the canonical monoid $([A, A], \circ)$ is part of a Frobenius structure, then A is nuclear.*

Related Work. Besides the works that we already mentioned, either those on the theory of quantales [26, 14, 16] or those generalizing the notion of Frobenius quantale to some kind of monoidal setting [25, 23, 39, 15], we also wish to mention that the notion of nuclearity, originally conceived for Banach spaces [21, 32], has been by now generalised in several directions [1, 7]. For example, this notion is generalised in [1] to the category of Hilbert spaces, with motivations from quantum mechanics. While our initial motivations for developing this research were of a purely logical and order-theoretic nature, we could recognize, via the formalization in a categorical language and the notion of nuclearity, the similarity of our questionings with those arising in this line of research. In particular, we could construct in [13] a unitless Frobenius quantale from trace class operators (i.e., nuclear

endomaps) of an infinite dimensional Hilbert space. We noticed that the questioning about unitless structures is pervasive in this line of research [2, 18, 12]. We describe in the last section our first non-conclusive remarks on the scope of our results within this family of monoidal categories. We are convinced that connecting with existing research on nuclearity and categorical quantum mechanics is a research direction worth to be fully explored.

Structure of the paper. After introducing elementary notions in Section 2, we study the notion of dual pairing in Section 3 and give an overview of actions in dual pairs in Section 4. This makes it possible to introduce Frobenius structures in Section 5. We then state in Section 6 the equivalence between the notion of adjoint and that of nuclear object in autonomous categories. We finally prove Theorems A and B in Section 7, and Theorem C in Section 8. We add a discussion of the results obtained and sketch future research in Section 9. For lack of space, proofs of the statements that might be easy to derive or be considered part of the folklore only appear in the preprint [37].

2 Background

Let us recall that a *quantale* is a semigroup in the $*$-autonomous category SLatt, see [16]. Otherwise said, it is a pair $(Q, *)$ with Q a complete lattice and $*$ a semigroup operation which distributes in each place with suprema. An essential feature of a quantale $(Q, *)$ are the two implication maps $(-\backslash -) : Q \otimes Q^{\mathrm{op}} \longrightarrow Q^{\mathrm{op}}$ and $(-/-) : Q^{\mathrm{op}} \otimes Q \longrightarrow Q^{\mathrm{op}}$ satisfying the adjointness relations

$$x * y \leq z \text{ iff } x \leq z/y \text{ iff } y \leq x\backslash z\,.$$

We rely on the standard monograph [27] for elementary facts and notation concerning categories. We shall say that a category is *autonomous* if it is symmetric monoidal closed. Thus, an autonomous category is of the form $\mathcal{V} = (\mathsf{V}, I, \otimes, \alpha, \lambda, \rho, \sigma, [-,-], ev)$, where V is a category, $\otimes : \mathsf{V} \times \mathsf{V} \longrightarrow \mathsf{V}$ is a bifunctor, $[X, -] : \mathsf{V} \longrightarrow \mathsf{V}$ is the right adjoint to $(X \otimes -)$, ev is the counit of this adjunction and $\alpha_{X,Y,Z} : (X \otimes Y) \otimes Z \cong X \otimes (Y \otimes Z)$, $\rho_X : X \otimes I \cong X$, $\lambda_X : I \otimes X \cong X$, $\sigma_{X,Y} : X \otimes Y \cong Y \otimes X$ are all natural isomorphisms satisfying well-known constraints. We shall work with monoidal categories that are partially strict, by which we mean that the associator α is the identity. If $\mathcal{V}$ is autonomous and 0 is a fixed object, then we get a contravariant functor $(-)^* := [-, 0]$. We define the map $j_X : X \longrightarrow X^{**}$ as the transpose of $ev_{X,0} \circ \sigma_{X^*,X}$. We say an object is *reflexive* if j_X is an isomorphism. If every object is reflexive, then 0 is said to be dualizing and $\mathcal{V}$ is said to be a *$*$-autonomous category.* If $0 = I$ then, we can define the morphism $\mathtt{mix}_{X,Y} : Y^* \otimes X \longrightarrow [Y, X]$ as the transpose of $\lambda_X \circ (ev_{Y,I} \otimes \mathtt{id}_X)$. An object X is *nuclear* if $\mathtt{mix}_{X,X}$ is invertible. A nuclear object is necessarily reflexive.

▶ **Definition 2.** *A* magma *in a monoidal category $\mathcal{V}$ is a pair (A, μ_A) with $\mu_A : A \otimes A \longrightarrow A$ a morphism in $\mathcal{V}$. A* bracketed magma *in $\mathcal{V}$ is a triple (A, μ_A, π_A) with (A, μ_A) a magma and $\pi_A : A \otimes A \longrightarrow I$. A bracketed magma is* associative *if (A, μ_A) is a semigroup in $\mathcal{V}$ (that is, if μ_A is associative) and π_A is associative, meaning that the following two horizontal arrows are equal:*

$$A \otimes A \otimes A \;\underset{A\otimes\mu_A}{\overset{\mu_A\otimes A}{\rightrightarrows}}\; A \otimes A \xrightarrow{\pi_A} I\,.$$

If $\mathcal{V}$ is symmetric monoidal and (A, μ_A) is a magma, a pairing π_A is said to be co-associative *if $(A, \mu_A \circ \sigma_{A,A}, \pi_A)$ is an associative bracketed magma.*

▶ **Fact 3.** A pairing π_A is associative if and only if the pairing $\pi_A \circ \sigma_{A,A}$ is co-associative.

Notice that if (A, μ_A) is a semigroup and $\mathrm{tr} : A \longrightarrow I$ is any arrow, then $\mathrm{tr} \circ \mu_A : A \otimes A \longrightarrow I$ is associative.

▶ **Example 4.** In every autonomous category, $([A, A], \circ)$ is a well-known semigroup where $\circ$ is the internal composition defined as the transpose of $ev_{A,A} \circ (ev_{A,A} \otimes [A, A])$. Also, if $0 = I$, then $(A^* \otimes A, \mu_{A^* \otimes A}, \pi_{A^* \otimes A})$ is an associative bracketed magma with $\mu_{A^* \otimes A} := (A^* \otimes \lambda_A) \circ (A^* \otimes ev_{A,I} \otimes A)$ and $\pi_{A^* \otimes A} := ev_{A,I} \circ \sigma_{A^*,A} \circ \mu_{A^* \otimes A}$, see e.g. [26].

▶ **Definition 5.** *Let (A, μ_A) be magma. A map $\alpha^\ell : A \otimes X \longrightarrow X$ is a* left action *if the two parallel arrows*

$$A \otimes A \otimes X \underset{A \otimes \alpha^\ell}{\overset{\mu_A \otimes X}{\rightrightarrows}} A \otimes X \xrightarrow{\alpha^\ell} X$$

are equal. A map $\alpha^\rho : X \otimes A \longrightarrow X$ is defined to be a right action *in a similar way.*

▶ **Definition 6.** *Let (A, μ_A), (B, μ_B) be two magmas. A* magma homomorphism *from (A, μ_A) to (B, μ_B) is a map $f : A \longrightarrow B$ making the diagram below on the left commutative. If (A, μ_A, π_A) and (B, μ_B, π_B) are bracketed magmas, a* bracketed magma homomorphism *from (A, μ_A, π_A) to (B, μ_B, π_B) is a magma homomorphism $f : (A, \mu_A) \longrightarrow (B, \mu_B)$ making the diagram below in the middle commutative.*

$$\begin{array}{ccc} A \otimes A & \xrightarrow{f \otimes f} & B \otimes B \\ \downarrow{\scriptstyle \mu_A} & & \downarrow{\scriptstyle \mu_B} \\ A & \xrightarrow{f} & B \end{array} \qquad \begin{array}{ccc} A \otimes A & \xrightarrow{f \otimes f} & B \otimes B \\ & \searrow^{\pi_A} & \downarrow{\scriptstyle \pi_B} \\ & & I \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{\pi_A^\sharp} & A^* \\ \downarrow{\scriptstyle f} & & \uparrow{\scriptstyle f^*} \\ B & \xrightarrow{\pi_B^\sharp} & B^* \end{array}$$

Magma homomorphisms between semigroups are called semigroup homomorphisms*.*

Let us notice that, when the environment category is autonomous, the diagram above in the middle commutes if and only if the diagram on the right does.

▶ **Fact 7.** The arrow $\mathtt{mix} : A^* \otimes A \longrightarrow [A, A]$ is a semigroup homomorphism.

▶ **Lemma 8.** *Let $\mathcal{V}$ be autonomous and let $e : (A, \mu_A, \pi_A) \longrightarrow (B, \mu_B, \pi_B)$ be an epimorphism of bracketed magmas. If (A, μ_A, π_A) is associative, then so is (B, μ_B, π_B).*

3 Dual pairings

The goal of this section is to introduce the technical notion of dual pairing, needed later to define Frobenius structures. We also present a few properties of dual pairings. Throughout the section we let 0 be a fixed object of an environment monoidal category $\mathcal{V}$.

▶ **Definition 9.** *A map $\epsilon : A \otimes B \longrightarrow 0$ in $\mathcal{V}$ is said to be a* dual pairing *(w.r.t. the object 0) if the induced natural transformations*

$$\hom(X, B) \longrightarrow \hom(A \otimes X, 0)\,, \qquad \hom(X, A) \longrightarrow \hom(X \otimes B, 0)\,,$$

are isomorphims.

Clearly, the above definition is equivalent to requiring that ϵ has two universal properties: 1. *for each* $f : A \otimes X \longrightarrow 0$ *there exists a unique map* $f^\sharp : X \longrightarrow B$ *such that* $\epsilon \circ (A \otimes f^\sharp) = f$, and 2. *for each* $g : X \otimes B \longrightarrow 0$ *there exists a unique map* $g^\sharp : X \longrightarrow A$ *such that* $\epsilon \circ (g^\sharp \otimes B) = g$. We call $f^\sharp$ (and $g^\sharp$) the *transpose* of f (resp. g). For $f : X \longrightarrow B$ (resp. $g : X \longrightarrow A$), we let $f^\flat = \epsilon \circ (A \otimes f)$ (resp. $g^\flat = \epsilon \circ (g \otimes B)$). If $\epsilon : A \otimes B \longrightarrow 0$ is a dual pairing, then we shall also say that the triple (A, B, ϵ) is a dual pairing, thus emphasizing the typing. We shall also informally say that (A, B) is a *dual pair*, leaving aside the arrow ϵ. A dual pairing $\epsilon : A \otimes B \longrightarrow 0$ is unique up to unique isomorphism, as stated next.

▶ **Proposition 10.** *If* $(A, B_1, \epsilon_{A,B_1})$ *is a dual pairing and* $\psi : B_0 \longrightarrow B_1$ *is iso, then* $(A, B_0, \epsilon_{A,B_1} \circ (A \otimes \psi))$ *is a dual pairing. Conversely, if* $(A, B_0, \epsilon_{A,B_0})$ *and* $(A, B_1, \epsilon_{A,B_1})$ *are two dual pairings, then there exists a unique iso* $\psi : B_0 \longrightarrow B_1$ *such that* $\epsilon_{A,B_0} = \epsilon_{A,B_1} \circ (A \otimes \psi)$.

Obviously, similar statements also hold with respect to the object on the left, for example: if $(A_0, B, \epsilon_{A_0,B})$ and $(A_1, B, \epsilon_{A_1,B})$ are two dual pairings, then there exists a unique iso $\chi : A_0 \longrightarrow A_1$ such that $\epsilon_{A_0,B} = \epsilon_{A_1,B} \circ (\chi \otimes B)$. While our presentation of the notion of dual pairing is symmetry-free, we shall work later within symmetric and autonomous categories. This imposes a few remarks.

▶ **Lemma 11.** *If* $\mathcal{V}$ *is symmetric and* (A, B, ϵ) *is a dual pairing, then so is* $(B, A, \epsilon \circ \sigma_{B,A})$.

For $\mathcal{V}$ an autonomous category and an arrow $f : X \longrightarrow Y^*$ in $\mathcal{V}$, we let $f^\perp : Y \longrightarrow X^*$ be the transpose of the map $X \otimes Y \xrightarrow{\sigma_{X,Y}} Y \otimes X \xrightarrow{Y \otimes f} Y \otimes Y^* \xrightarrow{ev_{Y,0}} 0$.

Notice that $(-)^\perp : \hom(X, Y^*) \longrightarrow \hom(Y, X^*)$ is natural in X and Y. If $g = f^\perp$, then we say that f and g are *mates*. Let us remark that the canonical arrow $j_X : X \longrightarrow X^{**}$ is the mate of $\mathtt{id}_{X^*}$, from which the following statement easily follows:

▶ **Lemma 12.** *For* $f : X \longrightarrow Y^*$, $f^\perp = f^* \circ j_Y$.

Mates allow to precisely characterize dual pairs in autonomous categories, as follows.

▶ **Proposition 13.** *If* $\mathcal{V}$ *is autonomous, then*

1. (A, B) *is a dual pair if and only if there are invertible mates* $\phi : A \to B^*$ *and* $\psi : B \to A^*$.
2. *If* (A, B) *is a dual pair, then* (A^*, B^*) *is also a dual pair.*
3. *If* (A, B) *is a dual pair, then both* A *and* B *are reflexive objects of* $\mathcal{V}$.
4. *If* A *is reflexive, then* $(A, A^*, ev_{A,0})$ *is a dual pairing.*
5. (A, B) *is a dual pair if and only if* A *is reflexive and there is some iso* $\psi : B \longrightarrow A^*$.

We next give some examples where the notion of dual pair naturally arises.

▶ **Example 14.** Proposition 13.5 shows that in a $*$-autonomous category $\mathcal{V}$, where all the objects are reflexive, any isomorphism $B \longrightarrow A^*$ is enough to build a dual pair (A, B). This is a key observation for the coherence theorem in [9]. For example, in SLatt, taking as object 0 the two-element Boolean algebra $\mathbf{2} := \{\bot, \top\}$ (which is also the unit of the tensor), the canonical isomorphism $L^{\mathrm{op}} \simeq L^*$ – which associates to every $y \in L$ the map $a_y : L \longrightarrow \mathbf{2}$ defined by $a_y(x) = \bot$ if and only if $x \leq y$ – yields the dual pair (L, L^{op}). The pairing $\epsilon : L \otimes L^{\mathrm{op}} \longrightarrow \mathbf{2}$ is given by $\epsilon(x, y) = \bot$, if $x \leq y$, and $\epsilon(x, y) = \top$, otherwise.

▶ **Example 15.** For any object L of a $*$-autonomous category $\mathcal{V}$, the pair $(L^* \otimes L, [L, L])$ is dual with pairing given by $\epsilon := ev_{L,0} \circ \sigma_{L^*,L} \circ (L^* \otimes ev_{L,L})$. Indeed, every object of $\mathcal{V}$ is reflexive, so $(L^* \otimes L, (L^* \otimes L)^*, ev_{L^* \otimes L,0})$ is a dual pairing by Proposition 13.4. It is well

known that in a $*$-autonomous category the transpose of $ev_{L,0} \circ \sigma_{L^*,L} \circ (L^* \otimes ev_{L,L})$ is an isomorphism $\psi : [L, L] \longrightarrow (L^* \otimes L)^*$. Then, by Proposition 10, $ev_{L^* \otimes L,0} \circ (L^* \otimes L \otimes \psi) = ev_{L,0} \circ \sigma_{L^*,L} \circ (L^* \otimes ev_{L,L})$ is a dual pairing.

▶ **Example 16.** It is also possible to understand dual pairs as a generalization of the well-known notion of *adjunction*. Recall that a tuple (A, B, ϵ, η) with $\epsilon : A \otimes B \longrightarrow I$ and $\eta : I \longrightarrow B \otimes A$ is said to be an adjunction if the two diagrams below commute.

$$\begin{array}{ccc} A \otimes B \otimes A & \xleftarrow{A \otimes \eta} & A \otimes I \\ \downarrow{\scriptstyle \epsilon \otimes A} & & \downarrow{\scriptstyle \rho_A} \\ I \otimes A & \xrightarrow{\lambda_A} & A \end{array} \qquad \begin{array}{ccc} I \otimes B & \xrightarrow{\eta \otimes B} & B \otimes A \otimes B \\ \downarrow{\scriptstyle \lambda_B} & & \downarrow{\scriptstyle B \otimes \epsilon} \\ B & \xleftarrow{\rho_B} & B \otimes I \end{array}$$

A is said to be left adjoint to B, B right adjoint to A, η is the unit of the adjunction and ϵ is the counit of the adjunction. In an autonomous category, if (A, B, η, ϵ) is an adjunction, then so is $(B, A, \sigma_{A,B} \circ \eta, \epsilon \circ \sigma_{B,A})$, and therefore we won't distinguish between left and right adjoints, we just say that A (or B) is part of an adjunction.[1] It is a standard exercise in monoidal categories to verify the following statement.

▶ **Fact 17.** If (A, B, η, ϵ) is an adjunction, then (A, B, ϵ) is a dual pairing.

Not all the dual pairs arise from adjunctions. This can be simply observed by looking at the $*$-autonomous category SLatt, where an object is part of an adjunction if and only if it is a nuclear object, by the content of Section 6, if and only it is completely distributive, by [29, 22]. Yet, (L, L^{op}, ϵ), described in Example 14, is always a dual pairing, even when L is not completely distributive.

In [10], the authors introduce the notion of *linear adjunction*. Recall that a linearly distributive category, see e.g. [11], is a category with distinct monoidal structures $(I, \otimes)$ and $(0, \oplus)$ coming with natural maps $\kappa^\lambda : X \otimes (Y \oplus Z) \longrightarrow (X \otimes Y) \oplus X$ and $\kappa^\rho : (X \oplus Y) \otimes Z \longrightarrow X \oplus (Y \otimes Z)$ satisfying some constraints. A linear adjunction is then a tuple (A, B, ϵ, η) with $\epsilon : A \otimes B \longrightarrow 0$ and $\eta : I \longrightarrow B \oplus A$ making the diagrams below commute.

$$\begin{array}{ccc} A \otimes (B \oplus A) & \xleftarrow{A \otimes \eta} & A \otimes I \\ \downarrow{\scriptstyle \kappa^\lambda_{A,B,A}} & & \\ (A \otimes B) \oplus A & & \downarrow{\scriptstyle \rho^\otimes_A} \\ \downarrow{\scriptstyle \epsilon \oplus A} & & \\ 0 \oplus A & \xrightarrow{\lambda^\oplus_A} & A \end{array} \qquad \begin{array}{ccc} I \otimes B & \xrightarrow{\eta \otimes B} & (B \oplus A) \otimes B \\ & & \downarrow{\scriptstyle \kappa^\rho_{B,A,B}} \\ \downarrow{\scriptstyle \lambda^\otimes_B} & & B \oplus (A \otimes B) \\ & & \downarrow{\scriptstyle B \oplus \epsilon} \\ B & \xleftarrow{\rho^\oplus_B} & B \oplus 0 \end{array}$$

A $*$-autonomous category is, canonically, a linearly distributive category when we let $X \oplus Y := (Y^* \otimes X^*)^*$. In view of Example 14 and Theorem 1.2 in [15], if the environment category is $*$-autonomous, then (A, B, ϵ) is a dual pairing if and only if we can find $\eta : I \longrightarrow A \oplus B$ such that (A, B, η, ϵ) is a linear adjunction.

More on mates, adjoints. In SLatt, for every sup-preserving map $f : L \longrightarrow M$ between complete lattices, there exists a unique sup-preserving map $\rho(f) : M^{\mathrm{op}} \longrightarrow L^{\mathrm{op}}$ such that, for each $x \in L$ and $y \in M$, $f(x) \leq y$ iff $x \leq \rho(f)(y)$. The map $\rho(f)$ is called the (right)

[1] This also to avoid the naming conflict with the notion of adjoint, presented below in this section. An object A which is part of an adjunction is often called *dualizable*.

adjoint of f. The equivalences defining the adjoint can be expressed diagrammatically, by means of the equation $\epsilon_{M,M^{\mathrm{op}}} \circ (f \otimes M^{\mathrm{op}}) = \epsilon_{L,L^{\mathrm{op}}} \circ (L \otimes \rho(f))$. This suggests that a similar notion can be defined for dual pairings in an arbitrary monoidal category.

Let (A_0, B_0, ϵ_0) and (A_1, B_1, ϵ_1) be two dual pairings. For a morphism $f : A_0 \longrightarrow A_1$, we denote by $\widetilde{f} : B_1 \longrightarrow B_0$ the arrow given by the two transposes below:

$$\dfrac{\dfrac{A_0 \longrightarrow A_1}{A_0 \otimes B_1 \longrightarrow 0}}{B_1 \longrightarrow B_0}$$

We say that $f : A_0 \longrightarrow A_1$ is *adjoint* to $g : B_1 \longrightarrow B_0$ if $\widetilde{f} = g$.

A dual pairing (A, B, ϵ) is a special object of the category $\mathrm{Chu}_{\mathcal{V},0}$, see [6]. Recall that a morphism between two objects (X_0, X_1, p_X) and (Y_0, Y_1, p_Y) of $\mathrm{Chu}_{\mathcal{V},0}$ is a pair of $\mathcal{V}$ morphisms $(f : X_0 \longrightarrow Y_0, g : Y_1 \longrightarrow X_1)$ such that the square below commutes.

$$\begin{array}{ccc} X_0 \otimes Y_1 & \xrightarrow{f \otimes Y_1} & Y_0 \otimes Y_1 \\ \downarrow{\scriptstyle X_0 \otimes g} & & \downarrow{\scriptstyle p_Y} \\ X_0 \otimes X_1 & \xrightarrow{p_X} & 0\,. \end{array}$$

If (A_0, B_0, ϵ_0) and (A_1, B_1, ϵ_1) are dual pairings, then a $\mathrm{Chu}_{\mathcal{V},0}$ morphism between them is necessarily of the form $(f, \widetilde{f})$ and, also, a pair $(f, \widetilde{f})$ is obviously a morphism in $\mathrm{Chu}_{\mathcal{V},0}$. Denoting by $\mathrm{Pair}_{\mathcal{V},0}$ the full subcategory of $\mathrm{Chu}_{\mathcal{V},0}$ whose objects are dual pairings, these remarks amount to the following statement, whose most relevant consequences is presented immediately after.

▶ **Lemma 18.** *The obvious projection functors, $\mathcal{P}_0 : \mathrm{Pair}_{\mathcal{V},0} \longrightarrow \mathcal{V}$ and $\mathcal{P}_1 : \mathrm{Pair}_{\mathcal{V},0} \longrightarrow \mathcal{V}^{\mathrm{op}}$, are full and faithful.*

▶ **Corollary 19.** *Let (A_i, B_i, ϵ_i), $i = 0, 1, 2$, be dual pairings. If $f : A_0 \longrightarrow A_1$ and $g : A_1 \longrightarrow A_2$, then $\widetilde{\mathtt{id}_{A_i}} = \mathtt{id}_{B_i}$ and $\widetilde{g} \circ \widetilde{f} = \widetilde{(f \circ g)}$. In particular, if $f : A_0 \longrightarrow A_1$ is inverted by $g : A_1 \longrightarrow A_0$, then $\widetilde{f}$ is inverted by $\widetilde{g}$.*

Let X, Y be reflexive, so $(X, X^*, ev_{X,0})$, $(Y, Y^*_{Y,0})$, and $(Y^*, Y, ev_{Y,0} \circ \sigma_{Y^*,Y})$ are dual pairings. Notice the following relations: for $f : X \longrightarrow Y$, $\widetilde{f} = f^*$, that is, the adjoint of f is the result of the functorial action on the map f; for $g : X \longrightarrow Y^*$, $\widetilde{g} = g^{\perp}$, that is, the adjoint of g is just its mate. This yields a number of relations. For example, if also Z is reflexive, $g : Z \longrightarrow Y$ and $f : X \longrightarrow Y^*$, then using Corollary 19 we have

$$(g^* \circ f)^{\perp} = f^{\perp} \circ g : Z \longrightarrow X^*\,. \tag{2}$$

Notice that equation (2) amounts to the naturality of $(-)^{\perp} : \hom(X, Y^*) \longrightarrow \hom(Y, X^*)$.

Finally, let us mention that we shall focus on maps $f : A \longrightarrow B$, where (A, B, ϵ) is a dual pairing and, consequently, $(B, A, \epsilon \circ \sigma_{B,A})$ as well. The following statement is then easily verified.

▶ **Lemma 20.** *For (A, B, ϵ) a dual pairing and $f : A \longrightarrow B$, the transposes of f and $\widetilde{f}$ differ by a symmetry: $\epsilon \circ (A \otimes \widetilde{f}) = \epsilon \circ (A \otimes f) \circ \sigma_{A,A}$.*

4 Actions in dual pairs

The implications of a quantale validate the two equations

$$z/(y * x) = (z/y)/x \qquad \text{and} \qquad (x * y)\backslash z = x\backslash(y\backslash z)\,.$$

These two equations can be understood by saying that the implications are, respectively, left and right actions of Q over Q^{op}, see Example 22. In this section we show that actions arise from dual pairs when one of the objects of a dual pair is a semigroup. From now on, $\mathcal{V}$ will be a symmetric monoidal category. Let (A, B, ϵ) be a dual pairing in $\mathcal{V}$ such that (A, μ_A) is a magma. We define two morphisms

$$\alpha^{\ell}_A : A \otimes B \longrightarrow B \qquad \text{and} \qquad \alpha^{\rho}_A : B \otimes A \longrightarrow B\,,$$

as the only morphisms making these two diagrams commute:

$$\begin{array}{ccc} A \otimes A \otimes B & \xrightarrow{A\otimes\alpha^{\ell}_A} & A \otimes B \\ \downarrow{\scriptstyle \mu_A\otimes B} & & \downarrow{\scriptstyle \epsilon} \\ A \otimes B & \xrightarrow{\epsilon} & 0 \end{array} \qquad \begin{array}{ccccc} B \otimes A \otimes A & \xrightarrow{B\otimes\mu_A} & B \otimes A & \xrightarrow{\sigma_{B,A}} & A \otimes B \\ \downarrow{\scriptstyle \alpha^{\rho}_A\otimes A} & & & & \downarrow{\scriptstyle \epsilon} \\ B \otimes A & & \xrightarrow{\sigma_{B,A}} & A \otimes B & \xrightarrow{\epsilon} 0 \end{array} \tag{3}$$

▶ **Fact 21.** Letting A^{op} be the magma $(A, \mu_A \circ \sigma_{A,A})$, the relation $\alpha^{\ell}_{A^{op}} = \alpha^{\rho}_A \circ \sigma_{A,B}$ holds.

▶ **Example 22.** As mentioned in Example 14, (Q, Q^{op}) is a dual pair in SLatt. If $(Q, *)$ is a quantale, commutativity of the left (resp. right) diagram in (3) amounts to the relation

$$x * y \leq z \;\text{ iff }\; x \leq \alpha^{\ell}_Q(y, z) \qquad (\text{resp.,}\quad x * y \leq z \;\text{ iff }\; y \leq \alpha^{\rho}_Q(z, x))\,.$$

Therefore, by the uniqueness of the adjoint, we have $\alpha^{\ell}_Q(y, z) = z/y$ and $\alpha^{\rho}_Q(z, x) = x\backslash z$.

▶ **Example 23.** Let A be a finite dimensional k-algebra, $x, y \in A$, $l \in A^* = [A, k]$. It is direct to verify that $\alpha^{\ell}_A(y, l) : x \mapsto l(xy)$ and $\alpha^{\rho}_A(l, x) : y \mapsto l(xy)$.

Let us give some elementary properties of α^{ℓ}_A and α^{ρ}_A.

▶ **Lemma 24.** *Let (A, B, ϵ) be a dual pairing and (A, μ_A) be a magma.*

1. *The following equation, relating α^{ℓ}_A and α^{ρ}_A, holds:*

$$\epsilon \circ (A \otimes \alpha^{\rho}_A) = \epsilon \circ \sigma_{B,A} \circ (\alpha^{\ell}_A \otimes A)\,. \tag{4}$$

2. *If (A, μ_A) is a semigroup, then the map α^{ℓ}_A is a left action (see Definition 5) and the map α^{ρ}_A is a right action.*
3. *If (A, μ_A) is a semigroup, then the two actions are equivariant to each other, that is*

$$\alpha^{\ell}_A \circ (A \otimes \alpha^{\rho}_A) = \alpha^{\rho}_A \circ (\alpha^{\ell}_A \otimes A)\,.$$

As the dual pair $([A, A], A^* \otimes A)$ is the main object studied in this paper, we characterize next the actions of $([A, A], \circ)$ over $A^* \otimes A$, and of $(A^* \otimes A, \mu_{A^*\otimes A})$ over $[A, A]$, see Examples 4 and 15.

▶ **Proposition 25.** *In a $*$-autonomous category $\mathcal{V}$, we have*

$$\alpha^{\rho}_{[A,A]} = A^* \otimes ev_{A,A}\,, \qquad \alpha^{\ell}_{[A,A]} = \mu_{A,A,0} \otimes A\,.$$

5 Frobenius structures

We finally define in this section the notion of Frobenius structure in a symmetric monoidal category $\mathcal{V}$ and give some elementary properties of this structure. While other approaches are possible, notably [39, 15], they are not strictly equivalent, starting from the assumptions on the environment category.

▶ **Definition 26.** *A* Frobenius structure *is a tuple* $(A, B, \epsilon, \mu_A, l, r)$ *where* (A, B, ϵ) *is a dual pairing,* (A, μ_A) *is a semigroup, and* l *and* r *are adjoint invertible maps from* A *to* B *such that diagram* (5) *commutes.*

$$\begin{array}{ccc} A \otimes A & \xrightarrow{A\otimes r} & A \otimes B \\ {\scriptstyle l\otimes A}\downarrow & & \downarrow{\scriptstyle \alpha^{\ell}_{A}} \\ B \otimes A & \xrightarrow[\alpha^{\rho}_{A}]{} & B. \end{array} \tag{5}$$

Notice that, by Lemma 20, the above definition coincides with the one given in the Introduction.

▶ **Example 27.** Definition 26 strictly mimics the definition of Frobenius quantales in [13]. A Frobenius quantale is defined there as a tuple $(Q, *, {}^{\perp}(-), (-)^{\perp})$ such that $(Q, *)$ is a quantale, and where $({}^{\perp}(-), (-)^{\perp})$ form an invertible Galois connection satisfying the equation

$$y \backslash^{\perp} x = y^{\perp} / x\,. \tag{6}$$

The commutative diagram (5) is just a direct translation of equation (6), called the contraposition law in [17].

Let us recall a few considerations developed in [13]. If a quantale has a dualizing element 0, then the two maps ${}^{\perp}(-) := 0/(-)$ and $(-)^{\perp} := (-)\backslash 0$ are inverse to each other, form a Galois connection, and satisfy equation (6). And if a Frobenius quantale has a unit 1 then $0 := {}^{\perp}1 = 1^{\perp}$ is a dualizing element. That is, contrarily to the usual definition (see for instance [40, 31, 26, 16]), a Frobenius quantale does not need a unit.[2] Now it is immediate to see that Frobenius quantales as defined in [13] are exactly the Frobenius structures $(A, B, \epsilon, \mu, l, r)$ in SLatt for which $B = A^{\mathrm{op}}$.

Before giving other examples of Frobenius structures, let us give some characterization of these structures. In a quantale, equation (6) is equivalent to the shift/associative relations:

$$x * y \leq {}^{\perp}z \quad \text{iff} \quad x^{\perp} \geq y * z \quad \text{iff} \quad x \leq {}^{\perp}(y * z)\,.$$

This also holds in the general setting, simply by transposing diagram (5) we have:

▶ **Lemma 28.** *A pair of morphisms* (l, r) *makes diagram* (5) *commute if and only if the shift diagram* (7) *commutes.*

$$\begin{array}{ccccc} A \otimes A \otimes A & & \xrightarrow{\mu_A \otimes l} & & A \otimes B \\ \downarrow{\scriptstyle r\otimes \mu_A} & & & & \downarrow{\scriptstyle \epsilon} \\ B \otimes A & \xrightarrow{\sigma_{B,A}} & A \otimes B & \xrightarrow{\epsilon} & 0 \end{array} \tag{7}$$

[2] As a matter of fact, the theorems that we shall present in the next two sections can be understood as characterizing the existence of units in specific cases.

Our next goal is to show that, for a Frobenius structure $(A, B, \epsilon, \mu_A, l, r)$, the dual pair (B, A) also carries a Frobenius structure, in particular B carries a canonical semigroup structure.

▶ **Lemma 29.** *Let (A, B, ϵ) be a dual pair, (A, μ_A) a semigroup, and (l, r) invertible adjoint maps from A to B. The tuple $(A, B, \epsilon, \mu_A, l, r)$ is a Frobenius structure if and only if diagram (8) commutes.*

$$\begin{array}{ccc} B \otimes B & \xrightarrow{B\otimes r^{-1}} & B \otimes A \\ {\scriptstyle l^{-1}\otimes B}\downarrow & \overset{\mu_B}{\searrow} & \downarrow{\scriptstyle \alpha^\rho_A} \\ A \otimes B & \xrightarrow[\alpha^\ell_A]{} & B \end{array} \tag{8}$$

For a Frobenius structure $(A, B, \epsilon, \mu_A, l, r)$, let us denote by μ_B the diagonal of diagram (8). We next consider the magma (B, μ_B). Note that from the definition of μ_B and of the actions of a magma over its dual, we obtain two pairs of parallel equal arrows

$$B \otimes B \otimes A \underset{\mu_B\otimes A}{\overset{B\otimes\alpha^\ell_B}{\rightrightarrows}} B \otimes A \otimes A \underset{\alpha^\rho_A\otimes A}{\overset{B\otimes\mu_A}{\rightrightarrows}} B \otimes A \xrightarrow{\sigma_{B,A}} A \otimes B \xrightarrow{\epsilon} 0\,,$$

$$A \otimes B \otimes B \underset{A\otimes\mu_B}{\overset{\alpha^\rho_B\otimes B}{\rightrightarrows}} A \otimes A \otimes B \underset{A\otimes\alpha^\ell_A}{\overset{\mu_A\otimes B}{\rightrightarrows}} A \otimes B \xrightarrow{\epsilon} 0\,,$$

showing that the actions of the magma (B, μ_B) on the dual A depends on μ_A by:

$$\alpha^\ell_B = \mu_A \circ (r^{-1} \otimes A)\,, \qquad \alpha^\rho_B = \mu_A \circ (A \otimes l^{-1})\,, \tag{9}$$

and that μ_A is obtained from the action of (B, μ_B) over A by:

$$\begin{array}{ccc} A \otimes A & \xrightarrow{A\otimes l} & A \otimes B \\ {\scriptstyle r\otimes A}\downarrow & \overset{\mu_A}{\searrow} & \downarrow{\scriptstyle \alpha^\rho_B} \\ B \otimes A & \xrightarrow[\alpha^\ell_B]{} & A\,. \end{array} \tag{10}$$

We collect the properties of Frobenius structures that we want to emphasize next.

▶ **Proposition 30.** *For a Frobenius structure $(A, B, \epsilon, \mu_A, l, r)$, the following statements hold:*

1. *$(A, B, \epsilon, \mu_A \circ \sigma_{A,A}, r, l)$ is a Frobenius structure.*
2. *The magma (B, μ_B) is a semigroup.*
3. *The following diagrams commute:*

$$\begin{array}{ccc} A \otimes A & \xrightarrow{\mu_A} & A \\ \downarrow{\scriptstyle A\otimes l} & & \downarrow{\scriptstyle l} \\ A \otimes B & \xrightarrow{\alpha^\ell_A} & B \end{array} \qquad \begin{array}{ccc} A \otimes A & \xrightarrow{\mu_A} & A \\ \downarrow{\scriptstyle r\otimes A} & & \downarrow{\scriptstyle r} \\ A \otimes B & \xrightarrow{\alpha^\rho_A} & B \end{array} \tag{11}$$

4. *The maps l and r are both semigroup homomorphisms from (A, μ_A) to (B, μ_B).*
5. *The tuple $(B, A, \epsilon \circ \sigma_A, \mu_B, r^{-1}, l^{-1})$ is a Frobenius structure.*

The last item of the Proposition exhibits the expected duality. Indeed, using diagram (10), it is easily seen that the dual of $(B, A, \epsilon \circ \sigma_{B,A}, \mu_B, r^{-1}, l^{-1})$ is $(A, B, \epsilon, \mu_A, l, r)$ itself. Notice that having such duality is not an obvious result, since our definition of Frobenius structure is not self dual. *E.g.*, the dual B of A is not required to be a semigroup itself and the requirement that diagram (5) commutes is a condition on the semigroup (A, μ_A).

Finally, let us establish the connection with associative bracketed magmas.

▶ **Proposition 31.** *Let $(A, B, \epsilon, \mu_A, l, r)$ be a Frobenius structure and define $\pi^l_A := \epsilon \circ (A \otimes l)$ and $\pi^r_A := \epsilon \circ (A \otimes r)$. Then (A, μ_A, π^l_A) is an associative bracketed magma, (A, μ_A, π^r_A) is a co-associative bracketed magma and π^l_A and π^r_A are dual pairings. Conversely, given an associative bracketed magma (A, μ_A, π_A) for which π_A is a dual pairing, and given another dual paring (A, B, ϵ), define l so that $\epsilon \circ (A \otimes l) = \pi_A$, $r = \widetilde{l}$, then $(A, B, \epsilon, \mu_A, l, r)$ is a Frobenius structure.*

From this correspondence between Frobenius structures and associative bracketed magmas, it is easy to find some examples from the literature.

▶ **Example 32.** A Frobenius algebra is a finite-dimensional k-algebra (A, μ_A) coming with a non degenerate form $\langle -, - \rangle$ such that $\langle xy, z \rangle = \langle x, yz \rangle$. That is, it is a bracketed semigroup $(A, \mu_A, \langle -, - \rangle)$ and $\langle -, - \rangle$ is a dual pairing as its transpose is an isomorphism between A and A^*. Hence, $(A, A^*, ev, \mu_A, r, l)$ is a Frobenius structure in the category of finite-dimensional k-vector spaces, where l and r are defined as stated in Proposition 31.

▶ **Example 33.** A Frobenius algebra (A, t, δ, e, μ) in a symmetric monoidal category, as defined in [23], yields an associative bracketed magma $(A, \mu, t \circ \mu)$. Moreover, in [23], it is noticed that $(A, A, t \circ \mu, \delta \circ e)$ is an adjunction which implies that $(A, A, t \circ \mu)$ is a dual pairing (see Example 16). Whence, a Frobenius structure on the semigroup (A, μ) is obtained as stated in Proposition 31.

▶ **Example 34.** A similar argument shows that a Frobenius monoid (A, t, δ, e, μ) in a linear distributive category as defined in [15] yields a Frobenius structure, when the category is actually $*$-autonomous. Indeed, one obtains an associative bracketed magma with the same construction as in the last example. Moreover, Theorem 3.1 of [15] ensures that the pairing is the co-unit of a linear adjunction. That is $(A, A, t \circ \mu)$ is a dual pairing (see Example 16) and we obtain a Frobenius structure just like before.

▶ **Example 35.** A Frobenius structure in the category of sets and relation is easily seen to amount to a tuple (X, R, l, r) with R a ternary relation on X which is associative – i.e. such that, for all $x, y, z, w \in X$, $xyRu$ and $uzRw$ (for some $u \in X$) if and only if $xvRw$ and $yzRv$ (for some $v \in X$) – and with l, r inverse bijections on X satisfying $xyRl(z)$ if and only if $yzRr(x)$, for all $x, y, z \in X$.

6 Nuclear objects and adjunctions

For completeness, we state here that, in an autonomous category, an object is nuclear if and only it is part of an adjunction. This statement is actually a folklore result in the literature. For example, it is explicitly mentioned in [32] that compact closed categories are those autonomous categories for which every object is nuclear, while the same categories are defined in [24] by the property that every object is part of an adjunction. We assume from now on that the object $0 = I$ is the unit of the autonomous category $\mathcal{V}$.

The next statement shows that, in an adjunction of the form (A, A^*, η, ϵ), we can assume that the counit ϵ is the evaluation map and, moreover, require commutativity of only one of the two diagrams defining an adjunction.

▶ **Lemma 36.** *In an autonomous category the following conditions are equivalent:*

1. *A is part of an adjunction,*
2. *there exists a map $\eta : I \longrightarrow A^* \otimes A$ such that diagram (12) commutes,*

$$\begin{array}{ccc} A \otimes I & \xrightarrow{A\otimes\eta} & A \otimes A^* \otimes A \\ \downarrow{\scriptstyle \rho_A} & & \downarrow{\scriptstyle ev_{A,I}\otimes A} \\ A & \xrightarrow{\lambda_A^{-1}} & I \otimes A \end{array} \tag{12}$$

3. *there exists a map $\eta : I \longrightarrow A^* \otimes A$ such that $(A, A^*, \eta, ev_{A,I})$ is an adjunction.*

Proof. If $(A, B, \upsilon, \epsilon)$ is an adjunction, then (A, B, ϵ) is a dual pairing. By Lemma 10, there exists an isomorphism $\psi : B \longrightarrow A^*$ such that $\epsilon = ev_{A,I} \circ (A \otimes \psi)$. Define $\eta := (\psi \otimes A) \circ \upsilon$, we derive

$$\begin{aligned} (ev_{A,I} \otimes A) \circ (A \otimes \eta) &= (ev_{A,I} \otimes A) \circ (A \otimes \psi \otimes A) \circ (A \otimes \upsilon) \\ &= (\epsilon \otimes A) \circ (A \otimes \upsilon) = \lambda_A^{-1} \circ \rho_A \,. \end{aligned}$$

Suppose now that diagram (12) commutes. We claim that the diagram

$$\begin{array}{ccc} I \otimes A^* & \xrightarrow{\eta\otimes A^*} & A^* \otimes A \otimes A^* \\ \downarrow{\scriptstyle \lambda_{A^*}} & & \downarrow{\scriptstyle A^*\otimes ev_{A,I}} \\ A^* & \xleftarrow[\rho_{A^*}]{} & A^* \otimes I \end{array} \tag{13}$$

commutes as well, thus making $(A, A^*, \eta, ev_{A,I})$ into an adjunction. Indeed, the transpose of λ_{A^*} is the canonical map $A \otimes I \otimes A^* \longrightarrow I$ arising, up to unital isomorphisms, from evaluation. The commutative diagram

$$\begin{array}{ccccc} A \otimes I \otimes A^* & \xrightarrow{A\otimes\eta\otimes A^*} & A \otimes A^* \otimes A \otimes A^* & \xrightarrow{A\otimes A^*\otimes ev_{A,I}} & A \otimes A^* \otimes I \\ \downarrow{\scriptstyle \rho_A\otimes A^*} & & \downarrow{\scriptstyle ev_{A,I}\otimes A\otimes A^*} & & \downarrow{\scriptstyle ev_{A,I}\otimes I} \\ A \otimes A^* & \xrightarrow{\lambda_A^{-1}\otimes A^*} & I \otimes A \otimes A^* & \xrightarrow{I\otimes ev_{A,I}} & I \otimes I \\ & & \downarrow{\scriptstyle \lambda_{A\otimes A^*}} & & \downarrow{\scriptstyle \lambda_I} \\ & & A \otimes A^* & \xrightarrow{ev_{A,I}} & I \end{array}$$

shows that the transpose of the rightmost path in (13) is the same canonical map.

Finally, if $(A, A^*, \eta, ev_{A,I})$ is an adjunction, then obviously A is part of an an adjunction. ◀

Recall now that an object A of an autonomous category is *nuclear* if the canonical map $\mathtt{mix} : A^* \otimes A \longrightarrow [A, A]$ is an isomorphism.

▶ **Theorem 37.** *In an autonomous category, the following are equivalent:*

1. *A is nuclear,*
2. *there exists a map $\eta : I \longrightarrow A^* \otimes A$ such that $\mathtt{mix} \circ \eta = \rho_A{}^\sharp : I \longrightarrow [A, A]$,*
3. *A is part of an adjunction.*

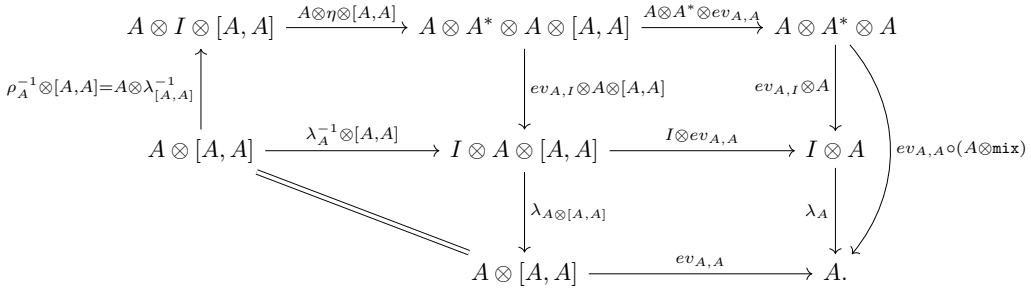
Proof. If $\mathtt{mix}$ is invertible then define $\eta := \mathtt{mix}^{-1} \circ \rho_A{}^\sharp$. That is, 1 implies 2.

To see that 2 implies 3, observe that the relation $\mathtt{mix} \circ \eta = \rho_A{}^\sharp$ is equivalent (under transposition) to commutativity of (12). Therefore, by Lemma 36, A is an adjoint.

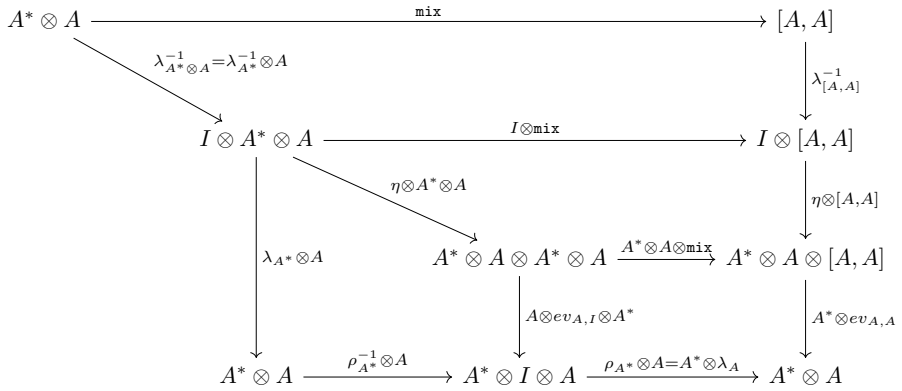
Let us argue that 3 implies 1. Suppose that A is an adjoint. By Lemma 36, there exists η such that $(A, A^*, \eta, ev_{A,I})$ is an adjunction. Then, we define the map $\psi : [A, A] \longrightarrow A^* \otimes A$ by

$$\psi := [A, A] \xrightarrow{\lambda^{-1}_{[A,A]}} I \otimes [A, A] \xrightarrow{\eta \otimes [A,A]} A^* \otimes A \otimes [A, A] \xrightarrow{A^* \otimes ev_{A,A}} A^* \otimes A\,.$$

To see that $\mathtt{mix} \circ \psi$ is the identity of $[A, A]$ observe that its transpose is the evaluation map, as witnessed by the commutativity of the diagram



Also, commutativity of the diagram



shows that $\psi \circ \mathtt{mix} = \mathtt{id}_{A^* \otimes A}$. ◀

7 From nuclearity to Frobenius structures

The next two sections present our main results. First we show how nuclearity yields Frobenius structures on the objects on the internal hom $[A, A]$.

▶ **Theorem 38.** *If A is a nuclear object in a $*$-autonomous category $\mathcal{V}$, then there is a map l such that $([A, A], [A, A]^*, ev_{[A,A],I}, \circ, l, l)$ is a Frobenius structure.*

Proof. As mentioned in Example 4, the object $A^* \otimes A$ has the semigroup structure $\mu_{A^* \otimes A} = \rho_{A^*} \circ (A^* \otimes \lambda_A) \circ (A^* \otimes ev_{A,I} \otimes A)$, see e.g. [26]. It is easily seen that $\epsilon \circ (A^* \otimes A \otimes \mathtt{mix}) = ev_{A,I} \circ \sigma_{A^*,A} \circ \mu_{A^* \otimes A}$, which immediately ensures that the map $\epsilon \circ (A^* \otimes A \otimes \mathtt{mix})$, that is, the transpose of $\mathtt{mix}$, is associative. It is also verified that $\epsilon \circ (A^* \otimes A \otimes \mathtt{mix}) \circ \sigma_{A^* \otimes A, A^* \otimes A} = \epsilon \circ (A^* \otimes A \otimes \mathtt{mix})$, whence $\mathtt{mix}$ is self-adjoint.

It follows that, if $\mathtt{mix}$ is invertible, then the tuples $(A^* \otimes A, [A,A], \epsilon, \mu_{A^*\otimes A}, \mathtt{mix}, \mathtt{mix})$ and $([A,A], A^* \otimes A, \epsilon \circ \sigma_{[A,A],A^*\otimes A}, \mu_{[A,A]}, \mathtt{mix}^{-1}, \mathtt{mix}^{-1})$ are Frobenius structures. Since $\mathtt{mix}$ is a semigroup homomorphism from $(A^* \otimes A, \mu_{A^*\otimes A})$ to $([A,A], \circ)$, then $\mu_{[A,A]}$ is the standard monoid structure $\circ$ induced from composition in $[A,A]$. There is a canonical isomorphism $\psi : A^* \otimes A \cong [A,A]^*$ such that $\epsilon \circ ([A,A] \circ \psi) = ev_{[A,A],I}$, see Example 15. Then, it is easily seen that $([A,A], [A,A]^*, ev_{[A,A],I}, \circ, \psi \circ \mathtt{mix}^{-1}, \psi \circ \mathtt{mix}^{-1})$ is a Frobenius structure. ◀

Let us notice that, in view of Theorem 37 identifying nuclear objects and adjoints, the previous statement can be seen as an instance of [39, Corollary 3.3]. This statement admits a sort of generalization. Since the category SLatt has an epi-mono factorization system which lifts to the category of quantales, we argued in [13] that, by taking the appropriate quantic nucleus, one can always obtain a Girard quantale from the image of $\mathtt{mix}$. This specific situation of SLatt is abstracted as follows:

▶ **Theorem 39.** *Let $\mathcal{V}$ be a $*$-autonomous category with an epi-mono factorization system, (A, B, ϵ) be a dual pairing, and (A, μ_A) be a semigroup in $\mathcal{V}$. Let $f : A \longrightarrow B$ be a map, put $\psi_A := \epsilon \circ (A \otimes f)$ and suppose that $\psi_A = \psi_A \circ \sigma_{A,A}$. Factor f as $f = m \circ e$ with $e : A \longrightarrow C$ epi and $m : C \longrightarrow B$ mono. If C is a magma with multiplication μ_C and e is a magma homomorphism, then there exists a map $g : C \longrightarrow C^*$ making $(C, C^*, ev_{C,I}, \mu_C, g, g)$ into a Frobenius structure.*

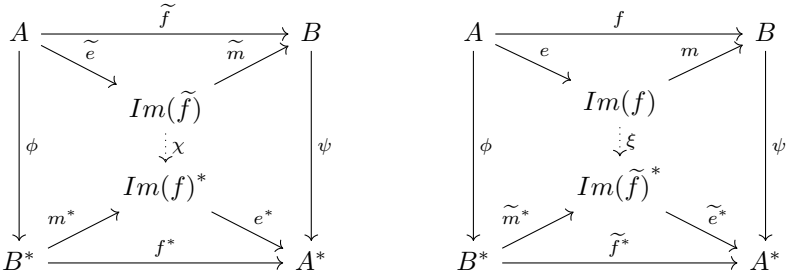
To prove the theorem, we need a Lemma relating factorization systems to dual pairs. If $f = m \circ e$ with e epi and m mono, then we denote by $Im(f)$ the codomain of e.

▶ **Lemma 40.** *Let $\mathcal{V}$ be $*$-autonomous with an epi-mono factorization system. If (A, B) is a dual pair and $f : A \longrightarrow B$ then $(Im(f), Im(\widetilde{f}))$ is a dual pair.*

Proof. As from Proposition 13.1, let $\phi : A \longrightarrow B^*$ and $\psi : B \longrightarrow A^*$ be isomorphisms such that $\phi = \psi^\perp$. We pretend that, up to these isomorphisms, $\widetilde{f}$ equals f^*, as stated below.

▷ Claim 41. We have $\psi \circ \widetilde{f} = f^* \circ \phi$.

Let now (e, m) and $(\widetilde{e}, \widetilde{m})$ be epi-mono factorizations of f and $\widetilde{f}$, respectively. By applying the duality functor $(-)^*$, (m^*, e^*) (resp. $(\widetilde{m}^*, \widetilde{e}^*)$) is an epi-mono factorization of f^* (resp. $\widetilde{f}^*$). Therefore, we obtain two different epi-mono factorizations of $\psi \circ \widetilde{f} = f^* \circ \phi$ (resp. $\psi \circ f = \widetilde{f}^* \circ \phi$), as follows:



and therefore isomorphisms $\chi : Im(\widetilde{f}) \longrightarrow Im(f)^*$ and $\xi : Im(f) \longrightarrow Im(\widetilde{f})^*$. In order to conclude that we have a dual pair, we need to argue that $\xi = \chi^\perp$. To this goal, compute as follows:

$$(\chi \circ \widetilde{e})^\perp \circ e = (e^* \circ \chi \circ \widetilde{e})^\perp = (f^* \circ \phi)^\perp = \phi^\perp \circ f = \psi \circ f = \widetilde{e}^* \circ \xi \circ e\,.$$

From this, considering that e is epi, it follows that $(\chi \circ \widetilde{e})^{\perp} = \widetilde{e^*} \circ \xi$. By applying the operator $(-)^{\perp}$ to both sides of this equality, we obtain $\chi \circ \widetilde{e} = (\widetilde{e^*} \circ \xi)^{\perp} = \xi^{\perp} \circ \widetilde{e}$, from which the desired equality $\chi = \xi^{\perp}$ follows. ◀

Proof of Theorem 39. Let A, B be dual and let $f : A \longrightarrow B$ be such that $\widetilde{f} = f$. Let $e : A \longrightarrow C$ and $m : C \longrightarrow B$ be a factorization of f. Lemma 40 exhibits (C, C) as a dual pair with isomorphism $\chi, \xi : C \longrightarrow C^*$ such that $\chi^{\perp} = \xi = \chi$. Define therefore the pairing $\pi_C := ev_{C,I} \circ (C \otimes \chi) : C \otimes C \longrightarrow I$ and observe that it is symmetric. The diagram in the proof of Lemma 40 also exhibits the equality $\psi \circ f = e^* \circ \chi \circ e : A \longrightarrow A^*$. Transposing this relation, we obtain the pairing

$$\pi_A = (\psi \circ f)^{\flat} = (e^* \circ \chi \circ e)^{\flat} = \chi^{\flat} \circ (e \otimes e) = \pi_C \circ (e \otimes e)\,.$$

Thus, assuming that $f : A \longrightarrow B$ is a semigroup homomorphism factoring as $f = m \circ e$ with $e : (A, \mu_A) \longrightarrow (C, \mu_C)$ a magma homomorphisms, we have that $e : (A, \mu_A, \pi_A) \longrightarrow (C, \mu_C, \pi_C)$ is a bracketed magma homomorphism. As argued in Proposition 8, (C, μ_C, π_C) is an associative bracketed magma, and, as mentioned in Proposition 31, this is enough to have a Frobenius structure on the dual pair (C, C^*). ◀

8 From Frobenius structures to nuclear objects

We finally give the converse of Theorem 38. However, to do so, we impose additional conditions on the carrier object A of a Frobenius quantale $[A, A]$.

▶ **Definition 42.** *We say that an object A of a monoidal category $\mathcal{V}$ is* pseudo-affine *if the tensor unit I embeds into A as a retract. If every object of $\mathcal{V}$ which is not terminal nor initial is pseudo-affine, then $\mathcal{V}$ is said to be* pseudo-affine.

For instance, the category SLatt is pseudo-affine. This property was actually used in [34] to prove that if $[L, L]$ is endowed with a Frobenius quantale structure, then L is completely distributive. Before going there, let us introduce some useful observations.

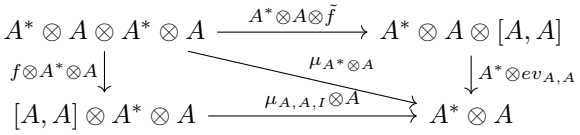
▶ **Lemma 43.** *Tensoring with a pseudo-affine object yields a faithful functor.*

▶ **Proposition 44.** *Identity morphisms of pseudo-affine objects are orthogonal to each other, in the following sense: if A, C are pseudo-affine, $f : A \otimes B_0 \longrightarrow A \otimes B_1$, $g : B_0 \otimes C \longrightarrow B_1 \otimes C$, and $f \otimes C = A \otimes g : A \otimes B_0 \otimes C \longrightarrow A \otimes B_1 \otimes C$, then there exists $h : B_0 \longrightarrow B_1$ such that $f = A \otimes h$ and $g = h \otimes C$.*

The following is the main result in this section.

▶ **Theorem 45.** *If A is pseudo-affine and the semigroup $([A, A], \circ)$ can be completed to a Frobenius structure, then A is part of an adjunction.*

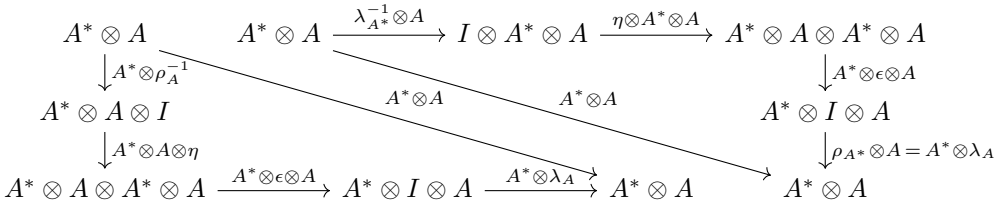
Proof. We can suppose that the dual of $[A, A]$ is $A^* \otimes A$. The dual multiplication $\mu_{A^* \otimes A} : A^* \otimes A \otimes A^* \otimes A \longrightarrow A^* \otimes A$ arises from a map $f : A^* \otimes A \longrightarrow [A, A]$ as the diagonal of the diagram below.



Since A is pseudo-affine, then, by Proposition 44, we necessarily have

$$\mu_{A^*\otimes A} = (A^* \otimes \lambda_A) \circ (A^* \otimes \epsilon \otimes A)$$

for a map $\epsilon : A \otimes A^* \longrightarrow I$. Moreover, since $([A, A], \circ)$ is unital and f is an isomorphism, $\mu_{A^*\otimes A}$ has a unit $\eta : I \longrightarrow A^* \otimes A$, which means that the following diagrams commute:



Since tensoring with A and A^* is faithful, commutativity of the above diagrams is equivalent to the equalities

$$\lambda_A \circ (\epsilon \otimes A) \circ (A \otimes \eta) \circ \rho_A^{-1} = id_A\,, \qquad id_{A^*} = \rho_A \circ (A^* \otimes \epsilon) \circ (\eta \otimes A^*) \circ \lambda_A^{-1}\,,$$

showing that (A, A^*, η, ϵ) is an adjunction. ◀

In view of Theorem 37, we can restate Theorem 45 as follows:

▶ **Theorem 46.** *If A is pseudo-affine and $([A, A], \circ)$ carries a Frobenius structure, then A is nuclear.*

9 Discussion and future work

The work [38] provides a wide playground where to test the strength of the results presented in this paper. For example, for a commutative Girard quantale Q, the category $Q\mathsf{Set}$, whose objects are the pairs (α, X) with $\alpha : X \longrightarrow Q$ and whose morphisms $(\alpha, X) \longrightarrow (\beta, Y)$ are the relations $R \subseteq X \times Y$ such that xRy implies $\alpha(x) \leq \beta(y)$, is $*$-autonomous. If the relation $1 = 1^\perp$ holds in Q, then the unit of the monoidal structure is a dualizing object. It is not difficult to characterize pseudo-affine and nuclear objects in this category. It turns out that, when Q does not contain an infinite chain, if a monoid on $[(\alpha, X), (\alpha, X)]$ can be endowed with a Frobenius structure, then (α, X) is nuclear, even when (α, X) is not pseudo-affine. Thus, the assumption made in Theorem 45 that an object is pseudo-affine is not necessary. On the other hand, we could also construct an infinite quantale Q and an object (α, X) for which the monoid $[(\alpha, X), (\alpha, X)]$ has a Frobenius structure and which is not nuclear.

The notion of nuclearity was originally conceived for Banach spaces [21]. In [13] we have shown how to construct unitless Girard quantales from nuclear endomaps (trace class operators) of an infinite dimensional Hilbert space. It is natural to ask how the results presented in this paper may be extended to categories of Banach spaces and other similar categories (e.g. the category of Hilbert spaces). Most of these categories are symmetric monoidal; however, they are not $*$-autonomous and, in the case of Hilbert spaces and continuous linear maps, not even autonomous. Even though the definitions of dual pair and of Frobenius structure only require the environment category to be symmetric monoidal, a closer look at our results exhibits their dependency on $*$-autonomy. For example, we have used in Theorem 39 the fact that, for m mono, m^* is epi. In categories of Banach spaces, due to the Hanh-Banach theorem, the theorem might still hold if m is a *regular* mono, that

is, an isometry. However, when considering the algebra of trace class operators – which, given the analogy with the tight endomaps of [13], is the natural candidate where to apply Theorem 39 – the relevant map m is not an isometry. We also have used in Theorem 46 and elsewhere that $(A^* \otimes A, [A, A])$ is a dual pair. This fails in autonomous categories, for example, if A is the reflexive Banach space ℓ_p with $1 < p < \infty$, then the tensor $A^* \otimes A$ is no longer reflexive, see [5]. One might restrict to categories of finite dimensional Banach space, which are $*$-autonomous. However, these categories turn out to be uninteresting: if bounded linear maps are taken as morphisms, then all object are nuclear, and if we restrict maps to linear contractions (those linear maps for which $\|f\| \leq 1$), then the only nuclear objects are isomorphic to the monoidal unit [32, §4].

Future research will continue investigating applications of these theorems in concrete $*$-autonomous categories. A main difficulty here will be finding adequate characterizations of nuclear and pseudo-affine objects. We shall also investigate how to relax $*$-autonomy, which might yield results on categories appealing for a wider community of computer scientists and mathematicians – and a step, on our side, towards categorical models of quantum computing.

The considerations just developed suggest, on the other hand, the existence of a close interdependence between provability models of classical linear logic (Frobenius quantales and, in a wider sense, Frobenius structures) and models of proofs of this logic (the $*$-autonomous categories). It is our desire to investigate further whether there is any relation between these structures, similarly to what happens for intuitionistic logic with the Heyting algebra of truth values of a topos. Steps in this direction have already been taken, see e.g. [38, 8].

References

1 Samson Abramsky, Richard Blute, and Prakash Panangaden. Nuclear and trace ideals in tensored *-categories. *J. Pure Appl. Algebra*, 143(1-3):3–47, 1999. doi:10.1016/S0022-4049(98)00106-6.

2 Samson Abramsky and Chris Heunen. H*-algebras and nonunital Frobenius algebras: first steps in infinite-dimensional categorical quantum mechanics. In *Mathematical foundations of information flow*, volume 71 of *Proc. Sympos. Appl. Math.*, pages 1–24. Amer. Math. Soc., Providence, RI, 2012. doi:10.1090/psapm/071/599.

3 Samson Abramsky and Steven Vickers. Quantales, observational logic and process semantics. *Math. Struct. Comput. Sci.*, 3(2):161–227, 1993. doi:10.1017/S0960129500000189.

4 Haroun Amira, Bob Coecke, and Isar Stubbe. How quantales emerge by introducing induction within the operational approach. *Phys. Acta*, 71:554–572, 1998.

5 Alvaro Arias and Jeff D. Farmer. On the structure of tensor products of l_p-spaces. *Pacific J. Math.*, 175(1):13–37, 1996. URL: http://projecteuclid.org/euclid.pjm/1102364179.

6 Michael Barr. *$*$-autonomous categories*, volume 752 of *Lecture Notes in Mathematics*. Springer, Berlin, 1979. doi:10.1007/BFb0064582.

7 R. F. Blute, J. R. B. Cockett, and R. A. G. Seely. Feedback for linearly distributive categories: Traces and fixpoints. *J. Pure Appl. Algebra*, 154(1-3):27–69, 2000. doi:10.1016/S0022-4049(99)00180-2.

8 Kenta Cho, Bart Jacobs, Bas Westerbaan, and Abraham Westerbaan. An introduction to effectus theory. *CoRR*, abs/1512.05813, 2015. arXiv:1512.05813.

9 J. R. B. Cockett, M. Hasegawa, and R. A. G. Seely. Coherence of the double involution on $*$-autonomous categories. *Theory Appl. Categ.*, 17:No. 2, 17–29, 2006.

10 J. R. B. Cockett, J. Koslowski, and R. A. G. Seely. Introduction to linear bicategories. *Math. Struct. Comput. Sci.*, 10(2):165–203, 2000. doi:10.1017/S0960129520003047.

11 J.R.B. Cockett and R.A.G. Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133–173, 1997.

12 Robin Cockett, Cole Comfort, and Priyaa V. Srinivasan. Dagger linear logic for categorical quantum mechanics. *Log. Methods Comput. Sci.*, 17(4):73, 2021. Id/No 8. URL: `https://lmcs.episciences.org/8716`.

13 Cédric de Lacroix and Luigi Santocanale. Unitless Frobenius quantales. Preprint, available at `arXiv:2205.04111`, May 2022.

14 J. M. Egger and David Kruml. Girard Couples of Quantales. *Applied Categorical Structures*, 18(2):123–133, April 2010. `doi:10.1007/s10485-008-9138-3`.

15 J.M. Egger. The Frobenius relations meet linear distributivity. *Theory and Applications of Categories [electronic only]*, 24:25–38, 2010. URL: `http://eudml.org/doc/223263`.

16 Patrik Eklund, Javier Gutiérrez Garcia, Ulrich Höhle, and Jari Kortelainen. *Semigroups in complete lattices*, volume 54 of *Developments in Mathematics*. Springer, Cham, 2018. `doi:10.1007/978-3-319-78948-4`.

17 Nikolaos Galatos, Peter Jipsen, Tomasz Kowalski, and Hiroakira Ono. *Residuated Lattices: An Algebraic Glimpse at Substructural Logics*, volume 151 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2007. `doi:10.1016/S0049-237X(07)80005-X`.

18 Stefano Gogioso and Fabrizio Genovese. Quantum field theory in categorical quantum mechanics. In *Proceedings of the 15th international conference on quantum physics and logic, QPL'18, Halifax, Canada, June 3–7, 2018*, pages 163–177. Waterloo: Open Publishing Association (OPA), 2019. URL: `https://eptcs.web.cse.unsw.edu.au/paper.cgi?QPL2018.9`.

19 Maria João Gouveia and Luigi Santocanale. Mix $\star$-autonomous quantales and the continuous weak order. In Jules Desharnais, Walter Guttmann, and Stef Joosten, editors, *RAMiCS 2018*, volume 11194 of *Lecture Notes in Computer Science*, pages 184–201. Springer, Cham, 2018. `doi:10.1007/978-3-030-02149-8_12`.

20 Maria João Gouveia and Luigi Santocanale. The continuous weak order. *J. Pure Appl. Algebra*, 225(2):37, 2021. Id/No 106472. `doi:10.1016/j.jpaa.2020.106472`.

21 Alexandre Grothendieck. Produits tensoriels topologiques et espaces nucléaires. *Mem. Amer. Math. Soc.*, 16:Chapter 1: 196 pp.; Chapter 2: 140, 1955.

22 D. A. Higgs and K. A. Rowe. Nuclearity in the category of complete semilattices. *J. Pure Appl. Algebra*, 57(1):67–78, 1989. `doi:10.1016/0022-4049(89)90028-5`.

23 Martin Hyland. Abstract interpretation of proofs: Classical propositional calculus. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic, 18th International Workshop, CSL 2004, Proceedings*, volume 3210 of *Lecture Notes in Computer Science*, pages 6–21. Springer, 2004. `doi:10.1007/978-3-540-30124-0_2`.

24 G. M. Kelly and M. L. Laplaza. Coherence for compact closed categories. *J. Pure Appl. Algebra*, 19:193–213, 1980. `doi:10.1016/0022-4049(80)90101-2`.

25 Joachim Kock. *Frobenius Algebras and 2-D Topological Quantum Field Theories*. London Mathematical Society Student Texts. Cambridge University Press, 2003. `doi:10.1017/CBO9780511615443`.

26 David Kruml and Jan Paseka. Algebraic and categorical aspects of quantales. In *Handbook of algebra. Vol. 5*, volume 5 of *Handb. Algebr.*, pages 323–362. Elsevier/North-Holland, Amsterdam, 2008. `doi:10.1016/S1570-7954(07)05006-1`.

27 Saunders Maclane. *Categories for the working mathematician*. Graduate Texts in Mathematics. Springer-Verlag New York, 1978.

28 Roger D. Maddux. *Relation algebras*, volume 150 of *Studies in Logic and the Foundations of Mathematics*. Elsevier B. V., Amsterdam, 2006.

29 George N. Raney. Tight Galois connections and complete distributivity. *Trans. Amer. Math. Soc.*, 97:418–426, 1960. `doi:10.2307/1993380`.

30 Pedro Resende. Quantales, finite observations and strong bisimulation. *Theor. Comput. Sci.*, 254(1-2):95–149, 2001. `doi:10.1016/S0304-3975(99)00123-1`.

31 Kimmo I. Rosenthal. *Quantales and their applications*, volume 234 of *Pitman Research Notes in Mathematics Series*. Longman Scientific & Technical, Harlow, 1990.

32 K. A. Rowe. Nuclearity. *Canad. Math. Bull.*, 31(2):227–235, 1988. doi:10.4153/CMB-1988-035-5.

33 Luigi Santocanale. On discrete idempotent paths. In Robert Mercaş and Daniel Reidenbach, editors, *Combinatorics on Words. WORDS 2019*, volume 11682 of *Lecture Notes in Computer Science*, pages 312–325. Springer, Cham, 2019. doi:10.1007/978-3-030-28796-2_25.

34 Luigi Santocanale. Dualizing sup-preserving endomaps of a complete lattice. In David I. Spivak and Jamie Vicary, editors, *Proceedings of ACT 2020, Cambridge, USA, 6-10th July 2020*, volume 333 of *EPTCS*, pages 335–346, 2020. doi:10.4204/EPTCS.333.23.

35 Luigi Santocanale. The involutive quantaloid of completely distributive lattices. In Uli Fahrenberg, Peter Jipsen, and Michael Winter, editors, *Proceedings of RAMiCS 2020, Palaiseau, France, April 8-11, 2020 [postponed]*, volume 12062 of *Lecture Notes in Computer Science*, pages 286–301. Springer, 2020. doi:10.1007/978-3-030-43520-2_18.

36 Luigi Santocanale. Skew metrics valued in Sugihara semigroups. In Uli Fahrenberg, Mai Gehrke, Luigi Santocanale, and Michael Winter, editors, *Relational and Algebraic Methods in Computer Science - 19th International Conference, RAMiCS 2021, Marseille, France, November 2-5, 2021, Proceedings*, volume 13027 of *Lecture Notes in Computer Science*, pages 396–412. Springer, 2021. doi:10.1007/978-3-030-88701-8_24.

37 Luigi Santocanale and Cédric de Lacroix. Frobenius structures in star-autonomous categories. Preprint, July 2022. URL: https://hal.archives-ouvertes.fr/hal-03739197.

38 Andrea Schalk and Valeria de Paiva. Poset-valued sets or how to build models for linear logics. *Theor. Comput. Sci.*, 315(1):83–107, 2004. doi:10.1016/j.tcs.2003.11.014.

39 Ross Street. Frobenius monads and pseudomonoids. *J. Math. Phys.*, 45(10):3930–3948, 2004. doi:10.1063/1.1788852.

40 David N. Yetter. Quantales and (noncommutative) linear logic. *J. Symb. Log.*, 55(1):41–64, 1990. doi:10.2307/2274953.

Translating Proofs from an Impredicative Type System to a Predicative One

Thiago Felicissimo ✉
Université Paris-Saclay, INRIA project, Deducteam, Laboratoire Méthodes Formelles, ENS Paris-Saclay, 91190 France

Frédéric Blanqui ✉
Université Paris-Saclay, INRIA project, Deducteam, Laboratoire Méthodes Formelles, ENS Paris-Saclay, 91190 France

Ashish Kumar Barnawal ✉
Indian Institute of Technology Guwahati, Guwahati 781039, Assam, India

Abstract

As the development of formal proofs is a time-consuming task, it is important to devise ways of sharing the already written proofs to prevent wasting time redoing them. One of the challenges in this domain is to translate proofs written in proof assistants based on impredicative logics, such as Coq, Matita and the HOL family, to proof assistants based on predicative logics like Agda, whenever impredicativity is not used in an essential way.

In this paper we present an algorithm to do such a translation between a core impredicative type system and a core predicative one allowing prenex universe polymorphism like in Agda. It consists in trying to turn a potentially impredicative term into a universe polymorphic term as general as possible. The use of universe polymorphism is justified by the fact that mapping an impredicative universe to a fixed predicative one is not sufficient in most cases.

During the algorithm, we need to solve unification problems modulo the max-successor algebra on universe levels. But, in this algebra, there are solvable problems having no most general solution. We however provide an incomplete algorithm whose solutions, when it succeeds, are most general ones.

The proposed translation is of course partial, but in practice allows one to translate many proofs that do not use impredicativity in an essential way. Indeed, it was implemented in the tool Predicativize and then used to translate semi-automatically many non-trivial developments from Matita's arithmetic library to Agda, including Bertrand's Postulate and Fermat's Little Theorem, which were not available in Agda yet.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Type theory; Theory of computation → Equational logic and rewriting

Keywords and phrases Type Theory, Impredicativity, Predicativity, Proof Translation, Universe Polymorphism, Unification Modulo Max, Agda, Dedukti

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.19

Related Version *Full Version*: `https://arxiv.org/abs/2211.05700`

Supplementary Material *Software (Source Code)*: `https://github.com/Deducteam/predicativize` archived at `swh:1:dir:b2a1f99fe459c3e9a30debb4285eda1419b5d52d`

Acknowledgements The authors would like to thank François Thiré for the help while developing Predicativize, Gilles Dowek for remarks about previous versions of this paper, Jesper Cockx and Vincent Moreau for discussions about universe levels and the anonymous reviewers for the very helpful comments and remarks.

1 Introduction

An important achievement of the research community in logic is the invention of proof assistants. Such tools allow for interactively writing proofs, which are then checked automatically and can then be reused in other developments. Unfortunately, a proof written in a proof assistant cannot be reused in another one, which makes each tool isolated in its own library of proofs. This is specially the case when considering two proof assistants with incompatible logics, as in this case simply translating from one syntax to another would not work. Therefore, in order to share proofs between systems it is very often required to do logical transformations.

One approach to share proofs from a proof assistant A to a proof assistant B is to define a transformation acting directly on the syntax of A and then implement it using the codebase of A. However, this code would be highly dependent on the implementation of A and can easily become outdated if the codebase of A evolves. Moreover, if there is another proof assistant A' whose logic is very similar to the one of A then this transformation would have to be implemented another time in order to be used with A'.

Dedukti

The logical framework Dedukti [3] is a good candidate for a system where multiple logics can be encoded, allowing for logical transformations to be defined uniformly *inside* Dedukti.

Indeed, first, the framework was already shown to be sufficiently expressive to encode the logics of many proof assistants [11]. Moreover, previous works have shown how proofs can be transformed inside Dedukti. For instance, Thiré describes in [23] a transformation to translate a proof of Fermat's Little Theorem from the Calculus of Inductive Constructions to Higher Order Logic (HOL), which can then be exported to multiple proof assistants such as HOL, PVS, Lean, etc. Géran also used Dedukti to export the formalization of Euclid's Elements Book 1 in Coq [5] to several proof assistants [17].

(Im)Predicativity

One of the challenges in proof interoperability is sharing proofs coming from impredicative proof assistants (the majority of them) to predicative ones such as Agda. Indeed, impredicativity, which is the ability in a logic to quantify over arbitrary entities, regardless of size considerations, is incompatible with predicative systems, in which each entity can only quantify over smaller ones.

Therefore, it is clear that any proof that uses such characteristic in an essential way cannot be translated to a predicative system. Nevertheless, one can wonder if most proofs written in impredicative systems really need impredicativity and, if not, how one could devise a way for detecting and translating them to predicative systems.

Our contribution

In this paper, we tackle this problem by proposing an algorithm that tries to do precisely this. This algorithm was implemented on top of the DkCheck type-checker for Dedukti with the tool Predicativize, allowing for the translation of proofs semi-automatically inside Dedukti. These proofs can then be exported to Agda, the main proof assistant based on predicative type theory.

This tool has been used to translate many proofs semi-automatically to Agda, including Matita's arithmetic library. It contains many-non trivial proofs, and in particular a proof of Bertrand's Postulate which was the subject of a whole publication [1] – thanks to our tool, the same hard work did not have to be repeated in order to make it available in Agda.

Outline

We start in Section 2 with an introduction to Dedukti, before moving to Section 3, where we present informally the problems that appear when translating proofs to predicative systems. We then introduce in Section 4 a predicative universe-polymorphic system, which is a subsystem of Agda and is used as the target of the translation. This is followed by Section 5, the main one, where we present our algorithm. Section 6 then proposes an (incomplete) unification algorithm for universe levels, which is used by the predicativization algorithm. We then introduce the tool Predicativize in Section 7, and describe the translation of Matita's library in Section 8. We end with some remarks in Section 9. The proofs not given in the main body of the article can be found in the long version (see link on the first page).

2 Dedukti

In this work we use Dedukti [3, 11] as the metatheory in which we express the various logic systems and define our proof transformation. Therefore, we start with a quick introduction to this system. The logical framework Dedukti has the syntax of the λ-calculus with dependent types ($\lambda\Pi$-calculus).

$$A, B, M, N ::= x \mid c \mid MN \mid \lambda x : A.M \mid \Pi x : A.B \mid \mathbf{Type} \mid \mathbf{Kind}$$

Here, c ranges in a set of constants $\mathcal{C}$, and x ranges in an infinite set of variables $\mathcal{V}$ disjoint from $\mathcal{C}$. We call a type of the form $\Pi x : A.B$ a *dependent product*, and we write $A \to B$ when x does not appear free in B. We use $\mathbf{s}$ to refer to either **Type** or **Kind**.

A *context* Γ is a finite sequence of entries of the form $x : A$. A *signature* Σ is a finite sequence of entries of the form $c : A$ (constant declarations) or $c : A := M$ (definitions). It can be useful to split the signature into a global signature Σ and a local signature Δ defined on top of the global one. The global signature holds the definition of the object logic we are working in, whereas the local one holds axioms and definitions inside the logic. For instance, when working with natural numbers in predicate logic we would have $\wedge : \mathit{Prop} \to \mathit{Prop} \to \mathit{Prop} \in \Sigma$, as $\wedge$ is in the definition of predicate logic, but $+ : \mathit{Nat} \to \mathit{Nat} \to \mathit{Nat} \in \Delta$, given that the natural numbers and addition are not part of predicate logic, but can be defined on top of it.

The main difference between Dedukti and the $\lambda\Pi$-calculus is that we also consider a set $\mathscr{R}$ of *rewrite rules*, which are pairs of the form $c\ l_1 ... l_k \hookrightarrow r$ where $l_1, ..., l_k, r$ are terms. Given a signature Σ, Δ, we also consider the δ rules allowing for the unfolding of definitions: we have $c \hookrightarrow M \in \delta$ for each $c : A := M \in \Sigma, \Delta$. We then denote by $\hookrightarrow_{\mathscr{R}}$ the closure by context and substitution of $\mathscr{R}$, and by $\hookrightarrow_\delta$ the closure by context of δ. Finally, we write $\hookrightarrow_{\beta\mathscr{R}\delta}$ for $\hookrightarrow_\beta \cup \hookrightarrow_{\mathscr{R}} \cup \hookrightarrow_\delta$ and $\equiv_{\beta\mathscr{R}\delta}$ for its reflexive, symmetric and transitive closure.

Rewriting allows us to define equality by computation, but not all equalities can be defined like this in a well-behaved way, e.g. the commutativity of some operator. Therefore, we also consider *rewriting modulo equations* [6]. If $\mathscr{E}$ is a set of pairs of Dedukti terms (written as $M \approx N$), we write $\simeq_{\mathscr{E}}$ for its congruent closure – that is, its reflexive, symmetric and transitive closure by context and substitution.

Because $\mathscr{R}$ and $\mathscr{E}$ are usually kept fixed, in the following we write $\hookrightarrow$ for $\hookrightarrow_{\beta\mathscr{R}\delta}$, $\simeq$ for $\simeq_{\mathscr{E}}$ and $\equiv$ for the reflexitive, symmetric and transitive closure of $\hookrightarrow \cup \simeq_{\mathscr{E}}$.

$$
\begin{array}{lll}
U_s : \mathbf{Type} & \text{for } s \in \mathcal{S} & u_{s_1} : U_{s_2} \quad \text{for } (s_1, s_2) \in \mathcal{A} \\
El_s : U_s \to \mathbf{Type} & \text{for } s \in \mathcal{S} & El_{s_2}\ u_{s_1} \hookrightarrow U_{s_1} \quad \text{for } (s_1, s_2) \in \mathcal{A} \\
\pi_{s_1,s_2} : \Pi A : U_{s_1}.(El_{s_1}\ A \to U_{s_2}) \to U_{s_3} & & \text{for } (s_1, s_2, s_3) \in \mathcal{R} \\
El_{s_3}\ (\pi_{s_1,s_2}\ A\ B) \hookrightarrow \Pi x : El_{s_1}\ A.El_{s_2}\ (B\ x) & & \text{for } (s_1, s_2, s_3) \in \mathcal{R}
\end{array}
$$

Figure 1 The Dedukti theory which defines the PTS specified by $(\mathcal{S}, \mathcal{A}, \mathcal{R})$.

One very important notion that we will use in this work is that of a *theory*, which is a triple $(\Sigma, \mathscr{R}, \mathscr{E})$ where Σ is a global signature and all constants appearing in $\mathscr{R}$ and $\mathscr{E}$ are declared in Σ. Theories are used to define in Dedukti the object logics in which we work (for instance, predicate logic).

The typing rules for Dedukti are given in Appendix A, along with some basic metaproperties that we use in the subsequent proofs. We remark in particular that the conversion rule of the system allows to exchange types which are equivalent modulo $\equiv$, which can use not only β but also δ, $\mathscr{R}$ and $\mathscr{E}$.

2.1 Defining Pure Type Systems in Dedukti

We briefly review how Pure Type Systems (PTSs) [4] can be defined in Dedukti [12] (other approaches also exists, such as [14]), as we will need this in the rest of the article. Recall that in PTSs, universes and function types can be specified by a set $\mathcal{S}$ of universes, and two relations $\mathcal{A} \subseteq \mathcal{S}^2$ and $\mathcal{R} \subseteq \mathcal{S}^3$ – which we suppose to be functional relations here, as is usually the case. These specify that, if $(s_1, s_2) \in \mathcal{A}$, then s_1 is of type s_2, and if $(s_1, s_2, s_3) \in \mathcal{R}$ then when $A : s_1$ and $B : s_2$ we have $\Pi x : A.B : s_3$. Given a PTS specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, we can define the corresponding PTS with a Dedukti theory in the following manner.

We first start with the definition of universes. For each universe $s \in \mathcal{S}$, we declare a Dedukti type $U_s : \mathbf{Type}$ holding the types in the universe s. We then also declare a function symbol $El_s : U_s \to \mathbf{Type}$ mapping each member of U_s to the type of its elements. We might see the elements of U_s as the codes for the types in s, and El_s as the decoding function, mapping a code to its true type.

In order to represent the fact that a universe s_1 is a member of s_2 when $(s_1, s_2) \in \mathcal{A}$ we add the constant $u_{s_1} : U_{s_2}$. However, now the universe s_1 is represented both by $El_{s_2}\ u_{s_1}$ and U_{s_1}. Therefore, we add the rewrite rule $El_{s_2}\ u_{s_1} \hookrightarrow U_{s_1}$, stating that u_{s_1} decodes to U_{s_1}.

Finally, to define dependent functions, for each $(s_1, s_2, s_3) \in \mathcal{R}$ we add a symbol $\pi_{s_1,s_2} : \Pi A : U_{s_1}.(El_{s_1}\ A \to U_{s_2}) \to U_{s_3}$. Intuitively, the type $El_{s_3}\ (\pi_{s_1,s_2}\ A\ (\lambda x.B))$ should hold the functions from $x : El_{s_1}\ A$ to $El_{s_2}\ B$, where x might occur in B. To make this representation explicit, we add a rewrite rule $\pi_{s_1,s_2}\ A\ B \hookrightarrow \Pi x : El_{s_1}\ A.El_{s_2}\ (B\ x)$. Because the type of functions from $x : A$ to B is now represented by the framework's function type, the framework's abstraction and application can be used to represent the ones of the encoded system.

In the following, we allow ourselves to write $\pi_{s_1,s_2}\ A\ (\lambda x.B)$ informally as $\pi_{s_1,s_2}\ x : A.B$ in order to improve clarity. When $x \notin FV(B)$, we might also write $A \rightsquigarrow_{s_1,s_2} B$.

3 An informal look at the challenges of proof predicativization

In this informal section we present the problem of proof predicativization and discuss the challenges that arise through the use of examples. Even though the examples might be unrealistic, they showcase real problems we found during our first predicativization attempt, of Fermat's little theorem library in HOL [23] – some of them being already noted in [13].

We first start by defining the theories **I** and **P**, which we will use to represent the core logics of impredicative and predicative proof assistants. These theories are defined as Pure Type Systems as explained in Subsection 2.1 and are described by the specifications bellow. Remember that a universe s is said to be impredicative when it is closed under dependent products indexed by some bigger sort, that is, for some s' with $(s, s') \in \mathcal{A}$ we have $(s', s, s) \in \mathcal{R}$. Therefore, **I** is an impredicative system and **P** is a predicative one.

$$\mathcal{S}_{\mathbf{I}} = \{*, \Box\} \qquad \mathcal{S}_{\mathbf{P}} = \mathbb{N}$$
$$\mathcal{A}_{\mathbf{I}} = \{(*, \Box)\} \qquad \mathcal{A}_{\mathbf{P}} = \{(n, n+1) \mid n \in \mathbb{N}\}$$
$$\mathcal{R}_{\mathbf{I}} = \{(*,*,*), (\Box,*,*), (\Box,\Box,\Box)\} \qquad \mathcal{R}_{\mathbf{P}} = \{(n, m, max\{n, m\}) \mid n, m \in \mathbb{N}\}$$

In this setting, the problem of proof predicativization consists in defining a transformation such that, given a local signature Δ with $\Sigma_I, \Delta; - \texttt{well-formed}$, allows to translate it to a local signature Δ' with $\Sigma_P, \Delta'; - \texttt{well-formed}$. Stated informally, we would like to translate constants (which represent axioms) and definitions (which also represent proofs) from **I** into **P**. Note in particular that such a transformation is not applied to a single term but to a sequence of constants and definitions, which can be related by dependency – this dependency turns out to be a major issue as we will see. In the following we represent the local signature Δ in a more readable way as a list of entries **constant** $c : A$ and **definition** $c : A := M$.

Now that our basic notions are explained, let us dive into proof predicativization. For our first step, consider a very simple development showing that for every type P in $*$ we can build an element of $P \rightsquigarrow_{*,*} P$ – if $*$ is a universe of propositions, then this is just a proof that each proposition in $*$ implies itself.

definition $thm_1 : El_* \ (\pi_{\Box,*} \ P : u_*.P \rightsquigarrow_{*,*} P) := \lambda P : U_*.\lambda p : El_* \ P.p$

To translate this simple development, the first idea that comes to mind is to define a mapping on universes: the universe $*$ is mapped to 0 and the universe $\Box$ is mapped to 1. However, because our syntax in Dedukti is heavily annotated, we should only apply this map to the constants u and U, which represent the universes, and then try to recalculate the annotations of the other constants El and π (remember that $\rightsquigarrow$ is just an alias for π). This would then yield the following local signature, which is indeed valid in **P**.

definition $thm_1 : El_1 \ (\pi_{1,0} \ P : u_0.P \rightsquigarrow_{0,0} P) := \lambda P : U_0.\lambda p : El_0 \ P.p$

This naive approach however quickly fails when considering other cases. For instance, suppose now that one adds the following definition – once again, if $*$ is a universe of propositions, then this is just a proof of the proposition $(\forall P.P \Rightarrow P) \Rightarrow \forall P.P \Rightarrow P$.

$$\begin{aligned}\textbf{definition}\ thm_3 : El_* \ ((\pi_{\Box,*} \ P : u_*.P \rightsquigarrow_{*,*} P) \rightsquigarrow_{*,*} \pi_{\Box,*} \ P : u_*.P \rightsquigarrow_{*,*} P)\\ := thm_1 \ (\pi_{\Box,*} \ P : u_*.P \rightsquigarrow_{*,*} P)\end{aligned}$$

If we try to perform the same syntactic translation as before, we get the following result:

$$\begin{aligned}\textbf{definition}\ thm_3 : El_1 \ ((\pi_{1,0} \ P : u_0.P \rightsquigarrow_{0,0} P) \rightsquigarrow_{1,1} \pi_{1,0} \ P : u_0.P \rightsquigarrow_{0,0} P)\\ := thm_1 \ (\pi_{1,0} \ P : u_0.P \rightsquigarrow_{0,0} P)\end{aligned}$$

However, one can verify that this term is not well typed. Indeed, in the original term one quantifies over all types in $*$ in the term $\pi_{\Box,*} \ P : u_*.P \rightsquigarrow_{*,*} P$, and because of impredicativity this term stays at $*$. However, in **P** quantifying over all elements of the universe 0 in $\pi_{1,0} \ P : u_0.P \rightsquigarrow_{0,0} P$ raises the type to the universe 1. As thm_1 expects a term in the universe 0, the term $thm_1 \ (\pi_{1,0} \ P : u_0.P \rightsquigarrow_{0,0} P)$ is not well-typed.

This suggests that impredicativity introduces a kind of *typical ambiguity*, as it allows us to hide in a single universe $*$ all kinds of bigger types which would have to be placed in bigger universes in a predicative setting. Hence, in order to handle cases like this one, which arise a lot in practice, we should not translate every occurrence of $*$ as 0 naively as we did, but try to compute for each occurrence of $*$ some natural number i such that replacing it by i would produce a valid result.

Thankfully, performing such kind of transformations is exactly the goal of Universo [24]. This tool allows one to transport typing derivations between two PTS specifications.

To understand how this works, let us come back to the previous example. Universo starts here by replacing all sorts by occurrences of l, where l is a fresh metavariable representing a natural number.

$$\textbf{definition } thm_1 : El_{l_1}\ (\pi_{l_2,l_3}\ P : u_{l_4}.P \rightsquigarrow_{l_5,l_6} P) := \lambda P : U_{l_7}.\lambda p : El_{l_8}\ P.p$$
$$\textbf{definition } thm_3 : El_{l_9}\ ((\pi_{l_{10},l_{11}}\ P : u_{l_{12}}.P \rightsquigarrow_{l_{13},l_{14}} P) \rightsquigarrow_{l_{15},l_{16}} \pi_{l_{17},l_{18}}\ P :\ u_{l_{19}}.P \rightsquigarrow_{l_{20},l_{21}} P)$$
$$:= thm_1\ (\pi_{l_{22},l_{23}}\ P : u_{l_{24}}.P \rightsquigarrow_{l_{25},l_{26}} P)$$

These of course are not valid proofs in **P**, but in the following step Universo typechecks such development and generates constraints in the process. These constraints are then given to a SMT solver, which is used to compute for each metavariable l a natural number so that the local signature is valid in **P**. For instance, applying Universo to our previous example would produce the following valid local signature in **P**.

$$\textbf{definition } thm_1 : El_2\ (\pi_{2,1}\ P : u_1.P \rightsquigarrow_{1,1} P)) := \lambda P : U_1.\lambda p : El_1\ P.p$$
$$\textbf{definition } thm_3 : El_1\ ((\pi_{1,0}\ P : u_0.P \rightsquigarrow_{0,0} P) \rightsquigarrow_{1,1} \pi_{1,0}\ P : u_0.P \rightsquigarrow_{0,0} P)$$
$$:= thm_1\ (\pi_{1,0}\ P : u_0.P \rightsquigarrow_{0,0} P)$$

By using Universo it is possible to go much further than with the naive syntactical translation. Still, this approach also fails when being employed with real libraries. To see the reason, consider the following minimum example, in which one uses an element of $\pi_{\square,*}\ P : u_*.P \rightsquigarrow_{*,*} P$ twice to build another element of the same type.

$$\textbf{definition } thm_1 : El_*\ (\pi_{\square,*}\ P : u_*.P \rightsquigarrow_{*,*} P) := \lambda P : U_*.\lambda p : El_*\ P.p$$
$$\textbf{definition } thm_2 : El_*\ (\pi_{\square,*}\ P : u_*.P \rightsquigarrow_{*,*} P) := thm_1\ (\pi_{\square,*}\ P : u_*.P \rightsquigarrow_{*,*} P)\ thm_1$$

If we repeat the same procedure as before, we get the following term, which generates unsolvable constraints.

$$\textbf{definition } thm_1 : El_{l_1}\ (\pi_{l_2,l_3}\ P : u_{l_4}.P \rightsquigarrow_{l_5,l_6} P) := \lambda P : U_{l_7}.\lambda p : El_{l_8}\ P.p$$
$$\textbf{definition } thm_2 : El_{l_9}\ (\pi_{l_{10},l_{11}}\ P : u_{l_{12}}.P \rightsquigarrow_{l_{13},l_{14}} P) := thm_1\ (\pi_{l_{15},l_{16}}\ P : u_{l_{17}}.P \rightsquigarrow_{l_{18},l_{19}} P)\ thm_1$$

This happens because the application $thm_1\ (\pi_{l_{15},l_{16}}\ P : u_{l_{17}}.P \rightsquigarrow_{l_{18},l_{19}} P)\ thm_1$ forces l_4 to be both l_{17} and $l_{17}+1$, which is impossible. This example suggests that impredicativity does not only hide the fact that types are stratified, but also the fact that they can be used at any level of this stratification. For instance, in our example we would like to use thm_1 one time with $l_4 = l_{17}$ and another time with $l_4 = l_{17+1}$. In general, when trying to translate libraries using Universo we found that at very early stages a translated proof or object was already needed at multiple universes at the same time, causing the translation to fail.

Therefore, in order to properly compensate for the lack of impredicativity, our solution uses *universe polymorphism*, a feature in type theory (and also present in Agda) that allows defining terms that can later be used at multiple universes [18, 21]. Our translation works by

trying to compute for each definition or declaration its most general universe polymorphic type, and using it when translating the subsequent declarations or definitions. To understand how this is done precisely, let us first introduce universe polymorphism, which is the subject of the following section.

4 A Universe-Polymorphic Predicative Type System

In this section, we define the Universe-Polymorphic Predicative Type System (or just **UPP**), which enriches the Predicative PTS **P** with prenex universe polymorphism [18, 16]. This is in particular a subsystem of the one underlying the Agda proof assistant [22]. As usual, we define this system as a Dedukti theory $\mathbf{UPP} = (\Sigma_{UPP}, \mathscr{R}_{UPP}, \mathscr{E}_{UPP})$.

The main change with respect to **P** is that, instead of indexing the constants $El_s, U_s, u_s, \pi_{s_1,s_2}$ externally, we index them inside the framework [2]. To do this, we first introduce a syntax for *universe levels* inside Dedukti by the following grammar

$$l, l' ::= i \in \mathcal{I} \mid \mathtt{z} \mid \mathtt{s}\ l \mid l \sqcup l'$$

where the constants $\mathtt{z}, \mathtt{s}$ and $\sqcup$ are defined bellow and $\mathcal{I} \subsetneq \mathcal{V}$ is a set of level variables. We also enforce that level variables can only be substituted by other levels.

$$\begin{array}{ll} \mathit{Level} : \mathbf{Type} & \mathtt{s} : \mathit{Level} \to \mathit{Level} \\ \mathtt{z} : \mathit{Level} & \sqcup : \mathit{Level} \to \mathit{Level} \to \mathit{Level} \quad \text{(written infix)} \end{array}$$

The definitions in the theory **P** of $El_s, U_s, u_s, \pi_{s_1,s_2}$ and the related rewrite rules are then replaced by the following ones.[1]

$$\begin{array}{ll} U : \mathit{Level} \to \mathbf{Type} & \pi : \Pi(i_A\ i_B : \mathit{Level})\ (A : U\ i_A).(El\ i_A\ A \to U\ i_B) \to U\ (i_A \sqcup i_B) \\ El : \Pi i : \mathit{Level}.U\ i \to \mathbf{Type} & El\ i'\ (u\ i) \hookrightarrow U\ i \\ u : \Pi i : \mathit{Level}.U\ (\mathtt{s}\ i) & El\ i'\ (\pi\ i_A\ i_B\ A\ B) \hookrightarrow \Pi x : El\ i_A\ A.El\ i_B\ (B\ x) \end{array}$$

We however still allow ourselves to write $El_l, U_l, u_l, \pi_{l,l'}$ in order to improve clarity. We also reuse the previous convention to write $\pi_{l,l'}\ A\ (\lambda x.B)$ as $\pi_{l,l'}\ x : A.B$, or even $A \rightsquigarrow_{l,l'} B$ when $x \notin FV(B)$.

Now, (prenex) universe polymorphism can be represented directly with the use of the framework's function type [2]. Indeed, if a definition contains free level variables, it can be made universe polymorphic by abstracting over such variables. The following example illustrates this.

▶ **Example 1.** The universe polymorphic identity function is given by

$$id = \lambda(i : \mathit{Level}).\lambda(A : U_i).\lambda(a : El_i\ A).a$$

which has type $\Pi i : \mathit{Level}.El_{(\mathtt{s}\ i)}\ (\pi_{(\mathtt{s}\ i),i}\ A : u_i.A \rightsquigarrow_{i,i} A)$. This then allows to use id at any universe level: for instance, we can obtain the polymorphic identity function at the level $\mathtt{z}$ with the application $id\ \mathtt{z}$, which has type $El_{(\mathtt{s}\ \mathtt{z})}\ (\pi_{(\mathtt{s}\ \mathtt{z}),\mathtt{z}}\ A : u_{\mathtt{z}}.A \rightsquigarrow_{\mathtt{z},\mathtt{z}} A)$.

[1] Note that in the following rewrite rules we do not need to impose i' to be equal or convertible to $\mathtt{s}\ i$ or $i_A \sqcup i_B$, given that, for well-typed instances of the rule, this is ensured by typing [7, 2, 20, 9].

Finally, in order to finish our definition we need to specify the definitional equality satisfied by levels, which is the one generated by the following equations [22]. Note that, as stated before, we enforce that $i, i_1, i_2, i_3 \in \mathcal{I}$ can only be replaced by other levels.

$$i_1 \sqcup (i_2 \sqcup i_3) \approx (i_1 \sqcup i_2) \sqcup i_3 \qquad \mathtt{s}\ (i_1 \sqcup i_2) \approx \mathtt{s}\ i_1 \sqcup \mathtt{s}\ i_2 \qquad i \sqcup \mathtt{z} \approx i$$
$$i_1 \sqcup i_2 \approx i_2 \sqcup i_1 \qquad i \sqcup \mathtt{s}\ i \approx \mathtt{s}\ i \qquad i \sqcup i \approx i$$

This definition is justified by the following property. Given a function $\sigma : \mathcal{I} \to \mathbb{N}$, define the interpretation $[\![l]\!]_\sigma$ of a level l by interpreting the symbols $\mathtt{z}, \mathtt{s}$ and $\sqcup$ as zero, successor and max, and by interpreting each variable i by $\sigma(i)$.

▶ **Proposition 2.** *We have $l_1 \simeq l_2$ iff $[\![l_1]\!]_\sigma = [\![l_2]\!]_\sigma$ holds for all σ.*

This also shows that our definition of $\simeq$ agrees with the one used in other works about universe levels [16, 15, 10]. The following basic properties show that $\hookrightarrow$ and $\simeq$ interact well.

▶ **Proposition 3.**
1. *$\hookrightarrow$ is confluent*
2. *If $M \simeq N \hookrightarrow N'$ then, for some M', we have $M \hookrightarrow M' \simeq N'$.*
3. *If $M \equiv N$ then $M \hookrightarrow^* M' \simeq N' \ {}^*\!\hookleftarrow N$.*

Using the third property, one can apply known techniques to show that $\hookrightarrow$ satisfies subject reduction [9, 19] (this can also be automatically verified using DkCheck or Lambdapi).

▶ **Proposition 4.** *If $\Sigma_{UPP}, \Delta; \Gamma \vdash M : A$ and $M \hookrightarrow M'$ then $\Sigma_{UPP}, \Delta; \Gamma \vdash M' : A$.*

The third property is also very important from a practical perspective: it shows that in order to check $M \equiv N$ one does not need to use matching modulo $\simeq$.

5 The algorithm

We are now ready to define the (partial) translation of a local signature Δ to the theory **UPP**. The idea of the translation is that we traverse the signature Δ and at each step we try to compute the most general universe polymorphic version of a definition or constant. The result of a previously translated definition or declaration can then be used at multiple levels for translating entries occurring later in the signature. In order to understand all the following steps intuitively, we will make use of a running example.

▶ **Example 5.** The last example of Section 3 corresponds to the local signature

$$\Delta_I = \mathit{thm}_1 : \mathit{El}_*\ (\pi_{\square,*}\ P : u_*.P \rightsquigarrow_{*,*} P) := \lambda P : U_*.\lambda p : \mathit{El}_*\ P.p\ ,$$
$$\mathit{thm}_2 : \mathit{El}_*\ (\pi_{\square,*}\ P : u_*.P \rightsquigarrow_{*,*} P) := \mathit{thm}_1\ (\pi_{\square,*}\ P : u_*.P \rightsquigarrow_{*,*} P)\ \mathit{thm}_1$$

which is well-formed in the theory **I**. Let us suppose that the first entry of the signature has already been translated, giving the following signature Δ_{thm_1}.

$$\Delta_{\mathit{thm}_1} = \mathit{thm}_1 : \Pi i : \mathit{Level}.\mathit{El}_{(\mathtt{s}\ i)}\ (\pi_{(\mathtt{s}\ i),i}\ P : u_i.P \rightsquigarrow_{i,i} P) := \lambda i : \mathit{Level}.\lambda P : U_i.\lambda p : \mathit{El}_i\ P.p$$

Therefore, as a running example, we will translate step by step the second entry thm_2.

Let us start with some basic auxiliary definitions. Given a local signature Δ such that $\Sigma_{UPP}, \Delta; - \ \mathtt{well\text{-}formed}$ and a constant c occurring in Δ, let us define $\textsc{Arity}(c)$ as the greatest natural number k such that the type of c is of the form $\Pi i_1\ ..\ i_k : \mathit{Level}.A$. Informally, it is the number of level arguments that this constant expects. For instance, we have $\textsc{Arity}(\mathit{thm}_1) = 1$.

Using this function, let us define $\textsc{InsertMetas}(M)$, by the following equations. This function allows us to insert the fresh level variables that will be used to compute the constraints. We suppose that the inserted variables come from a dedicated subset of level variables $\mathcal{M} \subsetneq \mathcal{I}$ and that each inserted variable is fresh.

$\textsc{InsertMetas}(El_s) = El_i$ $\qquad$ $\textsc{InsertMetas}(u_s) = u_i$

$\textsc{InsertMetas}(U_s) = U_i$ $\qquad$ $\textsc{InsertMetas}(\pi_{s_1,s_2}) = \pi_{i,j}$

$\textsc{InsertMetas}(c) = c\ i_1 \ldots i_k$ where $k = \textsc{Arity}(c)$ and $c \neq El_s, U_s, u_s, \pi_{s_1,s_2}$

$\textsc{InsertMetas}(M) = M$ if M is a variable x or $\mathbf{Type}$ or $\mathbf{Kind}$

$\textsc{InsertMetas}(\Pi x : A.B) = \Pi x : \textsc{InsertMetas}(A).\textsc{InsertMetas}(B)$

$\textsc{InsertMetas}(\lambda x : A.M) = \lambda x : \textsc{InsertMetas}(A).\textsc{InsertMetas}(M)$

$\textsc{InsertMetas}(MN) = \textsc{InsertMetas}(M)\ \textsc{InsertMetas}(N)$

▶ **Example 6.** By applying $\textsc{InsertMetas}$ to the type and body of thm_2 we get

$\textsc{InsertMetas}(El_* \ (\pi_{\square,*}\ P : u_*.P \rightsquigarrow_{*,*} P)) = El_{i_1}\ (\pi_{i_2,i_3}\ P : u_{i_4}.P \rightsquigarrow_{i_5,i_6} P)$

$\textsc{InsertMetas}(thm_1\ (\pi_{\square,*}\ P : u_*.P \rightsquigarrow_{*,*} P)\ thm_1) = thm_1\ i_7\ (\pi_{i_8,i_9}\ P : u_{i_{10}}.P \rightsquigarrow_{i_{11},i_{12}} P)\ (thm_1\ i_{13})$

▶ Remark 7. Note that because our first step is erasing the universes that appear in the terms, this translation is defined for all PTS local signatures, and not only those in $\mathbf{I}$. Therefore, it can be applied to proofs coming from systems featuring much more complex universes hierarchies then $\mathbf{I}$, such as the PTS underlying the type systems of Coq and Matita.

Once the fresh level variables are inserted, the next step is to compute the constraints between levels. To do this, we use an approach similar to [18] and define a bidirectional type checking/inference algorithm.

Figures 2 and 3 define rules for computing constraints required for two terms to be convertible or for a term to be typable, respectively. In these rules, we write $\widehat{M}$ for the weak head normal form of M when it exists – thus $\widehat{(-)}$ is a partial function, but becomes total if we suppose the termination of $\hookrightarrow$. As usual, $M \Rightarrow A$ denotes type inference, whereas $M \Leftarrow A$ denotes type checking. We also write $M \Rightarrow_{sort} \mathbf{s}$ or $M \Rightarrow_\Pi \Pi x : A.B$ as a shorthand for $M \Rightarrow A'$ and $\widehat{A'} = \mathbf{s}$ or $\widehat{A'} = \Pi x : A.B$ respectively.

$$\frac{l, l'\ Level}{l \equiv^? l' \downarrow \{l = l'\}} \qquad \frac{M = x, c, \mathbf{Type}, \mathbf{Kind}}{M \equiv^? M \downarrow \varnothing} \qquad \frac{M \equiv^? M' \downarrow C_1 \qquad \widehat{N} \equiv^? \widehat{N'} \downarrow C_2}{MN \equiv^? M'N' \downarrow C_1 \cup C_2}$$

$$\frac{\widehat{A} \equiv^? \widehat{A'} \downarrow C_1 \qquad \widehat{B} \equiv^? \widehat{B'} \downarrow C_2}{\Pi x : A.B \equiv^? \Pi x : A'.B' \downarrow C_1 \cup C_2} \qquad \frac{\widehat{A} \equiv^? \widehat{A'} \downarrow C_1 \qquad \widehat{M} \equiv^? \widehat{M'} \downarrow C_2}{\lambda x : A.M \equiv^? \lambda x : A'.M' \downarrow C_1 \cup C_2}$$

Figure 2 Inference rules for computing constraints for two terms in whnf to be convertible.

Intuitively, these judgments define a conditional typing relation that depends on the constraints being satisfied. This intuition is formalized by the following results.

▶ **Definition 8.** *Given a level substitution θ (sending level variables to levels) and a set of constraints C, containing pairs of levels $l = l'$, we write $\theta \vDash C$ when for all $l = l' \in C, l\theta \simeq l'\theta$.*

▶ **Lemma 9.** *If $M \equiv^? N \downarrow C$ and $\theta \vDash C$ then $M\theta \equiv N\theta$.*

Let us write $\vec{i}_X$ for the free level variables in X. We also shorten $\vec{i} : Level$ as $\vec{i}$.

▶ **Lemma 10.** *Given a level substitution θ, suppose $\Sigma_{UPP}, \Delta; \vec{i}_{\Gamma\theta}, \Gamma\theta$ `well-formed`.*

- *If $\Sigma_{UPP}, \Delta; \Gamma \vdash M \Rightarrow A \downarrow C$ and $\theta \vDash C$ then $\Sigma_{UPP}, \Delta; \vec{i}_{\Gamma\theta} \cup \vec{i}_{M\theta} \cup \vec{i}_{A\theta}, \Gamma\theta \vdash M\theta : A\theta$*
- *If $\Sigma_{UPP}, \Delta; \Gamma \vdash M \Leftarrow A \downarrow C$, $\theta \vDash C$ and $\Sigma_{UPP}, \Delta; \vec{i}_{\Gamma\theta} \cup \vec{i}_{A\theta}, \Gamma\theta \vdash A\theta : \mathbf{s}$ then we have $\Sigma_{UPP}, \Delta; \vec{i}_{\Gamma\theta} \cup \vec{i}_{M\theta} \cup \vec{i}_{A\theta}, \Gamma\theta \vdash M\theta : A\theta$*

$$\frac{c : A := M \in \Sigma_{UPP}, \Delta \text{ or } c : A \in \Sigma_{UPP}, \Delta}{\Sigma_{UPP}, \Delta; \Gamma \vdash c \Rightarrow A \downarrow \varnothing}\ \textsc{Cons} \qquad \frac{x : A \in \Gamma}{\Sigma_{UPP}, \Delta; \Gamma \vdash x \Rightarrow A \downarrow \varnothing}\ \textsc{Var}$$

$$\frac{i \in \mathcal{M}}{\Sigma_{UPP}, \Delta; \Gamma \vdash i \Rightarrow \mathit{Level} \downarrow \varnothing}\ \textsc{Lvl-Var} \qquad \frac{}{\Sigma_{UPP}, \Delta; \Gamma \vdash \mathbf{Type} \Rightarrow \mathbf{Kind} \downarrow \varnothing}\ \textsc{Sort}$$

$$\frac{\Sigma_{UPP}, \Delta; \Gamma \vdash A \Leftarrow \mathbf{Type} \downarrow C_1 \qquad \Sigma_{UPP}, \Delta; \Gamma, x : A \vdash B \Rightarrow_{sort} \mathbf{s} \downarrow C_2}{\Sigma_{UPP}, \Delta; \Gamma \vdash \Pi x : A.B \Rightarrow \mathbf{s} \downarrow C_1 \cup C_2}\ \textsc{Prod}$$

$$\frac{\Sigma_{UPP}, \Delta; \Gamma \vdash A \Leftarrow \mathbf{Type} \downarrow C_1 \qquad \Sigma_{UPP}, \Delta; \Gamma, x : A \vdash M \Rightarrow B \downarrow C_3 \qquad \Sigma_{UPP}, \Delta; \Gamma, x : A \vdash B \Rightarrow_{sort} \mathbf{s} \downarrow C_2}{\Sigma_{UPP}, \Delta; \Gamma \vdash \lambda x : A.M \Rightarrow \Pi x : A.B \downarrow C_1 \cup C_2 \cup C_3}\ \textsc{Abs}$$

$$\frac{\Sigma_{UPP}, \Delta; \Gamma \vdash M \Rightarrow_{\Pi} \Pi x : A.B \downarrow C_1 \qquad \Sigma_{UPP}, \Delta; \Gamma \vdash N \Leftarrow A \downarrow C_2}{\Sigma_{UPP}, \Delta; \Gamma \vdash MN \Rightarrow B\{N/x\} \downarrow C_1 \cup C_2}\ \textsc{App}$$

$$\frac{\Sigma_{UPP}, \Delta; \Gamma \vdash M \Rightarrow A \downarrow C_1 \qquad \widehat{A} \equiv^{?} \widehat{B} \downarrow C_2}{\Sigma_{UPP}, \Delta; \Gamma \vdash M \Leftarrow B \downarrow C_1 \cup C_2}\ \textsc{Check}$$

Figure 3 Inference rules for computing constraints for a term to be typable.

▶ **Example 11.** We can use the rules with our running example. We first calculate $\Sigma_{UPP}, \Delta_{thm_1}; - \vdash El_{i_1}\ (\pi_{i_2,i_3}\ P : u_{i_4}.P \rightsquigarrow_{i_5,i_6} P) \Rightarrow_{sort} \mathbf{s} \downarrow C_1$. Therefore, any substitution θ with $\theta \vDash C_1$ applied to the previous term results in a valid type. We then calculate

$$\Sigma_{UPP}, \Delta_{thm_1}; - \vdash thm_1\ i_7\ (\pi_{i_8,i_9}\ P : u_{i_{10}}.P \rightsquigarrow_{i_{11},i_{12}} P)\ (thm_1\ i_{13}) \Leftarrow El_{i_1}\ (\pi_{i_2,i_3}\ P : u_{i_4}.P \rightsquigarrow_{i_5,i_6} P) \downarrow C_2$$

This gives $C_1 \cup C_2 = \{i_8 = \mathbf{s}\ i_{10}, i_{11} = i_{10}, i_{12} = i_{10}, i_9 = i_{10} \sqcup i_{12}, i_8 \sqcup i_9 = i_7, i_{13} = i_{10}, i_1 = i_2 \sqcup i_3, \mathbf{s}\ i_4 = i_2, i_5 = i_4, i_6 = i_4, i_3 = i_5 \sqcup i_6, i_4 = i_{10}\}$.

Once the constraints are computed the next step is to solve them. However, as explained in Section 3, we do not want a numerical assignment of level variables that satisfies the constraints, but rather a general symbolic solution which allows the term to be instantiated later at different universe levels. This leads us to use equational unification but, as levels are not purely syntactic entities, one needs to devise a unification algorithm for the equational theory $\simeq$. For now, let us postpone this to the next section and assume we are given a (partial) function Unify which computes from a set of constraints C a unifier θ – that is, a substitution satisfying $\theta \vDash C$. We however do not assume that θ is the most general unifier – as we show later in Theorem 15, such a most general unifier might not exist.

After an unifier θ is found, the final step is then to apply it.

▶ **Example 12.** Given the previous computed constraints $C_1 \cup C_2$ we can compute the substitution θ which sends all variables to i_4, except for i_1, i_2, i_7, i_8, which are sent to $\mathbf{s}\ i_4$, and verify that $\theta \vDash C_1 \cup C_2$. By applying θ, and by Lemma 10, we have

$$\Sigma_{UPP}, \Delta_{thm_1}; i_4 \vdash thm_1\ (\mathbf{s}\ i_4)\ (\pi_{(\mathbf{s}\ i_4),i_4}\ P : u_{i_4}.P \rightsquigarrow_{i_4,i_4} P)\ (thm_1\ i_4) : El_{(\mathbf{s}\ i_4)}\ (\pi_{(\mathbf{s}\ i_4),i_4}\ P : u_{i_4}.P \rightsquigarrow_{i_4,i_4} P)$$

Note that in this term, the constant thm_1 is used at two different universe levels.

The final algorithm can now be described by the pseudocode in Figure 4. The algorithm might fail at any point when either it is not able to compute the constraints, or if the unification algorithm is not capable of inferring a substitution from the constraints. However, if the algorithm returns, its correctness is guaranteed by the following theorem:

▶ **Theorem 13.** *If* $|\Delta|$ *is defined, then* $\Sigma_{UPP}, |\Delta|; -$ `well-formed`.

$$
\begin{aligned}
|-| &= - \\
|\Delta, c : A| &= \text{let } A' = \textsc{InsertMetas}(A) \\
&\quad \text{let } C \text{ be such that } \Sigma_{UPP}, |\Delta|; - \vdash A' \Rightarrow_{sort} \mathbf{s} \downarrow C \text{ else } \mathit{raise}\ \bot \text{ if no such } C \\
&\quad \text{let } \theta = \textsc{Unify}(C) \text{ else } \mathit{raise}\ \bot \text{ if no such } \theta \\
&\quad \text{let } \vec{i} = \vec{i}_{A'\theta} \\
&\quad |\Delta|, c : \Pi \vec{i} : \mathit{Level}.A'\theta \\
|\Delta, c : A := M| &= \text{let } A', M' = \textsc{InsertMetas}(A), \textsc{InsertMetas}(M) \\
&\quad \text{let } C_1 \text{ be such that } \Sigma_{UPP}, |\Delta|; - \vdash A' \Rightarrow_{sort} \mathbf{s} \downarrow C_1 \text{ else } \mathit{raise}\ \bot \text{ if no such } C_1 \\
&\quad \text{let } C_2 \text{ be such that } \Sigma_{UPP}, |\Delta|; - \vdash M' \Leftarrow A' \downarrow C_2, \text{ else } \mathit{raise}\ \bot \text{ if no such } C_2 \\
&\quad \text{let } \theta = \textsc{Unify}(C_1 \cup C_2) \text{ else } \mathit{raise}\ \bot \text{ if no such } \theta \\
&\quad \text{let } \tau = i \mapsto \text{if } i \in \vec{i}_{M'\theta} \backslash \vec{i}_{A'\theta} \text{ then } \mathbf{z} \text{ else } i \\
&\quad \text{let } \vec{i} = \vec{i}_{A'\theta} \\
&\quad |\Delta|, c : \Pi \vec{i} : \mathit{Level}.A'\theta := \lambda \vec{i} : \mathit{Level}.M'\theta\tau
\end{aligned}
$$

Figure 4 Pseudocode of the predicativization algorithm.

Proof. By induction on Δ, the base case being trivial. For the induction step, we have either $\Delta = \Delta'; c : A$ or $\Delta = \Delta'; c : A := M$. In both cases, if $|\Delta|$ is defined, then so is $|\Delta'|$, and thus by induction hypothesis we have $|\Delta'|; - \texttt{well-formed}$. We proceed with a case analysis on the entry.

Definition: As $\Sigma_{UPP}, |\Delta'|; - \vdash A' \Rightarrow T \downarrow C_1$ and $\theta \vDash C_1$, by Lemma 10 we get $\Sigma_{UPP}, |\Delta'|; \vec{i}_{A'\theta} \cup \vec{i}_{T\theta} \vdash A'\theta : T\theta$. Because $\widehat{T} = \mathbf{s}$, we also have $T\theta \hookrightarrow^* \mathbf{s}$, hence we can derive $\Sigma_{UPP}, |\Delta'|; \vec{i}_{A'\theta} \cup \vec{i}_{T\theta} \vdash A'\theta : \mathbf{s}$. By applying the substitution lemma with the substitution sending every variable i in $\vec{i}_{T\theta}$ but not in $\vec{i}_{A'\theta}$ to $\mathbf{z}$, we get $\Sigma_{UPP}, |\Delta'|; \vec{i}_{A'\theta} \vdash A'\theta : \mathbf{s}$. Because we also have $\Sigma_{UPP}, |\Delta'|; - \vdash M' \Leftarrow A' \downarrow C_2$ and $\theta \vDash C_2$, by Lemma 10 again we get $\Sigma_{UPP}, |\Delta'|; \vec{i}_{M'\theta} \cup \vec{i}_{A'\theta} \vdash M'\theta : A'\theta$. By applying the substitution lemma with the substitution τ, we get $\Sigma_{UPP}, |\Delta'|; \vec{i} \vdash M'\theta\tau : A'\theta$, where $\vec{i} := \vec{i}_{A'\theta}$. Finally, by abstracting each free level variable, we get $\Sigma_{UPP}, |\Delta'|; - \vdash \lambda \vec{i} : \mathit{Level}.M'\theta\tau : \Pi \vec{i} : \mathit{Level}.A'\theta$. Hence, we can derive $\Sigma_{UPP}, |\Delta'|, c : \Pi \vec{i} : \mathit{Level}.A'\theta := \lambda \vec{i} : \mathit{Level}.M'\theta\tau; - \texttt{well-formed}$.

Constant: Similar to the previous case. ◀

▶ Remark 14. One could also wonder if the algorithm always terminates and produces a result (be it a valid signature or $\bot$). By supposing strong normalization for **UPP**, and by checking at each step of the rules in Figure 3 that the constraints are consistent, one could show termination of the algorithm by using a similar technique as in [18]. As we do not investigate strong normalization of **UPP** in this paper, we leave termination of the algorithm for future work. However, as we will see in Section 8, when using it in practice we were able to translate many proofs without non-termination issues.

Our algorithm relies on an unspecified function Unify in order to solve the constraints. In order to fully specify it, we thus present an unification algorithm for $\simeq$ in the next section. As we will discuss, the unification algorithm we propose is not guaranteed to always find a most general unifier whenever there is one. However, note that our predicativization algorithm can in principle be used with any unification algorithm for $\simeq$. Therefore, if we have a better unification algorithm in the future, we do not have the modify the algorithm of Figure 4.

6 Solving level constraints

Before addressing the problem of how to solve level constraints, the first natural question that comes to mind is if one can always find a most general unifier (mgu) when the constraints are solvable. The following result answers this negatively.

▶ **Theorem 15.** *Not all solvable problems of unification modulo $\simeq$ over levels have most general unifiers.*

Proof. Consider the equation $\mathtt{s}\ i_1 = i_2 \sqcup i_3$, which is a solvable unification problem, and suppose it had a mgu θ. Note that $\theta_1 = i_1 \mapsto \mathtt{z}, i_2 \mapsto \mathtt{s}\ \mathtt{z}, i_3 \mapsto \mathtt{z}$ is also a solution, thus there is some τ such that $i_3\theta\tau \simeq \mathtt{z}$. Therefore, there can be no occurrence of s in $i_3\theta$. By taking $\theta_2 = i_1 \mapsto \mathtt{z}, i_2 \mapsto \mathtt{z}, i_3 \mapsto \mathtt{s}\ \mathtt{z}$ we can show similarly that there can be no occurrence of $\mathtt{s}$ in $i_2\theta$. But by taking the substitution $\theta' = \_ \mapsto \mathtt{z}$ mapping all variables to $\mathtt{z}$, we get $(i_2 \sqcup i_3)\theta\theta' \simeq \mathtt{z}$, which cannot be equivalent to $(\mathtt{s}\ i_1)\theta\theta'$. Hence, $\mathtt{s}\ i_1 = i_2 \sqcup i_3$ has no mgu. ◀

Therefore, no unification algorithm for $\simeq$ can always produces a mgu. Hence, our algorithm will produce three kinds of results: either it produces a substitution, in which case it is a mgu; or it produces $\bot$, in which case there is no solution to the constraints; or it gets stuck on a set of constraints that it cannot handle. Still, it is not guaranteed to compute a mgu whenever there is one.

Before presenting the algorithm, the first issue we have to address is the fact that levels can have multiple equivalent representations. It would be convenient if we had a syntactical way to compare them. Thankfully, previous works have already addressed this problem.

Let us assume from this point on that level variables in $\mathcal{I}$ admit a total order $<$. Given a strictly increasing sequence of level variables $V = i_1, ..., i_k$, and an V-indexed family of levels $\{l_i\}_{i\in V}$, let $\sqcup_{i\in V} l_i$ denote the term $l_1 \sqcup (l_2 \sqcup ...(l_{k-1} \sqcup l_k)...)$. Moreover, given a natural number k, let $\mathtt{s}^k\ l$ be inductively defined by $\mathtt{s}^0\ l = l$ and $\mathtt{s}^{n+1}\ l = \mathtt{s}\ (\mathtt{s}^n\ l)$.

▶ **Definition 16** (Level normal form)**.** *A level is in normal form when it is of the form $\mathtt{s}^k\ \mathtt{z} \sqcup (\sqcup_{i\in V} \mathtt{s}^{n_i}\ i)$ with $n_i \leqslant k$ for all i.*

Previous works [16, 15, 10] have established that for every level l there is a unique level in normal form, which we refer to as $\widehat{l}$, with $l \simeq \widehat{l}$ – see for instance Lemma 6.2.5 of [15]. We will not describe explicitly here the algorithm for computing normal forms, as this has already been thoroughly explained in previous works, such as in [16, 10] – we note nevertheless that this procedure is sketched in the proof of Proposition 2.

We also define a notion of normal form for constraints.

▶ **Definition 17.** *A constraint $l_1 = l_2$ is said to be in normal form if*

1. *Both l_1, l_2 are in normal form – so we write $l_p = \mathtt{s}^{k_p}\ \mathtt{z} \sqcup (\sqcup_{i\in V_p} \mathtt{s}^{n_i^p}\ i)$ for $p = 1, 2$*
2. *If $i \in V_1 \cap V_2$, then $n_i^1 = n_i^2$*
3. *At least one of the numbers in $\{k_1, k_2\} \cup \{n_i^1\}_{i\in V_1} \cup \{n_i^2\}_{i\in V_2}$ is equal to 0*

Every constraint can be put in normal form, and for this we can use the algorithm in Figure 5. From the second line on, we use k^p, n_i^p to refer to the indices in the normal forms of l_1, l_2. Moreover, the pseudocode should be read imperatively, in the sense that l_1, l_2, V_1, V_2 are updated at each step.

Given a set of constraints C, let $\widehat{C}$ denote the result of putting all constraints of C in normal form by applying the algorithm of Figure 5.

▶ **Lemma 18.** *For all substitutions θ, we have $\theta \models C$ iff $\theta \models \widehat{C}$.*

let $l_1, l_2 = \widehat{l_1}, \widehat{l_2}$
for each $i \in V_1 \cap V_2$
  if $n_i^1 < n_i^2$ then remove $\mathtt{s}^{n_i^1}\ i$ from l_1
  else if $n_i^1 > n_i^2$ then remove $\mathtt{s}^{n_i^2}\ i$ from l_2
substract the minimum value of the set $\{k_1, k_2\} \cup \{n_i^1\}_{i \in V_1} \cup \{n_i^2\}_{i \in V_2}$ from all of its elements

Figure 5 Imperative algorithm for putting a constraint in normal form.

By putting constraints in normal form, we can help our unification algorithm to find a solution, as shown by the following example.

▶ **Example 19.** Consider the constraint $i \sqcup \mathtt{s}\ (i \sqcup \mathtt{s}\ j) = j \sqcup \mathtt{s}\ (\mathtt{s}\ i)$ – which, as we will see, cannot be treated by our unification algorithm if it is not normalized first. By first computing the level normal form of each side, we get $\mathtt{s}^2\ \mathtt{z} \sqcup \mathtt{s}\ i \sqcup \mathtt{s}^2\ j = \mathtt{s}^2\ \mathtt{z} \sqcup \mathtt{s}^2\ i \sqcup j$. As both variables appear in both sides, we remove from each of the sides the occurrence with the smaller index, giving $\mathtt{s}^2\ \mathtt{z} \sqcup \mathtt{s}^2\ j = \mathtt{s}^2\ \mathtt{z} \sqcup \mathtt{s}^2\ i$. Finally, as the minimum among all indices is 2, we subtract this from all of them, giving $\mathtt{z} \sqcup j = \mathtt{z} \sqcup i$ – a constraint that can be treated by our unification algorithm.

Write $\mathcal{I}(l)$ for the level variables appearing in l. Given a substitution θ, we define the sets $\mathit{dom}\ \theta = \{i \mid i \neq i\theta\}$ and $\mathit{range}\ \theta = \cup_{i \in \mathit{dom}\ \theta} \mathcal{I}(i\theta)$, and the substitutions $\widehat{\theta} = \{i \mapsto \widehat{i\theta}\}_{i \in \mathit{dom}\ \theta}$, and $\theta\{l/j\} = \{i \mapsto i\theta\{l/j\}\}_{i \in \mathit{dom}\ \theta}$. Finally, we also define $\mathcal{I}(\theta) = \mathit{range}\ \theta \cup \mathit{dom}\ \theta$.

(**Trivial**)	$\{l = l\} \cup C; \theta \rightsquigarrow C; \theta$	
(**Orient**)	$\{l = l'\} \cup C; \theta \rightsquigarrow \{l' = l\} \cup C; \theta$	if $l' = \mathtt{z}$ or $\mathtt{z} \sqcup i$
(**Eliminate 1**)	$\{\mathtt{z} \sqcup i = l\} \cup C; \theta \rightsquigarrow \widehat{C\{l/i\}}; \widehat{\theta\{l/i\}}, i \mapsto l$	if $i \notin l$
(**Eliminate 2**)	$\{\mathtt{z} \sqcup i = l\} \cup C; \theta \rightsquigarrow$	if $\mathtt{s}^m\ i \in l$ with $m = 0$
	let $l' = l\{i'/i\}$ in $\widehat{C\{l'/i\}}; \widehat{\theta\{l'/i\}}, i \mapsto l'$	for some $i' \in \mathcal{I}_{\mathit{fresh}} \backslash \mathcal{I}(i, l, C, \theta)$
(**Decompose**)	$\{\mathtt{z} = \mathtt{z} \sqcup (\sqcup_{i \in V}\ i)\} \cup C; \theta \rightsquigarrow \{\mathtt{z} \sqcup i = \mathtt{z}\}_{i \in V} \cup C; \theta$	
(**Clash**)	$\{\mathtt{z} = l\} \cup C; \theta \rightsquigarrow \bot$	if $\mathtt{s}^n\ i \in l$ or $\mathtt{s}^n\ \mathtt{z} \in l$ with $n \neq 0$

Figure 6 Unification algorithm for $\simeq$.

We are now ready to present the unification algorithm, whose rules are given in Figure 6. Steps are represented by rules of the form $C; \theta \rightsquigarrow C'; \theta'$, with the pre-conditions that constraints in C are in normal form, $\mathit{dom}\ \theta$ is disjoint from $\mathit{range}\ \theta$, the image of θ contains only levels in normal form, and $\mathit{dom}\ \theta$ is disjoint from $\mathcal{I}(C)$ – these properties are preserved by each step. In rule (Eliminate 2), $\mathcal{I}_{\mathit{fresh}} \subsetneq \mathcal{I}$ is an infinite set of fresh level variables. Finally, it may happen that, for some non-empty sets of constraints C, no rule applies. This corresponds to the cases in which our algorithm gets stuck and does not produce a solution.

Let us write $\theta_1 \subseteq \theta_2$ when for all $i \in \mathit{dom}\ \theta_1$, $i\theta_1 = i\theta_2$ and $\mathit{dom}\ \theta_2 \backslash (\mathit{dom}\ \theta_1) \subseteq \mathcal{I}_{\mathit{fresh}}$ – that is, θ_2 extends θ_1 only inside $\mathcal{I}_{\mathit{fresh}}$.

The following lemma is key in showing the main properties of our algorithm.

▶ **Lemma 20** (Key lemma). *Suppose $C; \theta \rightsquigarrow C'; \theta'$. For all τ, if $\tau \vDash C$ and $\tau \simeq \tau \circ \theta$ then there is a substitution τ' with $\tau \subseteq \tau'$ such that (1) $\tau' \vDash C'$ and (2) $\tau' \simeq \tau' \circ \theta'$. Conversely, for all τ, if $\tau \simeq \tau \circ \theta'$ and $\tau \vDash C'$, then $\tau \simeq \tau \circ \theta$ and $\tau \vDash C$.*

It is clear that $\rightsquigarrow$ does not always terminate, given that some rules create constraints and in particular that the rule (Orient) can loop. However, it is easy to check that these created constraints can always be eliminated by applying other rules. Indeed, the constraints created by (Decompose) can be eliminated by using (Eliminate 1), and the constraint created by (Orient) can be eliminated either by (Eliminate 1), (Eliminate 2), (Clash) or (Decompose), whose created constraints are eliminated once again using (Eliminate 1). Let $\rightsquigarrow_0$ be the relation that packs all of this into a single reduction, which therefore never creates constraints.

▶ **Lemma 21.** *$\rightsquigarrow_0$ terminates.*

In the following theorems, we suppose that $\mathcal{I}(C)$ and $\mathcal{I}_{\mathit{fresh}}$ are disjoint.

▶ **Theorem 22.** *If $C; \mathit{id} \rightsquigarrow_0^* \bot$, then for no θ we have $\theta \vDash C$.*

▶ **Theorem 23.** *If $C; \mathit{id} \rightsquigarrow_0^* \varnothing; \theta$, then θ is a most general unifier.*

Proof. Let τ be a unifier. We thus have $\tau \vDash C$. Moreover, we have $\tau \simeq \tau \circ \mathit{id}$. By iterating Lemma 20, we get a substitution τ' such that $\tau' \simeq \tau' \circ \theta$ and $\tau \subseteq \tau'$. Because $\mathit{dom}\ \tau' \backslash (\mathit{dom}\ \tau) \subseteq \mathcal{I}_{\mathit{fresh}}$, which is disjoint from $\mathcal{I}(C)$, we have $i\tau = i\tau'$ for $i \in \mathcal{I}(C)$. Hence, for $i \in \mathcal{I}(C)$ we have $i\tau \simeq i\theta\tau'$, showing that τ is an instance of θ.

To show that θ is a unifier, note that $\theta = \theta \circ \theta$ and $\theta \vDash \varnothing$, hence by iterating Lemma 20 in the inverse direction we get $\theta \vDash C$. ◀

We have seen that when the algorithm finishes with $\varnothing; \theta$, then θ is a mgu, and when it finishes with $\bot$, then there is no solution to the constraints. However, the algorithm can also get stuck on constraints that it does not know how to solve. In practice, it is very unsatisfying for the unification to get stuck, as this means that the whole predicativization algorithm has to halt. Thus, in order to prevent this, in our implementation we extended the unification with heuristics that are *only* applied when none of the presented rules applies. Then, whenever the heuristics are applied, the universe polymorphic definition or declaration that is produced might not be the most general one.

7 Predicativize, the implementation

In this section we present Predicativize, an implementation of our algorithm. It is publicly available at `https://github.com/Deducteam/predicativize/`.

Our tool is implemented on top of DkCheck [19], a type-checker for Dedukti, and thus does not rely neither on the codebase of Agda, nor on the codebase of any other proof assistant. Like Universo [24], our implementation instruments DkCheck's conversion checking in order to implement the constraint computation algorithm described in Section 5.

Because the currently available type-checkers for Dedukti do not implement rewriting modulo for equational theories other than AC (associative commutative), we used Genestier's encoding of levels [16] in order to define the theory **UPP** in a DkCheck file.

To see how everything works in practice, we invite the reader to download the code and run `make running-example`, which translates our running example and produces a Dedukti file `output/running_example.dk` and an Agda file `agda_output/running-example.agda`. In order to test the tool with a more realistic example, the reader can also run `make test_agda`, which translates a proof of Fermat's little theorem from the Dedukti encoding of HOL [23] to **UPP**.

In the following, let us go through some important particularities of how the tool works.

User added constraints

As we have seen, our tool tries to compute the most general type for a definition or declaration to be typable. However, it is not always desirable to have the most general type, as shown by the following example.

▶ **Example 24.** Consider the local signature

$$\Delta = \mathit{Nat} : U_\square; \mathit{zero} : \mathit{El}_\square\ \mathit{Nat}; \mathit{succ} : \mathit{El}_\square\ (\mathit{Nat} \rightsquigarrow_{\square,\square} \mathit{Nat})$$

defining the natural numbers in **I**. The translation of this signature by our algorithm is

$$|\Delta| = \mathit{Nat} : \Pi i : \mathit{Level}.U_i; \mathit{zero} : \Pi i : \mathit{Level}.\mathit{El}_i\ (\mathit{Nat}\ i); \mathit{succ} : \Pi i\ j : \mathit{Level}.\mathit{El}_{(i \sqcup j)}\ ((\mathit{Nat}\ i) \rightsquigarrow_{i,j} (\mathit{Nat}\ j))$$

However, we normally would like to impose i to be equal to j in the type of *succ*, or even to impose *Nat* not to be universe polymorphic.

In order to solve this problem, we added to Predicativize the possibility of adding constraints by the user, in such a way that we can for instance impose *Nat* to be in U_z, or $i = j$ in the type of the successor. Adding constraints can also be useful to help the unification algorithm.

Rewrite rules

The algorithm that we presented and proved correct covers two types of entries: definitions and constants. This is enough for translating proofs written in higher-order logic or similar systems, in which every step either poses an axiom or makes a definition or proof.

However, when dealing with full-fledged type theories, such as those implemented by Coq or Matita, which also feature inductive types, it is customary to use rewrite rules to encode recursion and pattern matching. If we simply ignore these rules when performing the translation, we would run into problems as the entries that appear after may need those rewrite rules to typecheck.

Therefore, our implementation extends the presented algorithm and also translate rewrite rules. In order to do this, we use DkCheck's subject reduction checker to generate constraints and proceed similarly as in the algorithm. Because this feature is still work in progress, this step can require user intervention in some cases. In this case, the user has to manually add constraints over some symbols to help the translation.

Agda output

Predicativize produces files in **UPP**, which is a subsystem of the encoding of Agda. In order to translate these files to Agda itself, we also integrated in Predicativize a translator that performs a simple syntactical translation from the Agda encoding in Dedukti to Agda. For instance, `make test_agda_with_typecheck` translates Fermat's Little Theorem proof from HOL to Agda and typechecks it.

8 Translating Matita's arithmetic library to Agda

We now discuss how we used Predicativize to translate Matita's arithmetic library to Agda. The translation is summarized in Figure 7, where **DK**[X] stands for the encoding of system X in Dedukti.

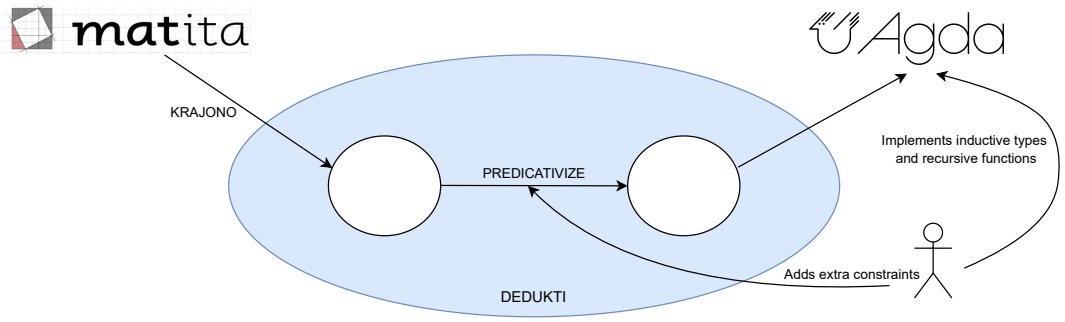


Figure 7 Diagram representing the translation of Matita's arithmetic library into Agda.

Matita's arithmetic library was already available in Dedukti thanks to Krajono [2], a translator from Matita to the encoding **DK**[Matita] in Dedukti. Therefore, the first step of the translation was already done for us.

Then, using Predicativize we translated the library from **DK**[Matita] to **DK**[Agda] (which is a supersystem of **UPP**). As the encoding of Matita's recursive functions uses rewrite rules, their translation required some user intervention to add constraints over certain symbols, as mentioned in the previous section. Once this step is done, the library is known to be predicative, as it typechecks in **DK**[Agda].

We then used Predicativize to translate these files to Agda files. However, because the rewrite rules in the Dedukti encoding cannot be translated to Agda, and given that they are needed for typechecking the proofs, the library does not typecheck directly.

Therefore, to finish our translation we had to define the inductive types and recursive functions manually in Agda. This step of our translation admittedly requires some time, however the effort is orders of magnitude less than rewriting the whole library in Agda, specially given that the great majority of the library is made of proofs, whose translations we did not need to change. Note that this manual step is not exclusive to our work as it is also needed in [23].

Defining inductive types also required us to add constraints. For instance, we saw in Example 24 that the successor symbol is translated as $\mathit{succ} : \Pi i\ j : \mathit{Level}.\mathit{El}_{(i \sqcup j)}\ ((\mathit{Nat}\ i) \rightsquigarrow_{i,j} (\mathit{Nat}\ j))$, but in order to be able to implement this symbol as a constructor of an inductive type, we need to impose $i = j$. If one then wishes to align Nat with the built-in type of natural numbers in Agda, we would also have to impose $i = \mathtt{z}$, which would then allow us to replace Nat by the built-in type in the result of the translation.

The result of this translation is available at `https://github.com/thiagofelicissimo/matita_lib_in_agda` and, as far as we know, contains the very first proofs in Agda of Bertrand's Postulate and Fermat's Little Theorem. It also contains a variety of other interesting results such as the Binomial Law, the Chinese Remainder Theorem, and the Pigeonhole Principle. Moreover, this library typechecks with Agda's `--safe` flag, attesting that it does not use any unsafe features.

9 Conclusion

We have tackled the problem of sharing proofs with predicative systems, by proposing an algorithm for it. Our implementation allowed to translate many non-trivial proofs from Matita's arithmetic library to Agda, showing that our algorithm works well in practice.

Our solution uses unification modulo arithmetic equivalence on universe levels. We designed an incomplete algorithm for this problem which is powerful enough for our needs. Still, one can wonder if there is an algorithm which always finds a most general solution when there is one, or if this problem is undecidable. One could improve our algorithm by using ACUI unification to solve constraints not containing `s`. However, there are problems with most general unifiers that would also not be handled by this extension. Agda also features an algorithm for solving level metavariables which uses an approach different from ours, but it does not seem to have been formalized in the literature. Therefore, the question if such an algorithm exists seems to be open.

For future work, we would also like to look at possible ways of making Predicativize less dependent on user intervention. In particular, the translation of inductive types and recursive functions involves some considerable manual work. Thus if we want to be able to translate larger libraries, there is definitely a need for automating this step.

References

1. Andrea Asperti and Wilmer Ricciotti. A proof of bertrand's postulate. *Journal of Formalized Reasoning*, 5(1):37–57, 2012.
2. Ali Assaf. *A framework for defining computational higher-order logics*. These, École polytechnique, September 2015. URL: `https://pastel.archives-ouvertes.fr/tel-01235303`.
3. Ali Assaf, Guillaume Burel, Raphaël Cauderlier, D Delahaye, G Dowek, C Dubois, F Gilbert, P Halmagrand, O Hermant, and R Saillard. Dedukti: a logical framework based on the λ π-calculus modulo theory. Unpublished, 2016.
4. H. P. Barendregt. *Lambda Calculi with Types*, pages 117–309. Oxford University Press, Inc., USA, 1993.
5. Michael Beeson, Julien Narboux, and Freek Wiedijk. Proof-checking Euclid. *Annals of Mathematics and Artificial Intelligence*, page 53, January 2019. `doi:10.1007/s10472-018-9606-x`.
6. F. Blanqui. Rewriting modulo in deduction modulo. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 2706, 2003. 15 pages. `doi:10.1007/3-540-44881-0_28`.
7. F. Blanqui. Definitions by rewriting in the calculus of constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005. `doi:10.1017/S0960129504004426`.
8. Frédéric Blanqui. *Théorie des types et réécriture. (Type theory and rewriting)*. PhD thesis, University of Paris-Sud, Orsay, France, 2001. URL: `https://tel.archives-ouvertes.fr/tel-00105522`.
9. Frédéric Blanqui. Type safety of rewrite rules in dependent types. In *5th International Conference on Formal Structures for Computation and Deduction*, 2020.
10. Frédéric Blanqui. Encoding type universes without using matching modulo AC. In *Proceedings of the 7th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics 228, 2022. `doi:10.4230/LIPIcs.FSCD.2022.24`.
11. Frédéric Blanqui, Gilles Dowek, Émilie Grienenberger, Gabriel Hondet, and François Thiré. Some axioms for mathematics. In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.FSCD.2021.20`.
12. Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications*, pages 102–117, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
13. Tristan Delort. Importer les preuves de Logipedia dans Agda. Internship report, Inria Saclay Ile de France, November 2020. URL: `https://hal.inria.fr/hal-02985530`.

14 Thiago Felicissimo. Adequate and Computational Encodings in the Logical Framework Dedukti. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, volume 228 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2022.25.

15 Gaspard Ferey. *Higher-Order Confluence and Universe Embedding in the Logical Framework*. These, Université Paris-Saclay, June 2021. URL: https://tel.archives-ouvertes.fr/tel-03418761.

16 Guillaume Genestier. Encoding agda programs using rewriting. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 31:1–31:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.31.

17 Yoan Géran. Euclid's elements book 1 in dedukti. URL: https://github.com/Karnaj/sttfa_geocoq_euclid [cited 2022].

18 Robert Harper and Robert Pollack. Type checking with universes. *Theor. Comput. Sci.*, 89(1):107–136, August 1991. doi:10.1016/0304-3975(90)90108-T.

19 R. Saillard. *Type checking in the Lambda-Pi-calculus modulo: theory and practice*. PhD thesis, Mines ParisTech, France, 2015. URL: https://pastel.archives-ouvertes.fr/tel-01299180.

20 Ronan Saillard. *Type checking in the Lambda-Pi-calculus modulo: theory and practice*. PhD thesis, PhD thesis, Mines ParisTech, France, 2015.

21 Matthieu Sozeau and Nicolas Tabareau. Universe polymorphism in coq. In *International Conference on Interactive Theorem Proving*, pages 499–514. Springer, 2014.

22 Agda Development Team. Agda 2.6.2.1 documentation. URL: https://agda.readthedocs.io/en/v2.6.2.1/index.html [cited 2022].

23 François Thiré. Sharing a library between proof assistants: Reaching out to the HOL family. In Frédéric Blanqui and Giselle Reis, editors, *Proceedings of the 13th International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP@FSCD 2018, Oxford, UK, 7th July 2018*, volume 274 of *EPTCS*, pages 57–71, 2018. doi:10.4204/EPTCS.274.5.

24 François Thiré. *Interoperability between proof systems using the logical framework Dedukti*. PhD thesis, ENS Paris-Saclay, 2020.

A Typing rules for Dedukti and basic metaproperties

We recall the following basic metaproperties of Dedukti. Proofs can be found in [8, 19].

▶ **Theorem 25** (Basic metaproperties)**.**

1. *Weakening: If* $\Sigma;\Gamma \vdash M : A$, $\Gamma \subseteq \Gamma'$ *and* $\Sigma;\Gamma'$ `well-formed` *then* $\Sigma;\Gamma' \vdash M : A$
2. *Substitution Lemma: If* $\Sigma;\Gamma, x : B, \Gamma' \vdash M : A$ *and* $\Sigma;\Gamma \vdash N : B$ *then* $\Sigma;\Gamma,\Gamma'\{N/x\} \vdash M\{N/x\} : A\{N/x\}$
3. *Well-sortedness: If* $\Sigma;\Gamma \vdash M : A$ *then either* $A = \mathbf{Kind}$ *or* $\Sigma;\Gamma \vdash A : \mathbf{s}$ *for* $\mathbf{s} = \mathbf{Type}$ *or* $\mathbf{Kind}$.
4. *Subject reduction of* δ*: If* $\Sigma;\Gamma \vdash M : A$ *and* $M \hookrightarrow_\delta M'$ *then* $\Sigma;\Gamma \vdash M' : A$
5. *Subject reduction of* β*: If injectivity of dependent product holds, then* $\Sigma;\Gamma \vdash M : A$ *and* $M \hookrightarrow_\beta M'$ *implies* $\Sigma;\Gamma \vdash M' : A$.
6. *Contexts are well typed: If* $x : A \in \Gamma$ *then* $\Sigma;\Gamma \vdash A : \mathbf{Type}$
7. *Signatures are well typed: If* $c : A \in \Sigma$ *then* $\Sigma;- \vdash A : \mathbf{s}$ *and if* $c : A := M \in \Sigma$ *then* $\Sigma;- \vdash M : A$
8. *Inversion of typing: Suppose* $\Sigma;\Gamma \vdash M : A$
   - *If* $M = x$ *then* $x : A' \in \Gamma$ *and* $A \equiv A'$

$$\frac{}{-;-\ \texttt{well-formed}}\ \texttt{Empty} \qquad c \notin \Sigma\ \frac{\Sigma;- \vdash A : \mathbf{s}}{\Sigma, c : A;-\ \texttt{well-formed}}\ \texttt{Decl-cons}$$

$$c \notin \Sigma\ \frac{\Sigma;- \vdash M : A}{\Sigma, c : A := M;-\ \texttt{well-formed}}\ \texttt{Decl-def} \qquad x \notin \Gamma\ \frac{\Sigma;\Gamma \vdash A : \mathbf{Type}}{\Sigma;\Gamma, x : A\ \texttt{well-formed}}\ \texttt{Decl-var}$$

$$c : A \text{ or } c : A := M \in \Sigma\ \frac{\Sigma;\Gamma\ \texttt{well-formed}}{\Sigma;\Gamma \vdash c : A}\ \texttt{Cons} \qquad x : A \in \Gamma\ \frac{\Sigma;\Gamma\ \texttt{well-formed}}{\Sigma;\Gamma \vdash x : A}\ \texttt{Var}$$

$$\frac{\Sigma;\Gamma\ \texttt{well-formed}}{\Sigma;\Gamma \vdash \mathbf{Type} : \mathbf{Kind}}\ \texttt{Sort} \qquad A \equiv B\ \frac{\Sigma;\Gamma \vdash M : A \qquad \Sigma;\Gamma \vdash B : \mathbf{s}}{\Sigma;\Gamma \vdash M : B}\ \texttt{Conv}$$

$$\frac{\Sigma;\Gamma, x : A \vdash B : \mathbf{s}}{\Sigma;\Gamma \vdash \Pi x : A.B : \mathbf{s}}\ \texttt{Prod} \qquad \frac{\Sigma;\Gamma \vdash M : \Pi x : A.B \qquad \Sigma;\Gamma \vdash N : A}{\Sigma;\Gamma \vdash MN : B\{N/x\}}\ \texttt{App}$$

$$\frac{\Sigma;\Gamma, x : A \vdash B : \mathbf{s} \qquad \Sigma;\Gamma, x : A \vdash M : B}{\Sigma;\Gamma \vdash \lambda x : A.M : \Pi x : A.B}\ \texttt{Abs}$$

Figure 8 Typing rules for Dedukti.

- *If* $M = c$ *then* $c : A' \in \Sigma$ *and* $A \equiv A'$
- *If* $M = \mathbf{Type}$ *then* $A \equiv \mathbf{Kind}$
- $M = \mathbf{Kind}$ *is impossible*
- *If* $M = \Pi x : A_1.A_2$ *then* $\Sigma;\Gamma, x : A_1 \vdash A_2 : \mathbf{s}$ *and* $\mathbf{s} \equiv A$
- *If* $M = M_1M_2$ *then* $\Sigma;\Gamma \vdash M_1 : \Pi x : A_1.A_2$, $\Sigma;\Gamma \vdash M_2 : A_1$ *and* $A_2\{M_2/x\} \equiv A$
- *If* $M = \lambda x : B.N$ *then* $\Sigma;\Gamma, x : B \vdash C : \mathbf{s}$, $\Sigma;\Gamma, x : B \vdash N : C$ *and* $A \equiv \Pi x : B.C$

A Normalized Edit Distance on Infinite Words

Dana Fisman
Dept. of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Joshua Grogin
Dept. of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Gera Weiss
Dept. of Computer Science, Ben-Gurion University, Beer-Sheva, Israel

Abstract

We introduce $\overline{\omega}$-NED, an edit distance between infinite words, that is a natural extension of NED, the normalized edit distance between finite words. We show it is a metric on (equivalence classes of) infinite words. We provide a polynomial time algorithm to compute the distance between two ultimately periodic words, and a polynomial time algorithm to compute the distance between two regular ω-languages given by non-deterministic Büchi automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Pattern matching; Hardware → Robustness

Keywords and phrases Edit Distance, Infinite Words, Robustness

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.20

Related Version `https://jgrogin.github.io/A_Normalized_Edit_Distance_on_Finite_and_Infinite_Words_thesis.pdf`

Funding This work was supported in part by ISF grants 2714/19 and 2507/21.

1 Introduction

Quantifying distances between words is a field of research that provides tools for measuring semantic differences between sequential objects and sets thereof. In this work, we are interested in quantifying the distances between infinite words and between sets of infinite words. The main motivation that led us to examine this issue is the use of infinite words in formal methods for describing behaviors of reactive systems. In particular, in software verification and other applications, it is common to refer to software systems as state machines and to use automata whose languages are the sets of runs of the systems. Specifically, it is customary to specify requirements by automata representing the sets of allowed/desired runs. We argue that in this context, it is natural to define the distance between two runs as the average amount of "discrepancies" between the two runs and to study algorithmic problems related to those distances. Since we deal with infinite words, a description of a set of words by an automaton usually uses the Büchi acceptance condition, where a word is accepted by an automaton if and only if there is a trajectory in the automaton that reads the word and passes through accepting states infinitely many times. Sets of infinite words that can be described by Büchi automata are called ω-regular languages.

Significance of our contribution. Verification tools, which deal primarily with compliance and non-compliance with requirements defined using temporal logic, provide a yes/no answer but do not quantify the degree of deviation of implementations from the requirements nor the robustness of an implementation. In this work, we present a metric for quantifying distances between infinite words reflecting deviations between runs of implementation and runs specified in the requirements. Specifically, we extend a normalized version of the known

edit distance for this purpose and describe algorithms for calculating distances between ultimately periodic words and between ω-regular languages. The distance between two languages L_1 and L_2 is defined as the infimum between any pair of words w_1, w_2 in L_1 and L_2, respectively.

It can be used, for example, to measure the *robustness of an implementation* [1,2,6,16,17]. Consider a system S implementing a specification φ. The system is considered robust if it continues to satisfy the specification, even when errors disrupt its normal behavior. That is, robustness asks what is the minimum number of errors that would render the system to violate the specification. Thus, robustness is taken to be the infimum between any pair of words w, w' in $L(S)$ and $L(\neg\varphi)$, which is exactly the distance between the language of S and the language of $\neg\varphi$.

Relations to other notions of distance between words. Our work is based on the well-known *edit distance* (aka *Levenshtein* distance) [13]. For two finite words $u_1, u_2 \in \Sigma^*$, this distance, denoted $\text{ED}(u_1, u_2)$, is defined as the minimum number of edits we need to apply to u_1 to obtain u_2. Here, edit operations are *delete* a letter, *insert* a letter, or *replace* a letter. For instance, $\text{ED}(aabcde, abpcg) = 4$ since by deleting the first a, inserting p, replacing d by g and deleting e we get from the first word to the second, and there is no shorter sequence of edit operations transforming the first word to the latter. In the setting of this paper, differences between words model violations of specifications. We, therefore, treat all types of differences equally, i.e., while in some applications, different edit operations may weigh differently, we consider the case of uniform weights for all edit operations.

Since we are interested in infinite words, we consider normalized versions of ED. To see the role of normalization, even for finite terms, note that $\text{ED}(a^{98}b^4, a^{100})$ is also four, even though these words are almost identical. Note that the distance remains four even if we replace 100 by 10^6 and 98 by $10^6 - 2$ in which case the words are even more identical. Intuitively, the problem is that we need to count error rates, not just numbers, if we want to compare words of different, even infinite, lengths. To achieve normalization, people have considered dividing ED by the sum, max, or min of the lengths of the words, but these do not satisfy the triangle inequality [5,15]. To bypass this problem, Marzal and Vidal [15] proposed to divide ED by the length of the sequence of operations transforming the first word to the second, *the edit path*, as formally defined in Section 2. Using NED (for *normalized edit distance*) to refer to this function, we get that $\text{NED}(aabcde, abpcg) = 4/7$ and $\text{NED}(a^{98}b^4, a^{100}) = 4/102$, which better reflects the "average" number of required edit operations, i.e., this captures a notion of "error rates" as needed. In their paper, Marzal and Vidal demonstrated that NED is not necessarily a metric when the weights are not uniform. The question of whether it is a metric when costs are uniform remained unsettled. This led others to propose alternative notions such as the *generalized normalized edit distance* [14] and the *contextual edit distance* [5]. Still, because of its simplicity and based on empirical data that showed that it behaves very close to a metric, it is widely used in applications. In this paper, we rely on a recent work [7] that established that NED is indeed a metric when the weights are uniform (all edit operations cost the same). The paper above also lists properties of NED and the other two normalized edit distances demonstrating that NED is more suitable when considering the edit operations as errors.

We turn now to discuss possibilities for distances between infinite words. The most famous distance function on infinite words is the one on which the Cantor topology is defined, according to which the distance between $w_1, w_2 \in \Sigma^\omega$ decreases exponentially with the length of the longest common prefix [10]. Formally, using CTD to denote this distance function, $\text{CTD}(w_1, w_2)$ is zero if $w_1 = w_2$ and otherwise it is $2^{-\min\{i \mid w_1[i] \neq w_2[i]\}}$ where $w[i]$ denotes

the ith letter of w starting from 0. The intuition behind CTD and the edit distance functions mentioned above (ED and NED) is very different. We have that $\text{CTD}(ab^\omega, a^\omega) = 1/2$ and $\text{CTD}(ba^\omega, a^\omega) = 1$ while more edit operations are needed to get from ab^ω to a^ω than from ba^ω (infinitely many operations are required in the former and only one in the latter). Other commonly used distance functions for infinite words are defined using some weight function and some summation function defined on that (see, e.g., [3, 4]). This is more similar in spirit to our motivation. Formally, for $w_1, w_2 \in \Sigma^\omega$ we define their weight difference sequence as $\eta(w_1, w_2) = e_1, e_2, \ldots$ where $e_i = 0$ if $w_1[i] = w_2[i]$ and $e_i = W(w_1[i], w_2[i])$ for some weight function $W\colon \Sigma^2 \to \mathbb{R}$ otherwise. In the examples, we assume, for simplicity, that W assigns the number 1 to all pairs of distinct letters. Given an infinite sequence $\eta = e_1, e_2, e_3, \ldots$ with $e_i \in \mathbb{R}$ the common summation functions are defined as follows:

$$\begin{aligned}
\mathit{Sup}(\eta) &= \sup_{n\in\mathbb{N}} e_n & \mathit{LimSup}(\eta) &= \limsup_{n\to\infty} e_n = \lim_{n\to\infty} \sup_{m\geq n} e_m \\
\mathit{Inf}(\eta) &= \inf_{n\in\mathbb{N}} e_n & \mathit{LimInf}(\eta) &= \liminf_{n\to\infty} e_n = \lim_{n\to\infty} \inf_{m\geq n} e_m \\
\mathit{LimSupAvg}(\eta) &= \limsup_{n\to\infty} \tfrac{1}{n}\textstyle\sum_{i=1}^{n} e_i & \mathit{Disc}_\lambda(\eta) &= \textstyle\sum_{n=1}^{\infty} \lambda^n e_n \\
\mathit{LimInfAvg}(\eta) &= \liminf_{n\to\infty} \tfrac{1}{n}\textstyle\sum_{i=1}^{n} e_i
\end{aligned}$$

For finite words *Sum*, *Min*, *Max*, *Avg* and Disc_λ, for $\lambda \in (0, 1)$, are also used see, e.g., [6]. Different summation functions are called for in different situations. For instance, *Sup* is used for peak consumption; *LimSupAvg*, *LimInfAvg* for average response time or rate of failures; and Disc_λ when late failures are less important than early ones.

We next demonstrate why *Sup*, *Inf*, *LimSup*, *LimInf*, and Disc_λ do not capture the notion of distance we seek in this paper. Applying *Sup* to $\eta(w_1, w_2)$ would give 1 if $w_1 \neq w_2$ and 0 otherwise. Applying *Inf* to $\eta(w_1, w_2)$ would give 1 only if $w_1[i] \neq w_2[i]$ for every i, i.e., they completely disagree. These two are too coarse to quantify the closeness of implementations to specifications. Applying *LimSup* to $\eta(w_1, w_2)$ would give 1 if there are infinitely many indices i in which $w_1[i] \neq w_2[i]$ and 0 otherwise, and *LimInf* would give 1 if there is i such that $w_1[j] \neq w_2[j]$ for every $j > i$. These are still too coarse for our purpose. The summation Disc_λ disregards what happens ad infinitum, or more precisely, gives later events an exponentially smaller weight making them negligible. Even for words that agree on the suffixes, e.g., $w_1 = a^\omega$, $w_2 = aba^\omega$, $w_3 = aaaba^\omega$ we get that $\mathit{Disc}_{\frac{1}{2}}(\eta(w_1, w_2)) = 1/4$ and $\mathit{Disc}_{\frac{1}{2}}(\eta(w_1, w_3)) = 1/16$ though both w_2 and w_3 have one discrepancy compared to w_1.

The summation functions closest to what we seek are *LimSupAvg* and *LimInfAvg*. Indeed, if we consider the words $w_1 = a^\omega$ and $w_2 = (aaaab)^\omega$, applying the uniform weight function, we obtain the sequence $\eta = (00001)^\omega$ since every fifth letter is different. Thus, $\mathit{LimSupAvg}(\eta) = 1/5$, which is consistent with our intuition on the normalized number of edit operations required to apply to w_1 to obtain w_2. For the words $v_1 = c^{100}a^\omega$ and $v_2 = d^{35}(aaaab)^\omega$ we also get the desired $1/5$ by applying *LimSupAvg* on $\eta(v_1, v_2)$ which is $1^{100}(00001)^\omega$. Indeed, *LimSupAvg* (and *LimInfAvg*) is indifferent to any finite prefix, as we expect the case to be since a finite prefix is always negligible compared to the infinite suffix. Still, *LimSupAvg* and *LimInfAvg* do not always correspond to our intuition of the normalized number of edits required to get from one word to the other. Consider $x_1 = (abc)^\omega$ and $x_2 = (acb)^\omega$. We have that $\eta(x_1, x_2) = (011)^\omega$ so *LimInfAvg* is 2/3. But if we consider edit operations, we can transform *abcabc* to *acbacb* using an edit path of length eight with two delete operations and two insert operations hence $\text{NED}(x_1x_1, x_2x_2)$=4/8=1/2 meaning the actual error rate should be 1/2 rather than 2/3. Another issue arises when considering, for example, $y_1 = (abcd)^\omega$ and $y_2 = (bcda)^\omega$. The obtained sequence $\eta(y_1, y_2)$ is $(1)^\omega$ thus *LimSupAvg* and *LimInfAvg* result in 1, meaning the words are farthest apart, while the number of edits required to get from y_1 to y_2 is one, since we can simply drop the first letter of y_1 to get y_2. Since this is one edit out of infinitely many letters we expect the error rate to be 0 in this case.

Desired criteria from an edit distance on infinite words. We want it to be *normalized* in the sense that the distance between two words is in $[0,1]$ and that it reflects the number of edits needed on average to get from one word to the other. In particular, it would be nice if we can find such a metric in which the distance between $(u_1)^\omega$ and $(u_2)^\omega$ would be close to $\text{NED}(u_1,u_2)$. We would also like it to return zero for words with a common infinite suffix. Do we want to require that the distance between two words is zero if and only if they have a common suffix? While this makes sense when considering ultimately periodic words, there are more cases where we would like the distance to be zero when we consider arbitrary words. Consider, e.g., $w_1 = a^\omega$ and $w_2 = ba^9ba^{99}ba^{999}b\cdots$. That is, w_2 has a in almost all positions, but b's creep in intervals of powers of 10. We expect the distance between w_1 and w_2 to be 0 since the normalized number of required edits is negligible because the necessity for an edit operation diminishes as the word progresses.

Due to space limitations, some proofs are deferred to the full version; they can also be found in the second author's master thesis [9].

2 Preliminaries

Sequences, sub-sequences, repetitions, projection. We use $[i..j]$ to denote the set $\{i, i+1,\ldots,j\}$ for naturals i,j such that $i \le j$. If $\rho = (r_0, r_1, \ldots)$ is a sequence, we use $\rho[i..j]$ for the subsequence $(r_i, r_{i+1}, \ldots, r_j)$. Similarly, $\rho[i]$ is used to denote r_i, and $\rho[..i]$ (resp. $\rho[i..]$) is used for the prefix (resp. suffix) of ρ ending (resp. starting) at i. If $\rho = (r_0, r_1, \ldots r_{l-1})$ is a sequence, we use ρ^k for the k-times repetition of ρ and ρ^ω for the infinite repetition of ρ. Then, if l is the length of ρ, we have that $\rho^\omega[i] = \rho[i \bmod l]$ for every $i \in \mathbb{N}$. Given a tuple $t = \langle a_1, a_2, \ldots, a_n\rangle$ we use $\pi_i(t)$ for a_i, namely the projection of t on the i-th coordinate. These notions extend to sets and sequences in the usual manner, thus, e.g., given a sequence $\rho{=}(\langle\sigma_1,\sigma'_1\rangle, \langle\sigma_2,\sigma'_2\rangle \ldots)$ we use $\pi_1(\rho)$ for the sequence $(\sigma_1, \sigma_2, \ldots)$.

Words, ω-words, ultimately periodic words, rotations. An *alphabet* Σ is a finite non-empty set of symbols. A finite sequence over Σ is a word and an infinite sequence over Σ is an ω-word. We use $|w|$ to denote the length of w. Thus, $|w| = l$ if $w = \sigma_0\sigma_1\ldots\sigma_{l-1}$ and $|w| = \omega$ if w is an ω-word. An ω-word w is termed *ultimately periodic* if $w = uv^\omega$ for some $u \in \Sigma^*$ and $v \in \Sigma^+$. An ultimately periodic word $w = u(v)^\omega$ can be finitely represented as the pair (u,v). The set of finite words is denoted Σ^*, the set of infinite words is denoted Σ^ω, their union is denoted Σ^∞. The set of ultimately periodic words over Σ is denoted Σ^{UP}. Let $w \in \Sigma^*$, we use $rot(w)$ for all rotations of w, namely, the set of words uv such that $w = vu$.

Directions and Paths. We consider a set $\mathbb{D} = \{(0,1),(1,0),(1,1)\}$ of three *directions*. The element $(0,1)$ denotes the *south* direction, and is abbreviated as d_{S}; the element $(1,0)$ the *east* direction, and is abbreviated as d_{E}; and the element $(1,1)$ the *south-east* direction, and is abbreviated as d_{SE}. A sequence $\rho \in \mathbb{D}^\infty$ is called a *path*.

A South-East Graph, Endpoint. A *south-east graph* is composed of a set $V = [0..n_1]\times[0..n_2]$ of vertices, for some $n_1, n_2 \in \mathbb{N}\cup\{\omega\}$, and a set $E = (E_{\text{S}} \cup E_{\text{E}} \cup E_{\text{SE}}) \cap V^2$ of edges, where $E_{\text{S}} = \{(\langle i,j\rangle, \langle i, j+1\rangle)\}$, $E_{\text{E}} = \{(\langle i,j\rangle, \langle i+1, j\rangle)\}$, $E_{\text{SE}} = \{(\langle i,j\rangle, \langle i+1, j+1\rangle)\}$. We refer to such a southeast graph as *an* $(n_1 \times n_2)$*-south-east graph*. The vertices of the graph can be placed on a $[0..n_1]\times[0..n_2]$ grid. We visualize the top-left position as the $\langle 0,0\rangle$ point. Thus, a path $\rho = (d_0,\ldots,d_{l-1})$ starting in $\langle 0,0\rangle$ will end in $\sum_{i=0}^{l-1} d_i$ where summation is coordinate wise. We use $endpoint(\rho)$ to denote the endpoint of path ρ starting at $\langle 0,0\rangle$.

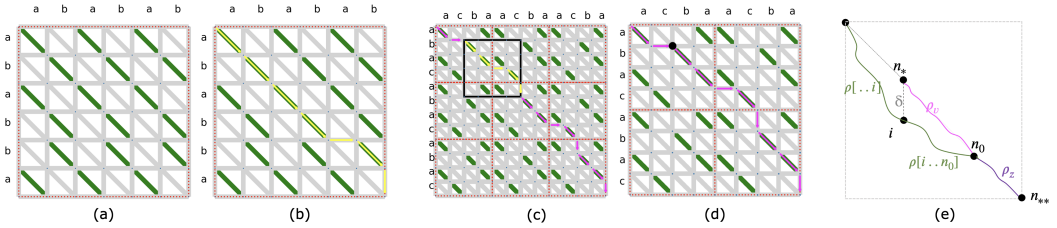


Figure 1 **(a)** the words graph $\mathcal{G} = \mathcal{G}(ababab, ababba)$, gray edges are assigned the weight 1, green edges are assigned the weight 0. **(b)** an edit path ρ in $\mathcal{G}$ (in yellow) where $endpoint(\rho) = (6, 6)$, $wgt(\rho) = 2$, $len(\rho) = 7$, and $cost(\rho) = 2/7$. **(c)** an edit path ρ and points $s = (2, 1)$, $t = (6, 5)$ on $\mathcal{G}((acba)^3, (abac)^3)$ where $\rho[s..t]$ (in yellow) fits a 4-square. **(d)** the path $\rho' = \rho_s \cdot \rho_t$ obtained by removing from ρ the sub-path $\rho[s..t]$ (and replacing it by a black dot) as in the proof of Prop. 9. **(e)** notations for proof of Theorem 30.

Words Graph. Given two words $w_1, w_2 \in \Sigma^\infty$ their corresponding graph $\mathcal{G}(w_1, w_2)$ is the weighted graph (G, θ) where $G = (V, E)$ is the $(|w_1| \times |w_2|)$-south-east graph and the weight of edges $\theta\colon E \to \{0, 1\}$ is defined as follows

$$\theta(e) = \begin{cases} 0 & \text{if } e{=}(v, v') \text{ for } v{=}\langle i, j\rangle, w_1[i]{=}w_2[j], v'{-}v{=}d_{\text{SE}}, \\ 1 & \text{otherwise} \end{cases}$$

That is, the weight of all east edges and south edges is 1 and the weight of a south-east edge starting at $\langle i, j\rangle$ is 0 if the letters $w_1[i]$ and $w_2[j]$ are the same, and 1 otherwise. The word graph $\mathcal{G}(ababab, ababba)$ is given in Figure 1 (a).

Edit Path. A sequence $\rho \in \mathbb{D}^*$ is termed an *edit path* for (w_1, w_2) if $endpoint(\rho) = (|w_1|, |w_2|)$. The *weight* of ρ, denoted $wgt(\rho)$, is the sum of weights of the corresponding edges in $\mathcal{G}(w_1, w_2)$. Formally, if $\rho = (d_0, d_1, \ldots, d_{l-1})$ then the traversed edges are $(e_0, e_1, \ldots, e_{l-1})$ where $e_i = (s_{i-1}, s_i)$, $s_{-1} = \langle 0, 0\rangle$ and $s_i = endpoint(\rho[..i])$ for $0 \le i < l$. Hence, we can define $wgt(\rho) = \sum_{i=0}^{l-1} \theta(e_i)$. We use the notation $len(\rho)$ to denote the length $|\rho|$ of ρ. The *cost* of ρ, denoted $cost(\rho)$, is defined to be $\frac{wgt(\rho)}{len(\rho)}$. See Figure 1 (b). Intuitively, an edit path for (w_1, w_2) prescribes how to transform w_1 into w_2. In particular, a south direction from $\langle i, j\rangle$ marks that letter $w_1[i]$ is deleted, an east direction from $\langle i, j\rangle$ marks that letter $w_2[j]$ is added, a south-east direction from $\langle i, j\rangle$ marks substitution of $w_1[i]$ by $w_2[j]$, thus it costs nothing if $w_1[i] = w_2[j]$.

Edit Distance and the Normalized Edit Distance. Using the above notations we provide the formal definitions of the *edit distance* and the *normalized edit distance*. The (not normalized) *edit distance* of $u_1, u_2 \in \Sigma^*$, denoted $\text{ED}(u_1, u_2)$, is the minimum weight of an edit path for (u_1, u_2). That is,

$$\text{ED}(u_1, u_2) = \min\{wgt(\rho) \mid \rho \text{ is an edit path for } (u_1, u_2)\}.$$

The *normalized edit distance* of two finite words $u_1, u_2 \in \Sigma^*$, denoted $\text{NED}(u_1, u_2)$, is the minimum cost of an edit path for (u_1, u_2). That is,

$$\text{NED}(w_1, w_2) = \min\{cost(\rho) \mid \rho \text{ is an edit path for } (u_1, u_2)\}.$$

Since we are interested in the NED metric, we say that an edit path ρ for (u_1, u_2) is *optimal* if $\text{NED}(u_1, u_2) = cost(\rho)$.

A Metric Space. A metric space is an ordered pair $(\mathbb{M}, d)$ where $\mathbb{M}$ is a set and $d\colon \mathbb{M}\times\mathbb{M} \to \mathbb{R}$ is a *metric*, i.e., it satisfies the following for all $m_1, m_2, m_3 \in \mathbb{M}$:

1. $d(m_1, m_2) = 0$ iff $m_1 = m_2$;
2. $d(m_1, m_2) = d(m_2, m_1)$;
3. $d(m_1, m_3) \leq d(m_1, m_2) + d(m_2, m_3)$.

The first condition is referred to as *identity of indiscernibles*, the second as *symmetry*, and the third as the *triangle inequality*.

▶ **Theorem 1** ([7]). *(Σ^*, NED) is a metric space.*

3 A Normalized Edit Distance for Infinite Words

Intuitively, it makes sense to define the *normalized edit distance* for two infinite words as the limit of NED of their prefixes. Since the limit may not exist, we define two candidate versions, one using lim inf and one using lim sup as follows.

▶ **Definition 2** ($\overline{\omega}$-NED,$\underline{\omega}$-NED). *Let $w_1, w_2 \in \Sigma^\omega$ be two infinite words. We define two candidate notions of a normalized edit distance, as follows:*

$$\overline{\omega}\text{-}NED(w_1, w_2) \stackrel{\text{def}}{=\!=} \limsup_{i\to\infty} NED(w_1[..i], w_2[..i])$$
$$\underline{\omega}\text{-}NED(w_1, w_2) \stackrel{\text{def}}{=\!=} \liminf_{i\to\infty} NED(w_1[..i], w_2[..i])$$

Since for every pair of finite words u_1, u_2, $\text{NED}(u_1, u_2)$ is bounded (between 0 and 1) both $\overline{\omega}\text{-NED}(w_1, w_2)$ and $\underline{\omega}\text{-NED}(w_1, w_2)$ converge for every pair w_1, w_2 of infinite words.

▶ **Example 3.**

a. $\overline{\omega}\text{-NED}(a^\omega,\ (aaaab)^\omega) = 1/5$
$\underline{\omega}\text{-NED}(a^\omega,\ (aaaab)^\omega) = 1/5$

b. $\overline{\omega}\text{-NED}(a^\omega,\ a\cdot b^1\cdot a\cdot b^2\cdot a\cdot b^3\cdot a\cdot b^4\cdots) = 1$
$\underline{\omega}\text{-NED}(a^\omega,\ a\cdot b^1\cdot a\cdot b^2\cdot a\cdot b^3\cdot a\cdot b^4\cdots) = 1$

c. $\overline{\omega}\text{-NED}(a^\omega,\ a^1\cdot b^1\cdot a^2\cdot b^2\cdot a^4\cdot b^4\cdots) = 1/2$
$\underline{\omega}\text{-NED}(a^\omega,\ a^1\cdot b^1\cdot a^2\cdot b^2\cdot a^4\cdot b^4\cdots) = 1/3$

Example 3 (c) shows that $\overline{\omega}$-NED and $\underline{\omega}$-NED, in general, may converge to different numbers. Since our motivation is to quantify errors, the worst-case view reflected by $\overline{\omega}$-NED seems more appropriate. Moreover, as we show next, $\overline{\omega}$-NED satisfies the triangle inequality while $\underline{\omega}$-NED does not.

▶ **Proposition 4.** *$\overline{\omega}$-NED satisfies the triangle inequality and $\underline{\omega}$-NED does not.*

Proof. We show that $\overline{\omega}$-NED satisfies the triangle inequality by proving a more general claim stating that given a function $d\colon \Sigma^* \times \Sigma^* \to \mathbb{R}_+$ satisfying the triangle inequality then the function $\overline{\omega}\text{-}d\colon \Sigma^\omega \times \Sigma^\omega \to \mathbb{R}_+$ where $\overline{\omega}\text{-}d(w_1, w_2)$ is defined by $\limsup_{i\to\infty} d(w_1[..i], w_2[..i])$ satisfies the triangle inequality as well. To see why this holds, let $w_1, w_2, w_3 \in \Sigma^\omega$. We have

$$\begin{aligned}
\overline{\omega}\text{-}d(w_1, w_3) &= \limsup_{i\to\infty} d(w_1[..i], w_3[..i]) \\
&\leq \limsup_{i\to\infty} \left(d(w_1[..i], w_2[..i]) + d(w_2[..i], w_3[..i])\right) \\
&\leq \limsup_{i\to\infty} d(w_1[..i], w_2[..i]) + \limsup_{i\to\infty} d(w_2[..i], w_3[..i]) \\
&= \overline{\omega}\text{-}d(w_1, w_2) + \overline{\omega}\text{-}d(w_2, w_3)
\end{aligned}$$

where the first inequality holds since d satisfies the triangle inequality and the second inequality is a property of sum of lim sup of non-negative sequences.

To see that $\underline{\omega}$-NED does not satisfy the triangle inequality, take $w_1 = a^\omega$, $w_3 = b^\omega$ and $w_2 = u_1v_1u_2v_2\cdots$, where $u_1 = a$, $v_1 = bb$, $u_2 = aaa$ and $v_{i+1} = b^{2|u_i|}$ for $i \geq 1$ and $u_{i+1} = a^{2|v_i|}$ for $i \geq 2$. Thus $w_2 = ab^2a^3b^6a^{12}b^{24}a^{48}b^{96}\ldots$. Note that after reading u_i the number of a's is twice the number of b's, and two-thirds of the total number of letters (and the number of b's is a third of the total number of letters), and likewise after reading v_i the number of b's is twice the number of a's, and two-thirds of the total number of letters (and the number of a's is a third of the total number of letters). We get that $\underline{\omega}\text{-NED}(w_1, w_2) = 1/3 = \underline{\omega}\text{-NED}(w_2, w_3)$, which contradicts the triangle inequality since $\underline{\omega}\text{-NED}(w_1, w_3) = 1 > 1/3 + 1/3 = \underline{\omega}\text{-NED}(w_1, w_2) + \underline{\omega}\text{-NED}(w_2, w_3)$. ◀

In order for $\overline{\omega}$-NED to be a metric it also needs to satisfy symmetry (which clearly it does) and the condition of identity of indiscernibles. The following example shows that $\overline{\omega}$-NED does not satisfy identity of indiscernibles: $\overline{\omega}\text{-NED}((ab)^\omega, (ba)^\omega) = 0$ though these are non-identical words. However, as per the discussion in the introduction, we do want to allow zero distance between such words. Specifically, we want the metric to be defined on equivalence classes of words so that the distance between two words is zero if and only if they are in the same equivalence class. To define this equivalence relation, let us revisit the examples where a distance of zero is expected. The first examples considered words that have a common suffix. Indeed, in such pairs of words, the number of required operations is finite (can be used to eliminate both prefixes), and thus negligible compared to the length of infinite words. The last example was $w_1 = a^\omega$ and $w_2 = ba^9ba^{99}ba^{999}b\cdots$. In this example, we view the number of edits as negligible since the necessity for edit operations diminishes as the work progresses. In this example, the number of required edits decreases exponentially. Should this be a requirement? What if it decreases quadratically or logarithmically? Observe that we can create words where the number of edits from a^ω decreases as slowly as desired by considering $w = (ab)^{n_1}(aab)^{n_2}(aaab)^{n_3}\cdots$. The larger $n_1, n_2, n_3, \ldots$ are, the slower the number of edits decreases. Still, in all such words, it diminishes over the infinite word and thus we expect the difference from a^ω to be zero.

We therefore define two infinite words w_1, w_2 to be *almost equal*, denoted $w_1 \equiv w_2$, if $\lim_{i\to\infty} \text{NED}(w_1[..i], w_2[..i]) = 0$. Note that words that have a common suffix are almost equal according to this definition, as are the other discussed examples. With respect to the equivalence classes of $\equiv$, identity of indiscernibles holds for $\overline{\omega}$-NED as formally stated in Prop. 5.

▶ **Proposition 5.** *$\overline{\omega}$-NED$(w_1, w_2) = 0$ iff $w_1 \equiv w_2$.*

Proof. If $w_1 \equiv w_2$ then $\lim_{i\to\infty} \text{NED}(w_1[..i], w_2[..i]) = 0$. Thus $\limsup_{i\to\infty} \text{NED}(w_1[..i], w_2[..i]) = 0$. Therefore, by definition $\overline{\omega}\text{-NED}(w_1, w_2) = 0$.

If $\overline{\omega}\text{-NED}(w_1, w_2) = 0$ then $\limsup_{i\to\infty} d(w_1[..i], w_2[..i]) = 0$. Since $\text{NED}(v_1, v_2)$ is bounded between 0 and 1 for any $v_1, v_2 \in \Sigma^*$, it follows that $\liminf_{i\to\infty} d(w_1[..i], w_2[..i]) = 0$. Hence $\lim_{i\to\infty} \text{NED}(w_1[..i], w_2[..i]) = 0$ implying $w_1 \equiv w_2$. ◀

Thus $\overline{\omega}$-NED satisfies the three conditions of being a metric on the space $\Sigma^\omega/\equiv$.

▶ **Theorem 6.** *$(\Sigma^\omega/\equiv, \overline{\omega}\text{-NED})$ is a metric space.*

We therefore henceforth focus on $\overline{\omega}$-NED.

4 The Case of Ultimately Periodic Words

We turn to discuss *ultimately periodic words*. The interest in ultimately periodic words stems from the fact that (a) they allow a finite representation of an infinite word, so we can ask whether we can compute $\overline{\omega}$-NED for such words; (b) deterministic finite state machines generate ultimately periodic words and; (c) two regular ω-languages are equivalent iff they agree on the set of ultimately periodic words.

We next show that $\overline{\omega}$-NED for ultimately periodic words can be computed using NED on the best rotations of the periodic parts:

▶ **Definition 7** (best rotation). *Let $u_1, u_2 \in \Sigma^+$. Let n be the least common multiple of $|u_1|$ and $|u_2|$. Let $u_1' = u_1^\omega[..n]$ and $u_2' = u_2^\omega[..n]$. We say that (v_1, v_2) is a* best rotation *for (u_1, u_2) if $v_1 \in rot(u_1')$, $v_2 \in rot(u_2')$ and for every $v_1' \in rot(u_1')$ and $v_2' \in rot(u_2')$ it holds that $NED(v_1, v_2) \leq NED(v_1', v_2')$. If (v_1, v_2) is a best rotation for some (u_1, u_2) we say that (v_1, v_2) is a* best rotation pair. *The* size *of such a best rotation is defined to be n.*

▶ **Theorem 8** ($\overline{\omega}$-NED for ultimately periodic words). *Let $w_1 = z_1 u_1^\omega$ and $w_2 = z_2 u_2^\omega$. Let (v_1, v_2) be a best rotation for (u_1, u_2). Then $\overline{\omega}$-NED$(w_1, w_2) =$ NED(v_1, v_2).*

To prove Theorem 8, it is tempting to consider using the South-East graph with wrap-around as a game graph towards a reduction to 1-player MeanPayoff games, but unfortunately, it does not work. The problem is that such a reduction allows finding an optimal path that does not correspond to prefixes of the same length, while $\overline{\omega}$-NED is defined as $\limsup_{i\to\infty}(\text{NED}(w_1[..i], w_2[..i])$ thus insists on the same length of prefixes. Consider $w_1 = (aaab)^\omega$ and $w_2 = (aab)^\omega$. The LCM is 12, and $\overline{\omega}$-NED$(w_1, w_2) = 4/14$ where transformation of $(aaab)^3$ to $(aab)^4$ is via $aaabaaabaa\_a\_b \mapsto aa\_baa\_baabaab$. However, the reduction returns $1/4$, via the path deleting every first a in a "block" of w_1, essentially returning $\limsup_{i\to\infty}(\text{NED}(w_1[..3i], w_2[..4i])$ rather than $\limsup_{i\to\infty}(\text{NED}(w_1[..i], w_2[..i])$.

We, therefore, continue with a series of propositions that lead to a proof of Theorem 8. The first proposition concerns the cost of prefixes that are multiples of n (the least common multiple of the two periods).

▶ **Proposition 9** (The cost of prefixes which are multiplications of the best rotation). *Let $u_1, u_2 \in \Sigma^+$ be a best rotation pair of size n. Let ρ be an optimal edit path for (u_1^i, u_2^i) for some $i > 0$. Let ρ_* be an optimal edit path for (u_1, u_2). Then $cost(\rho) = cost(\rho_*)$.*

The proof of Prop. 9 builds on the following proposition, claiming that when looking at ρ on the words graph $\mathcal{G}(u_1^i, u_2^i)$, then some infix of ρ fits an $n \times n$ square, as formally stated in Prop. 10, and illustrated in Figure 1 (c).

▶ **Proposition 10** (Fitting an n-square). *Let $u_1, u_2 \in \Sigma^+$ be a best rotation pair of size n. Let ρ be an optimal edit path for (u_1^i, u_2^i) for some $i > 1$. There exists s and t, $0 \leq s < t < |\rho|$, such that $endpoint(\rho[s..t]) = \langle n, n\rangle$.*

The proof of Prop. 10 uses the intermediate value theorem over how much a $k \times k$ square drifts from the main diagonal. The proof can be found in the full version of the paper.

Now, using Prop. 10, we can show that the cost of an optimal edit path ρ for (u_1^i, u_2^i) is the same as the cost of ρ_*, an optimal edit path for (u_1, u_2).

Proof of Prop. 9. Clearly, since $(\rho_*)^i$ is an edit path for (u_1^i, u_2^i) and ρ is defined to be optimal for (u_1^i, u_2^i) it must hold that $cost(\rho) \leq cost((\rho_*)^i) = cost(\rho_*)$.

The proof that $cost(\rho) \geq cost(\rho_*)$ is by induction on i. For $i = 1$, the path ρ clearly cannot cost less than the path ρ_* which is optimal for this dimension.

Consider $i > 1$. By Prop. 10 there exists $0 < s < n-1$ and $t > s$ such that $endpoint(\rho[s..t]) = \langle n, n\rangle$. Let $\rho_s = \rho[..s-1]$, $\rho_t = \rho[t+1..]$. Consider the path $\rho' = \rho_s \cdot \rho_t$ obtained by removing the sub-path of ρ from s to t (as illustrated in Figure 1 (c) and (d)). It is an edit path for (u_1^{i-1}, u_2^{i-1}). Thus, by the induction hypothesis, we have that $cost(\rho') \geq cost(\rho_*)$. Since $endpoint(\rho[s..t]) = \langle n, n\rangle$ we also have $cost(\rho[s..t]) \geq cost(\rho_*)$. Note that the cost of ρ can be computed using its sub-paths $\rho[s..t]$ and ρ' which combines ρ_s and ρ_t. Let $l' = len(\rho')$ and $d' = wgt(\rho')$. Similarly, let $l_{st} = len(\rho[s..t])$ and $d_{st} = wgt(\rho[s..t])$. Last, let $l_* = len(\rho_*)$ and $d_* = wgt(\rho_*)$. We get that

$$cost(\rho) = \frac{wgt(\rho[s..t])+wgt(\rho')}{len(\rho[s..t])+len(\rho')} = \frac{d_{st}+d'}{l_{st}+l'} \geq \min\left\{\frac{d_{st}}{l_{st}}, \frac{d'}{l'}\right\} \geq \min\left\{\frac{d_*}{l_*}, \frac{d_*}{l_*}\right\} = \frac{d_*}{l_*} = cost(\rho_*)$$

where the first inequality holds by Fact 11 (proven in the full version). ◀

▶ **Fact 11.** Let $a_1, \ldots, a_n > 0$, $b_1, \ldots, b_n > 0$. Then $\frac{\sum_{1\leq i\leq n} a_i}{\sum_{1\leq i\leq n} b_i} \geq \min_{1\leq i\leq n}\left\{\frac{a_i}{b_i}\right\}$.

Next, we bound from above and below, the cost of prefixes that are not a multiplication of n.

▶ **Proposition 12** (cost of prefixes which are not multiplications of n). *Let $u_1, u_2 \in \Sigma^+$ be a best rotation pair of size n. Let $m = i \cdot n + j$ for some $i > 0$ and $0 < j < n$ and let ρ be an optimal edit path for $(u_1^\omega[..m], u_2^\omega[..m])$. Let ρ_* be an optimal path for (u_1, u_2), and assume $d_* = wgt(\rho_*)$ and $l_* = len(\rho_*)$. Then*

$$\frac{d_*}{l_*} - \frac{2(n-j)}{i\cdot n+j} \leq cost(\rho) \leq \frac{d_*+\frac{2j}{i}}{l_*+\frac{2j}{i}}.$$

Proof. For the left inequality, consider the path $\rho' = \rho \cdot (d_{\mathrm{S}})^{n-j} \cdot (d_{\mathrm{E}})^{n-j}$. That is, the path obtained from ρ by extending it with $(n-j)$ south and $(n-j)$ east steps. Note that $endpoint(\rho') = \langle (i+1)n, (i+1)n\rangle$. It follows from Prop. 9 that the cost of ρ' is at least $\frac{d_*}{l_*}$. Thus we have $cost(\rho') = \frac{wgt(\rho)+2(n-j)}{len(\rho)+2(n-j)} \geq \frac{d_*}{l_*}$. From here we get:

$$wgt(\rho)l_* \geq d_* \cdot len(\rho) + d_* \cdot 2(n-j) - l_* \cdot 2(n-j) \geq d_* \cdot len(\rho) - l_* \cdot 2(n-j)$$

dividing both sides by $l_* \cdot len(\rho)$ gives us $\frac{wgt(\rho)}{len(\rho)} \geq \frac{d_*}{l_*} - \frac{2(n-j)}{len(\rho)} \geq \frac{d_*}{l_*} - \frac{2(n-j)}{i\cdot n+j}$ where the last inequality follows from the fact that $len(\rho) \geq i \cdot n + j$.

For the right inequality, consider the path $\rho' = (\rho_*)^i \cdot (d_{\mathrm{S}})^j \cdot (d_{\mathrm{E}})^j$. That is, the path obtained from ρ_*^i by extending it with j south and j east steps. Note that $endpoint(\rho') = \langle i\cdot n+j, i\cdot n+j\rangle$. Since ρ is optimal we get: $cost(\rho) \leq cost(\rho') = \frac{wgt(\rho_*^i)+2j}{len(\rho_*^i)+2j} = \frac{i\cdot d_*+2j}{i\cdot l_*+2j} = \frac{d_*+2j/i}{l_*+2j/i}$. ◀

We are now ready to prove Theorem 8.

Proof of Theorem 8. Assume that $w_1 = z_1 u_1^\omega$ and $w_2 = z_2 u_2^\omega$. Let n be the gcd of $|u_1|$ and $|u_2|$ and let (v_1, v_2) be a best rotation of (u_1, u_2). Since $z_i u_i^\omega$, u_i^ω and v_i^ω are almost equal, for $i \in \{1, 2\}$, by Theorem 6, $\overline{\omega}\text{-NED}(w_1, w_2) = \overline{\omega}\text{-NED}(v_1^\omega, v_2^\omega)$. Let ρ^* be an optimal path for v_1, v_2. Let d_* be its weight and l_* its length. Consider $\text{NED}(v_1^\omega[..i], v_2^\omega[..i])$. If i is a multiple of n then by Prop. 9 we get that $\text{NED}(v_1^\omega[..i], v_2^\omega[..i]) = d_*/l_*$. If i is not a multiple of n then by Prop. 12 we have that $\text{NED}(v_1^\omega[..i], v_2^\omega[..i])$ approaches d_*/l_* as i grows. Thus, $\lim_{i\to\infty} \text{NED}(v_1^\omega[..i], v_2^\omega[..i]) = d_*/l_* = \text{NED}(v_1, v_2)$. ◀

5 Computing $\overline{\omega}$-NED for Languages of Infinite Words

We define the distance between two languages as the infimum of distances between two words in the respective languages (as is common in metrics when extending the distance between pair of elements in the space to pair of sets in the space). That is, $\text{NED}(L_1, L_2) = \inf_{w_1 \in L_1, w_2 \in L_2} \text{NED}(w_1, w_2)$ and $\overline{\omega}\text{-NED}(L_1, L_2) = \inf_{w_1 \in L_1, w_2 \in L_2} \overline{\omega}\text{-NED}(w_1, w_2)$. In this section we tackle the problem of computing $\overline{\omega}$-NED for regular languages of infinite words. Two questions that should be answered first, are (a) how to compute $\overline{\omega}$-NED for two ultimately periodic words (which we discuss in Subsection 5.1), and (b) how to compute NED for languages of finite words (which we discuss in Subsection 5.2). After discussing these, in Subsection 5.3, we tackle the problem of computing $\overline{\omega}$-NED for regular languages of infinite words.

5.1 Computing $\overline{\omega}$-NED for ultimately periodic words

We summarize first how computation of NED for finite words can be done.

Computing NED for words. Computation of NED for two finite words can be done in PTIME using a dynamic programming algorithm as follows [15]. Let w_1, w_2 be words of lengths m and n, respectively. Any edit path of size k satisfies $\max\{m, n\} \leq k \leq m + n$. Following the definition,

$$D(i, j, k) = \min\left\{ wgt(\rho) \;\middle|\; \rho \text{ is an edit path for } (w_1[..i-1], w_2[..j-1]) \text{ and } len(\rho) = k \right\}$$

we obtain $\text{NED}(w_1, w_2) = \min_{max(m,n) \leq k \leq m+n} \frac{1}{k} D(m, n, k)$ and it can be computed in $O(m \cdot n \cdot \min\{m, n\})$, since for k there are $m + n - \max\{m, n\}$ entries in D.

Computing $\overline{\omega}$-NED for ultimately periodic words. Let $w_1 = z_1 u_1^\omega$ and $w_2 = z_2 u_2^\omega$. Let $v_1 = u_1^\omega[..m]$ and $v_2 = u_2^\omega[..m]$ where m is the least common multiple of $|u_1|$ and $|u_2|$. It follows from Theorem 8 that $\overline{\omega}\text{-NED}(w_1, w_2)$ equals NED of a best rotation (v_1', v_2') of (v_1, v_2). We note that to look for a best rotations it suffices to check only rotations of either v_1 or v_2.

▷ Claim 13. Let $v_1, v_2 \in \Sigma^*$ such that $|v_1| = |v_2|$. There exists a best rotation (v_1', v_2') of (v_1, v_2) where $v_2' = v_2$.

Therefore $\overline{\omega}\text{-NED}(w_1, w_2) = \min_{v_1' \in rot(v_1)} \text{NED}(v_1', v_2)$; and it can be computed in $O(m^4)$. Hence:

▶ **Theorem 14.** *Let $w_1, w_2 \in \Sigma^{UP}$. Then $\overline{\omega}$-NED(w_1, w_2) can be computed in PTIME.*

5.2 Computing NED for regular languages

It is known that the infimum of the mean of a weighted graph can be computed in PTIME [6].

▶ **Lemma 15** ([6]). *Let $G = (V, E, \theta \colon E \to \mathbb{Q}_{\geq 0})$ be a weighted graph, and $V_I \subseteq V$, $V_F \subseteq F$ source and target vertices. The infimum of the mean weights of paths from V_I to V_F can be computed in PTIME.*

We can use this result to compute NED of regular languages, by building a weighted graph that corresponds to the product of two NFAs, where instead of allowing only transitions where both NFAs read the same letter, we also allow transitions where they read different letters, and transitions where one reads a letter and the other one does not (instead it reads ε). Transitions where both read a letter correspond to *replace*, where only the first reads a letter to *delete* and where only the second reads a letter to *insert*.

▶ **Definition 16** (Edit Distance Graph of two NFAs). *For $i \in \{1,2\}$ let $\mathcal{N}_i = (\Sigma, Q_i, s_i, \delta_i, F_i)$ be an NFA with no ε-moves. The* edit-distance graph of $\mathcal{N}_1$ and $\mathcal{N}_2$ *is a labeled weighted graph, denoted $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ which is defined as follows $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2) = (V, L, E, \theta)$ where $V = Q_1 \times Q_2$ is the set of vertices; the set of labels is $\Sigma_\varepsilon \times \Sigma_\varepsilon$ where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$; the set of edges $E \subseteq V \times L \times V$ is given by*

$$E = \left\{ (\langle q_1, q_2\rangle, \langle \sigma_1, \sigma_2\rangle, \langle q_1', q_2'\rangle) \;\middle|\; \begin{array}{l} \text{either } \sigma_1 \neq \varepsilon \text{ or } \sigma_2 \neq \varepsilon \\ \text{and if } \sigma_i \neq \varepsilon \text{ then } q_i' \in \delta_i(q_i, \sigma_i) \text{ otherwise } q_i' = q_i \end{array} \right\}$$

and the weight function θ associates a weight with edge $(u, l, u') \in E$ solely based on l as follows: $\theta(u, l, u') = \theta_{\text{ED}}(l)$ where if $\sigma_1 = \sigma_2$ then $\theta_{\text{ED}}(\langle \sigma_1, \sigma_2\rangle) = 0$ otherwise $\theta_{\text{ED}}(\langle \sigma_1, \sigma_2\rangle) = 1$.

Clearly, any path in $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ corresponds to an edit path of the respective words and vice versa.

▷ Claim 17. $\rho_{\text{ED}} = ((u_0, l_1, u_1),\ (u_1, l_2, u_2),\ \ldots, (u_{k-1}, l_k, u_k))$ for $l_i = (\sigma_i, \sigma_i')$ is a path in $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ if and only if $\rho = (d_1, d_2, \ldots, d_k)$ is a path in the words graph $\mathcal{G}(w_1, w_2)$ where $w_1 = \sigma_1\sigma_2\cdots\sigma_k$, $w_2 = \sigma_1'\sigma_2'\cdots\sigma_k'$ and $d_i = d_{\text{S}}$ if $\sigma_i = \varepsilon$, $d_i = d_{\text{E}}$ if $\sigma_i' = \varepsilon$, and $d_i = d_{\text{SE}}$ otherwise.

Note that in addition, all edges but those corresponding to pairs of identical letters cost 1. Thus, the sum of weights of a path ρ_{ED} in $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ corresponds to $wgt(\rho)$ and the length of ρ_{ED} to $len(\rho)$. Therefore, the infimum of the mean path in $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ corresponds exactly to the desired NED value. Hence, the NED distance between two regular languages given by NFAs $\mathcal{N}_1$ and $\mathcal{N}_2$ can be reduced to computing the infimum of the mean cycle in $\mathcal{G}_{\text{ED}}(\mathcal{N}_1, \mathcal{N}_2)$ from $V_I = \{(s_1, s_2)\}$ to $V_F = F_1 \times F_2$, and following Lemma 15 it can be computed in PTIME.

▶ **Theorem 18.** *The NED distance between two regular languages given by NFAs $\mathcal{N}_1$ and $\mathcal{N}_2$ can be computed in PTIME.*

5.3 Computing $\overline{\omega}$-NED for Regular ω-Languages

The gist of the proof of Lemma 15 is important for the development of the algorithm for infinite words. The idea is that the infimum of the mean path in a weighted graph is obtained either on a simple path, or on a path with a simple cycle, by repeating the cycle over and over. Thus, one can use Karp's dynamic programming algorithm to compute the minimal mean value amongst simple paths and cycles, that works in PTIME [12].

We would like to lift these ideas to compute $\overline{\omega}$-NED between two regular ω-languages, given by non-deterministic Büchi automata, henceforth NBA.[1] To cope with the fact that we work with Büchi automata, we need to insist that the path has a cycle, and when projected on either component, visits an accepting state somewhere along the cycle (this will guarantee that the corresponding runs of both NBAs accept). Unfortunately, this alone does not suffice.

The problem is that if we find such a path with a cycle, which indeed corresponds to traversing infixes of the corresponding words, it might not correspond to infixes of the same length, as required by the definition of $\overline{\omega}$-NED. This is since the edit-distance graph has edges corresponding to deletes and inserts, and such edges process letters only on one of the given NBA, not simultaneously on both. We should consider only ultimately periodic paths,

[1] For lack of space we do not include the standard definition of Büchi automata and refer the unfamiliar reader to [8, Chapter 1.3].

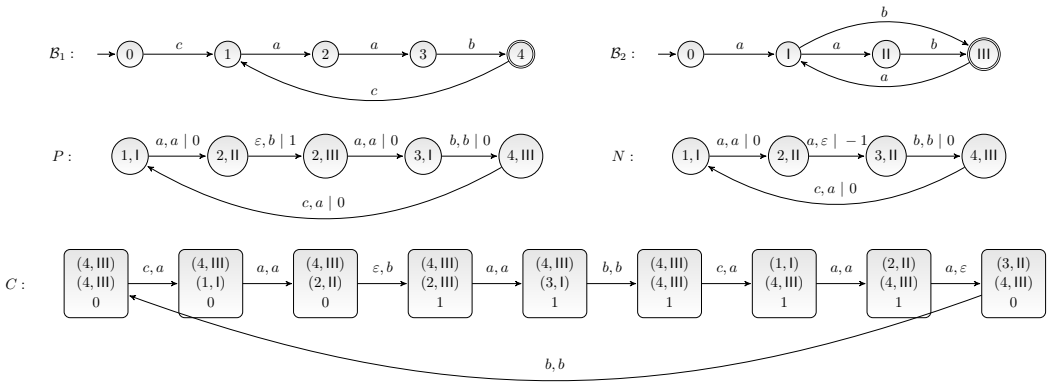


Figure 2 First row: an NBA $\mathcal{B}_1$ for $L_1 = \{caab\}^\omega$, and an NBA $\mathcal{B}_2$ for $L_2 = \{aab, ab\}^\omega$. Second row: a positive cycle P and a negative cycle N in $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$. Recall that edge labels are of the form $\sigma_1, \sigma_2 \mid bal(\sigma_1, \sigma_2)$ (the weight of the edge is not indicated; it can be inferred from the σ_1, σ_2 component of the label). Third row: an optimal cycle C in the balance t-counter graph $\mathcal{G}^t_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$ (achieving the cost $\frac{3}{9}$ via words $(caab)^\omega \in L_1$ and $(aabaabab)^\omega \in L_2$) whose balance index never goes above 1 or below 0 (i.e. here $t = 1$ suffices).

henceforth *lasso paths*, with the same number of delete and insert operations, as these are the paths that correspond to prefixes of the same length. To this aim, we add an additional annotation to edges: the *balance*.

▶ **Definition 19** (Edit Distance Graph of two NBAs). *Let $\mathcal{B}_i = (\Sigma, Q_i, s_i, \delta_i, F_i)$ be an NBA for $i \in \{1, 2\}$. The* edit-distance graph of $\mathcal{B}_1$ and $\mathcal{B}_2$ *is a labeled weighted graph, denoted $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2) = (V, L', E, \theta)$. It is defined similarly to the edit-distance graph for NFAs with one difference: the set of labels L' carries an additional annotation, the* balance. *Formally, $L' = L \times B$ where $L = \Sigma_\varepsilon \times \Sigma_\varepsilon$ is defined as for NFAs, $B = \{-1, 0, 1\}$, and label l is replaced by $(l, bal(l))$ where $bal(l) = 1$ if $l \in \{\varepsilon\} \times \Sigma$, $bal(l) = -1$ if $l \in \Sigma \times \{\varepsilon\}$, and otherwise $bal(l) = 0$. For a path $\rho = v_0 \xrightarrow{l_1, b_1} v_1 \xrightarrow{l_2, b_2} \cdots \xrightarrow{l_t, b_t} v_t$ we use $bal(\rho)$ for $\sum_{i=1}^{t} b_i$.*

Figure 2, first row shows two NBAs $\mathcal{B}_1, \mathcal{B}_2$. It does not depict the entire graph $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$ which has 5×4 states. Instead, the second row, depicts two cycles of $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$: cycle P that is positively balanced, and cycle N that is negatively balanced. We will explain the third row in the sequel.

With this definition, we are looking for the infimum mean lasso path whose balance is zero and contains a cycle that visits both components F_1 and F_2 at least once. Note that in a lasso path, it is the weight of the cycle that matters, since by repeating the cycle as much as desired, the weight of the path until the cycle begins becomes negligible. Observe that the infimum mean balanced lasso path need not be simple, i.e., it can involve two or more cycles. E.g., if we have a cycle with balance -2, another cycle with balance -3 and a cycle with balance 7 that share a vertex, the non-simple cycle repeating the -2 balanced cycle twice, and the other two cycles once is balanced. In this example since the cycles have a shared vertex, there is a cycle corresponding to the concatenation of one cycle twice with the other two cycles. But, even if they do not share a vertex, that would be fine, because, again, by repeating the cycles a large number of times the paths that connect them become negligible (though these parts as well are repeated infinitely often). The next section shows that it suffices to consider lasso paths containing at most two simple cycles – either one (that is zero balanced) or two (one that is negatively balanced and one that is positively balanced).

5.4 Enough to consider two cycles

Consider the graph $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$. Let $\mathcal{C} = \mathcal{C}_= \uplus \mathcal{C}_+ \uplus \mathcal{C}_-$ where $\mathcal{C}_= = \{\alpha \mid \alpha \text{ is a simple cycle and } bal(\alpha) = 0\}$, $\mathcal{C}_- = \{\alpha \mid \alpha \text{ is a simple cycle and } bal(\alpha) < 0\}$, $\mathcal{C}_+ = \{\alpha \mid \alpha \text{ is a simple cycle and } bal(\alpha) > 0\}$. That is, $\mathcal{C}_=$ represents cycles that are zero balanced, $\mathcal{C}_+$ represents cycles that are positively balanced (have more inserts than deletes), and $\mathcal{C}_-$ those that are negatively balanced. We define for a set of cycles $\alpha_1, \ldots, \alpha_n$, their value as follows.

▶ **Definition 20** (Value of a set of cycles). *Given a set of cycles $\{\alpha_1, \ldots, \alpha_n\}$, we use the following notations for $1 \le i \le n$: $w_i = \theta(\alpha_i)$, $l_i = |\alpha_i|$, $b_i = bal(\alpha_i)$. We define*

$$val(\alpha_1, \ldots, \alpha_n) = \min \left\{ \frac{\sum_{i=1}^{n} t_i \cdot w_i}{\sum_{i=1}^{n} t_i \cdot l_i} \;\middle|\; \sum_{i=1}^{n} t_i \cdot b_i = 0, \quad t_i \ge 0 \text{ for } 1 \le i \le n, \quad \exists j.\ t_j > 0 \right\}$$

That is, $val(\alpha_1, \ldots, \alpha_n)$ computes the minimum among the NED value of the (imaginary) path obtained by concatenating all cycles, where cycle α_i is repeated t_i times, such that the balance of the constructed path is zero.

When we have one negatively balanced and one positively balanced cycles, their *val* can be easily computed, as follows.

▶ **Observation 21.** *Let $\alpha_- \in \mathcal{C}_-, \alpha_+ \in \mathcal{C}_+$. Assume $w_\bullet = \theta(\alpha_\bullet)$, $b_\bullet = bal(\alpha_\bullet)$, $l_\bullet = len(\alpha_\bullet)$ for $\bullet \in \{-, +\}$. Then $val(\alpha_+, \alpha_-) = \frac{b_+ \cdot w_- \; - \; b_- \cdot w_+}{b_+ \cdot l_- \; - \; b_- \cdot l_+}$.*

The following proposition states that from a set involving no zero balanced cycles, it is possible to choose just one positively balanced cycle and one negatively balanced cycle and the cost of the resulting path would not be worse than the one touring many cycles.

▶ **Proposition 22** (Two cycles suffice). *Let $\alpha_1, \ldots, \alpha_m \in \mathcal{C}_-$, $\alpha_{m+1}, \ldots, \alpha_{m+k} \in \mathcal{C}_+$ for some $m, k \ge 1$. There exists $\alpha_+ \in \mathcal{C}_+$ and $\alpha_- \in \mathcal{C}_-$ with $val(\alpha_1, \ldots, \alpha_{m+k}) = val(\alpha_+, \alpha_-)$.*

The proof makes use of the following fact (proven in the full version of the paper).

▶ **Fact 23.** Given two sequences of positive numbers $\vec{a} = (a_1, \ldots, a_m)$ and $\vec{b} = (b_1, \ldots, b_k)$ satisfying $\sum_{i=1}^{m} a_i = \sum_{j=1}^{k} b_j$ there exists a matrix $P \in [0,1]^{m \times k}$ such that $\sum_{i=1}^{m} p_{ij} = 1$ for every $1 \le j \le k$ and $\vec{a} = P \cdot \vec{b}$. In other words, for all $1 \le i \le m$ we have $a_i = \sum_{j=1}^{k} p_{ij} \cdot b_j$.

Proof of Prop. 22. Assume $t_1 \ldots t_{m+k} \in \mathbb{N}$ are an optimal solution for $val(\alpha_1, \ldots, \alpha_{m+k})$. Following Def. 20 the t_is need to satisfy the following constraint:

$$0 \neq \sum_{i=1}^{m} t_i \cdot (-b_i) = \sum_{j=m+1}^{m+k} t_j \cdot b_j \tag{5.1}$$

By Fact 23 we can write Equation 5.1 as the following m linear combinations, one for each $1 \le i \le m$.

$$t_i \cdot (-b_i) = \sum_{j=m+1}^{m+k} \alpha_{ji} \cdot t_j \cdot b_j$$

such that $\alpha_{ji} \in [0,1]$ for all i, j; and $\sum_{i=1}^{m} \alpha_{ji} = 1$ for every j.

Following this observation and due to the optimal solution we get that $val(\alpha_1, \ldots, \alpha_{m+k}) =$

$$
\begin{aligned}
&= \frac{\sum_{i=1}^{m+k} t_i \cdot w_i}{\sum_{i=1}^{m+k} t_i \cdot l_i} = \frac{\sum_{j=m+1}^{m+k} \sum_{i=1}^{m} t_j \cdot \alpha_{ji} \cdot (w_j - \frac{b_j}{b_i} \cdot w_i)}{\sum_{j=m+1}^{m+k} \sum_{i=1}^{m} t_j \cdot \alpha_{ji} \cdot (l_j - \frac{b_j}{b_i} \cdot l_i)} \\
&\geq \min_{m+1 \leq j \leq m+k, t_j \neq 0} \left\{ \frac{\sum_{i=1}^{m} t_j \cdot \alpha_{ji} \cdot (w_j - \frac{b_j}{b_i} \cdot w_i)}{\sum_{i=1}^{m} t_j \cdot \alpha_{ji} \cdot (l_j - \frac{b_j}{b_i} \cdot l_i)} \right\} \\
&\geq \min_{m+1 \leq j \leq m+k, t_j \neq 0} \left\{ \min_{1 \leq i \leq m, \alpha_{ji} \neq 0} \left\{ \frac{t_j \cdot \alpha_{ji} \cdot (w_j - \frac{b_j}{b_i} \cdot w_i)}{t_j \cdot \alpha_{ji} \cdot (l_j - \frac{b_j}{b_i} \cdot l_i)} \right\} \right\} \\
&= \min_{\substack{m+1 \leq j \leq m+k \\ 1 \leq i \leq m,\ \alpha_{ji}, t_j \neq 0}} \left\{ \frac{t_j \cdot \alpha_{ji} \cdot (w_j - \frac{b_j}{b_i} \cdot w_i)}{t_j \cdot \alpha_{ji} \cdot (l_j - \frac{b_j}{b_i} \cdot l_i)} \right\} \\
&= \min_{\substack{m+1 \leq j \leq m+k \\ 1 \leq i \leq m,\ \alpha_{ji}, t_j \neq 0}} \left\{ \frac{b_i \cdot w_j - b_j \cdot w_i}{b_i \cdot l_j - b_j \cdot l_i} \right\}
\end{aligned}
$$

◀

Henceforth, we use the following notations

$$\mu_= = \min_{\alpha \in \mathcal{C}_=} \{val(\alpha)\}, \quad \mu_{+-} = \min_{\alpha_+ \in \mathcal{C}_+, \alpha_- \in \mathcal{C}_-} \{val(\alpha_+, \alpha_-)\},$$

$$\mu_* = \min_{\{\alpha_1, \ldots, \alpha_n\} \subseteq \mathcal{C}} \{val(\alpha_1, \ldots, \alpha_k)\}.$$

With these notations, we can conclude from Prop. 22 that μ_*, the best value achieved for an arbitrary set of cycles, is no better than the best value achieved for one cycle or two.

▶ **Corollary 24.** $\mu_* = \min\{\mu_=, \mu_{+-}\}$.

In Subsection 5.5 we show how μ_* can be computed. Then, in Subsection 5.6 we show that there exists words w_1, w_2 achieving μ_* and that no pair of words can achieve a value better than μ_*, i.e. that $\overline{\omega}$-NED(L_1, L_2) is indeed μ_*.

5.5 Computing μ_*

Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be complete NBAs for ω-regular languages L_1, L_2. We want to calculate $\overline{\omega}$-NED(L_1, L_2). We create $\mathcal{G}_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$, the edit distance graph of the NBAs (from Def. 19). By Claim 17 a path in $\mathcal{G}_{\text{ED}}$ represents an edit path from a word in L_1 to a word in L_2. For the path to be accepted by both NBAs it needs to visit an accepting state of $\mathcal{B}_i$ infinitely often for $i \in \{1, 2\}$. Thus we are interested in maximal strongly connected components (MSCC) that have at least one state from F_1 and at least one state from F_2. We can hence remove all vertices that do not reach such an MSCC. For simplicity we assume that our graph has one such MSCC (if this is not the case, we apply our algorithm to each such MSCC separately).

We need the following construction to make sure pairs of cycles are balanced and that we evaluate all pairs of cycles:

▶ **Definition 25** (The balance t-counter graph). *Let $\mathcal{G}_{ED}(\mathcal{B}_1, \mathcal{B}_2) = (V, L \times B, E, \theta)$ be the edit distance graph of the NBAs, and let $n = |V|$. Recall that $L = \Sigma_\varepsilon \times \Sigma_\varepsilon$ and $B = \{-1, 0, 1\}$. For threshold $t \in \mathbb{N}$ we define $\mathcal{G}^t_{ED}(\mathcal{B}_1, \mathcal{B}_2)$ as the labeled weighted graph $(V_t, L_t, E_t, \theta_t)$ where*

$V_t = V \times V \times [-t, t]$; the labeling function L_t omits the balance label from edges, i.e. $L_t = L$; the weight function θ_t associates with edge (v, l, v') for $v, v' \in V_t$ and $l \in L$ the weight $\theta_{\text{ED}}(l)$ (as in Def. 16); and the edges are $E_t = E'_t \cap (V_t \times L_t \times V_t)$ where

$$E'_t = \{ (\langle u, v, i\rangle, l, \langle u', v, j\rangle) \mid (u, \langle l, b\rangle, u') \in E,\ b = j - i \} \cup \{ (\langle u, v, i\rangle, l, \langle u, v', j\rangle) \mid (v, \langle l, b\rangle, v') \in E,\ b = j - i \}$$

Figure 2, third row shows a balanced cycle C of $\mathcal{G}^t_{\text{ED}}(\mathcal{B}_1, \mathcal{B}_2)$ for $t \geq 1$.

We continue by claiming that μ_* can be computed on the balanced graph, $\mathcal{G}^t_{\text{ED}}$ for some t. Later, we will bound the size of the required t. Let μ_t be the minimal simple cycle in $\mathcal{G}^t_{\text{ED}}$.

▶ **Lemma 26.** *There exist $t \in \mathbb{N}$ such that $\mu_t = \mu_*$.*

Proof. Recall that by Cor. 24 $\mu_* = \min\{\mu_=, \mu_{+-}\}$. For the first direction ($\geq$) let $C = ((u_1, v_1, b_1), (u_2, v_2, b_2), \ldots, (u_r, v_r, b_r), (u_1, v_1, b_1))$ be a simple cycle in $\mathcal{G}_t$ such that $\theta(C)/|C| = \mu_t$. By projecting the path C onto each coordinate we get two closed walks in $\mathcal{G}^t_{\text{ED}}$: $C_1 = (u_1, \ldots, u'_r, u_1)$ and $C_2 = (v_1, \ldots, v''_r, v_1)$. Since we have two closed walks we can decompose them into simple cycles $\alpha_1, \ldots \alpha_{m+k+l}$. We can partition them into sets according to their balances. Let $\{\alpha_1, \ldots, \alpha_m\} \subseteq \mathcal{C}_-$, $\{\alpha_{m+1}, \ldots, \alpha_{m+k}\} \subseteq \mathcal{C}_+$, and $\{\alpha_{m+k+1}, \ldots, \alpha_{m+k+l}\} \subseteq \mathcal{C}_=$, and let $t_1, \ldots, t_{m+k+l}$ denote their respective number of repetitions in the path C. The claim now follows from Cor. 24.

For the second direction ($\leq$), we show that we can find paths corresponding to $\mu_=$ and μ_{+-} of $\mathcal{G}_{\text{ED}}$ in $\mathcal{G}^t_{\text{ED}}$.

Case of $\mu_=$. Let $c_=$ be a zero balanced simple cycle in $\mathcal{G}_{\text{ED}}$ achieving $\mu_=$. Assume

$$c_= = v_1 \xrightarrow{l_1,b_1} v_2 \xrightarrow{l_2,b_2} \cdots \xrightarrow{l_{k-1},b_{k-1}} v_k \xrightarrow{l_k,b_k} v_1$$

where $\sum_{i=1}^k b_i = 0$ and $\frac{\theta(c_=)}{|c_=|} = \mu_=$. We observe that for big enough t and for every $r \in [-n, n]$ the following cycle c^r_t is in $\mathcal{G}^t_{\text{ED}}$ and it achieves the same value.

$$c^r_t = (v_1, v_1, r) \xrightarrow{l_1} (v_2, v_1, r+b_1) \xrightarrow{l_2} \cdots \xrightarrow{l_{k-1}} (v_k, v_1, r+b_k) \xrightarrow{l_k} (v_1, v_1, r)$$

Since $c_=$ is reachable from the initial state and the balance of a simple path is never larger than the length of the path there exists an $r \in [-n, n]$ such that (v_1, v_1, r) is reachable. Therefore, taking $t > n + n$ suffices.

Case of μ_{+-}. Now, let c_- and c_+ be a negatively and positively balanced simple cycles, resp., in $\mathcal{G}_{\text{ED}}$ achieving μ_{+-}. Assume

$$c_- = u_1 \xrightarrow{l_1,b_1} u_2 \xrightarrow{l_2,b_2} \cdots \xrightarrow{l_{m-1},b_{m-1}} u_m \xrightarrow{l_m,b_m} u_1$$

and

$$c_+ = v_1 \xrightarrow{l'_1,b'_1} v_2 \xrightarrow{l'_2,b'_2} \cdots \xrightarrow{l'_{k-1},b'_{k-1}} v_k \xrightarrow{l'_k,b'_k} v_1$$

where $b_- = \sum_{i=1}^m b_i < 0$, $b_+ = \sum_{j=1}^k b'_j > 0$, $\frac{\theta(c_-)}{|c_-|} = \mu_-$ and $\frac{\theta(c_+)}{|c_+|} = \mu_+$.

In $\mathcal{G}^t_{\text{ED}}$ for big enough t we can find a cycle c^t_r corresponding to repeating b_+ times the cycle c_-, and then repeating b_- times the cycle c_+. It will have the following form

$$\begin{array}{llll}
(u_1, v_1, r_{1,1}), & (u_2, v_1, r_{1,2}), & \dots, & (u_m, v_1, r_{1,m}),\\
(u_1, v_1, r_{2,1}), & (u_2, v_1, r_{2,2}), & \dots, & (u_m, v_1, r_{2,m}),\\
 & & \dots & \\
(u_1, v_1, r_{b_+,1}), & (u_2, v_1, r_{b_+,2}), & \dots, & (u_m, v_1, r_{b_+,m}),\\
(u_1, v_1, r'_{1,1}), & (u_1, v_2, r'_{1,2}), & \dots, & (u_1, v_k, r'_{1,k}),\\
(u_1, v_1, r'_{2,1}), & (u_1, v_2, r'_{2,2}), & \dots, & (u_1, v_k, r'_{2,k}),\\
 & & \dots & \\
(u_1, v_1, r'_{b_+,1}), & (u_1, v_2, r'_{b_+,2}), & \dots, & (u_1, v_k, r'_{b_-,k}),
\end{array}$$

where if $r_{1,1} = r$ then $r_{i,j} = r + i(b_-) + \sum_{k=1}^{j} b_i$ and $r'_{i,j} = r + (b_+)(b_-) + i(b_+) + \sum_{k=1}^{j} b'_i$. Hence $r'_{b_-,k} = r + (b_+)(b_-) + (b_-)(b_+) = r$ and c^t_r is a balanced cycle in $\mathcal{G}^t_{\text{ED}}$. Since c_- is reachable from the initial state and the balance of a path is never larger than the length of the path there exists $r \in [-n, n]$ such that (u_1, v_1, r) is reachable. Since both $-b_-, b_+ < n$ we have that $t > n + n^2$ suffices. ◀

The next proposition shows that we can bound t better than $n + n^2$, more precisely, that taking $t = n$ suffices. The idea of the proof is that in $\mathcal{G}^t_{\text{ED}}$ it is possible to traverse part of the negative cycle, then move to traverse part of the positive cycle and continue traversing the negative cycle from where we left off. Alternating between portions of the cycle, we can ensure the balance is never more than n or less than $-n$.

▶ **Proposition 27.** $\mu_n = \mu_*$.

Proof. We use the same idea as in the previous proof, except instead of touring the first cycle and then the second cycle, we alternate between them. Consider the cycles c_+ and c_- repeated indefinitely. Then since the accumulated balance is unbounded and *discretely-continuous* in the sense that it changes in jumps of $\{-1, 0, 1\}$, it follows from the discrete version of the intermediate value theorem [11] that we will eventually encounter all values. We can thus choose the indices $0 = i_0 < i_1 < i_2 \dots$ on the path for which the accumulated balance reaches exactly $\lceil \frac{n}{2} \rceil \cdot j$ for increasing j's: $bal(c^\omega_+[0..i_j]) = \lceil \frac{n}{2} \rceil \cdot j$ for all $j > 0$. This means that $bal(c^\omega_+[i_{j-1} \,..\, i_j - 1]) = \lceil \frac{n}{2} \rceil$. Similarly there exist indices $i'_1, i'_2 \dots$ such that $bal(c^\omega_-[i'_{j-1} \,..\, i'_j - 1]) = -\lceil \frac{n}{2} \rceil$ for all $j > 0$. We can therefore alternate between the cycles by progressing in the positive cycle until we reach $\min\{i_j, -b_-|c_+|\}$ then progress on the negative cycle until we reach $\min\{i'_j, b_+|c_-|\}$ and so on. We can keep alternating between them until we reach a balance of 0 (after $-b_-$ of the positive cycle and b_+ of the negative cycle). Since both b_+ and $-b_-$ are at most n we have that at any given point our total balance will be in the range of $[-n, n]$. Therefore $t = n$ suffices. ◀

This gives us that computing μ_* is polynomial in the size of the given automata. Specifically, for $t = n$, the graph $\mathcal{G}^t_{\text{ED}}$ is $2n + 1$ times the size of $\mathcal{G}_{\text{ED}}$ which is polynomial in the size of the automata. The overall computation of μ_* is in PTIME since (i) computing minimal cycles can be done in PTIME (ii) $\mu_* = \mu_n$ and (iii) μ_n is the minimal mean cycle in $\mathcal{G}^n_{\text{ED}}$.

5.6 Proving $\overline{\omega}$-NED$(L_1, L_2) = \mu_*$

Next, we would like to show that $\overline{\omega}\text{-NED}(L_1, L_2) = \mu_*$. Prop. 28 shows that for every ultimately periodic words $w_1 \in L_1$ and $w_2 \in L_2$ we have $\overline{\omega}\text{-NED}(w_1, w_2) \geq \mu_*$. Theorem 30 shows that this is the case also for arbitrary words $w_1 \in L_1$ and $w_2 \in L_2$.

On the other hand, we show that μ_* can be achieved by respective words $w_1 \in L_1$ and $w_2 \in L_2$. Prop. 29 shows that we can get arbitrarily close to μ_* in the sense that for every $\varepsilon > 0$ we can find ultimately periodic words w_1, w_2 such that $|\,\overline{\omega}\text{-NED}(w_1, w_2) - \mu_*| < \varepsilon$.

▶ **Proposition 28** (Lower-bound). *For $w_1 \in L_1 \cap \Sigma^{UP}$, $w_2 \in L_2 \cap \Sigma^{UP}$, $\overline{\omega}$-NED$(w_1, w_2) \geq \mu_*$.*

Proof. Let (v_1, v_2) be a best rotation pair for the periods of w_1, w_2 and let $\rho \in \mathbb{D}^*$ be an optimal edit path for (v_1, v_2). We show that there exists a cycle c in $\mathcal{G}^t_{\text{ED}}$ that is accepting in both NBAs and satisfies $\theta(c)/|c| = \text{NED}(v_1, v_2)$.

For $i \in \{1, 2\}$ let $\beta_i \cdot \gamma_i^\omega$ be an accepting lasso run of $\mathcal{B}_i$ on w_i where γ_i reads the same multiple m of v_i. Then ρ^m is a best edit path for v_1^m and v_2^m. To simplify the notations, we use v_i and ρ instead of v_i^m and ρ^m, i.e., assume w.l.o.g. $|\gamma_i| = |v_i| = k$. Assume $\beta_1 = q_1 q_2 \dots q_{\ell_1}$, $\beta_2 = q'_1 q'_2 \dots q'_{\ell_2}$, $\gamma_1 = p_1 p_2 \dots p_k$, $\gamma_2 = p'_1 p'_2 \dots p'_k$. Consider the paths $\beta = (q_1, q'_1), (q_2, q'_1), \dots, (q_{\ell_1}, q'_1), (q_{\ell_1}, q'_2), (q_{\ell_1}, q'_3) \dots (q_{\ell_1}, q'_{\ell_2})$ and $\gamma = (p_{i_1}, p'_{j_1}), \dots, (p_{i_k}, p'_{j_k})$ where $i_1 = 1$, $j_1 = 1$ and

$$i_r = \begin{cases} i_{r-1} & \text{if } \rho[r] = d_{\text{E}} \text{ or } i_r = |\gamma_1| \\ i_{r-1}{+}1 & \text{otherwise} \end{cases} \qquad j_r = \begin{cases} j_{r-1} & \text{if } \rho[r] = d_{\text{S}} \text{ or } j_r = |\gamma_2| \\ j_{r-1}{+}1 & \text{otherwise} \end{cases}$$

Since $|\gamma_1| = |\gamma_2|$ we have that $|\rho| = |\gamma|$ and γ is a reachable cycle with value: $\theta(\gamma)/|\gamma| = \mathit{wgt}(\rho)/\mathit{len}(\rho) = \mathit{cost}(\rho) \geq \mu_*$. ◀

▶ **Proposition 29** (Upper-bound). *For all $\varepsilon > 0$ there exists $w_1 \in L_1 \cap \Sigma^{UP}, w_2 \in L_2 \cap \Sigma^{UP}$ such that $|\,\overline{\omega}$-NED$(w_1, w_2) - \mu_*| < \varepsilon$.*

Proof. We show there are valid edit paths corresponding to both $\mu_=$ and μ_{+-}. Let $\varepsilon > 0$.

Case of $\mu_=$: Let $c_=$ be a simple cycle in $\mathcal{G}_{\text{ED}}$ with $\mathit{val}(c_=) = \mu_=$ and $\mathit{bal}(c_=) = 0$. By our assumption on the MSCC, there exists a cycle c traversing an accepting state from both F_1 and F_2. W.l.o.g. it intersects $c_=$. We claim that we can assume $\mathit{bal}(c) = 0$. Assume w.l.o.g. $\mathit{bal}(c) = b > 0$. We can find a negatively balanced cycle c_- that starts at the intersection of c and $c_=$ by tracing only edges corresponding to $\mathcal{B}_1$ and staying put on a fixed state of $\mathcal{B}_2$ (for details see Lemma 36 in the full version). Then $c' = (c)^{(b_-)}(c_-)^{(b)}$ is a zero balanced cycle in $\mathcal{G}_{\text{ED}}$ visiting both F_1 and F_2.

Let ρ_u be a path from the initial state to $c_=$. Let ρ_v be the cycle $((c_=)^{n_\varepsilon} \cdot c)$. Consider $\rho = \rho_u \cdot (\rho_v)^\omega$. Recall that a path in $\mathcal{G}_{\text{ED}}$ is an element of $(V \times L' \times V)^\infty$ where $V = Q_1 \times Q_2$ and $L' = \Sigma_\varepsilon \times \Sigma_\varepsilon \times \{-1, 0, 1\}$. Given a path ρ in $\mathcal{G}_{\text{ED}}$, we use $\mathit{labels}(\rho)$ for $\pi_2(\rho) \in (L')^\infty$. For $i \in \{1, 2\}$, let $u_i = \pi_i(\mathit{labels}(\rho_u))$, $v_i = \pi_i(\mathit{labels}(\rho_v))$ and $w_i = u_i(v_i)^\omega$. Then $w_i \in L_i$ since the corresponding runs of $\mathcal{B}_i$ visit an accepting state in F_i infinitely often.

We turn to show that $\overline{\omega}\text{-NED}(w_1, w_2) - \mu_= < \varepsilon$. Note that since ρ_v is balanced $|v_1| = |v_2|$. Following Theorem 8, $\overline{\omega}\text{-NED}(w_1, w_2) \leq \text{NED}(v_1, v_2)$. Thus, for a large enough n_ε:

$$\overline{\omega}\text{-NED}(w_1, w_2) - \mu_= \;\leq\; \text{NED}(v_1, v_2) - \mu_= \;=\; \frac{\theta(\rho_v)}{|\rho_v|} - \mu_= \;=$$
$$\frac{n_\varepsilon \theta(c_=) + \theta(c)}{n_\varepsilon |c_=| + |c|} - \frac{\theta(c_=)}{|c_=|} \;\leq\; \frac{\theta(c)}{n_\varepsilon |c_=|} < \varepsilon$$

Case of μ_{+-}: In a similar fashion we have the simple cycles c_+, c_- $\mathcal{G}_{\text{ED}}$ with $\mathit{val}(c_+) = \mu_+$, $\mathit{val}(c_-) = \mu_-$ and $\mathit{bal}(c_+) = b_+ > 0, \mathit{bal}(c_-) = b_- < 0$. Since c_+, c_- are in the same SCC, we have a cycle $c = p_1 p_2$ where p_1 is from the first state of c_+ to the first state of c_- and p_2 comes back to the first state in c_+ while containing accepting states from both F_1 and F_2. We can assume that $\mathit{bal}(c) = b$ is 0 since if this is not the case we can extract from c a positive and a negative cycle (see Lemma 36 in the full version), and achieve a balance of zero by repeating each number of times that corresponds to the balance of the other.

Since c_+ is reachable from the initial state we have a path ρ_u from the initial state to c_+. Let ρ_v be the cycle $(c_+)^{-b_-\cdot n_\varepsilon} \cdot p_1 \cdot (c_-)^{b_+\cdot n_\varepsilon} \cdot p_2$. For $i \in \{1,2\}$, let $u_i = \pi_i(\mathit{labels}(\rho_u))$, $v_i = \pi_i(\mathit{labels}(\rho_v))$ and $w_i = u_i(v_i)^\omega$. Then $w_i \in L_i$ since the corresponding runs of $\mathcal{B}_i$ visit an accepting state in F_i infinitely often. Thus, for large enough n_ε,

$$\begin{aligned}
\overline{\omega}\text{-NED}(w_1,w_2) - \mu_{+-} &= \overline{\omega}\text{-NED}(v_1^\omega, v_2^\omega) - \mu_{+-} \\
&\le \frac{\theta(\rho_v)}{|\rho_v|} - \mu_{+-} \\
&= \frac{n_\varepsilon\cdot(-b_-\cdot\theta(c_+)+b_+\cdot\theta(c_-))+\theta(c)}{n_\varepsilon\cdot(-b_-\cdot|c_+|+b_+\cdot|c_-|)+|c|} - \frac{-b_-\cdot|c_+|+b_+\cdot|c_-|}{-b_-\cdot|c_+|)+b_+\cdot|c_-|} \\
&\le \frac{\theta(c)}{n_\varepsilon\cdot(-b_-\cdot|c_+|+b_+\cdot|c_-|)} < \varepsilon.
\end{aligned}$$

◀

▶ **Theorem 30** (Ultimately periodic words suffice). $\overline{\omega}\text{-}\mathit{NED}(L_1,L_2) = \overline{\omega}\text{-}\mathit{NED}(L_1\cap\Sigma^{UP}, L_2\cap\Sigma^{UP})$

Proof. Clearly $\overline{\omega}\text{-NED}(L_1,L_2) \le \overline{\omega}\text{-NED}(L_1\cap\Sigma^{\text{UP}}, L_2\cap\Sigma^{\text{UP}})$. For the other direction, it suffices to show that for all $\varepsilon > 0$ and any $w_1 \in L_1, w_2 \in L_2$ there exists $w_1' \in L_1\cap\Sigma^{\text{UP}}, w_2' \in L_2\cap\Sigma^{\text{UP}}$ such that $|\overline{\omega}\text{-NED}(w_1',w_2') - d| < \varepsilon$ where $d = \overline{\omega}\text{-NED}(w_1,w_2)$. Let r_1, r_2 be accepting runs of w_1, w_2 in their respective NBAs, $\mathcal{B}_1$ and $\mathcal{B}_2$. Let q_1 and q_2 be such that $r_1[i] = q_1$ and $r_2[i] = q_2$ for infinitely many is. Such a pair exists by the pigeonhole principle. Let n_* be the first index such that $r_1[n_*] = q_1$ and $r_2[n_*] = q_2$.

Since the number of states in both automata is finite, there is a $C \in O(|\mathcal{G}_{\text{ED}}|^2)$ such that if there is a balanced path between some pair of states in $\mathcal{G}_{\text{ED}}$ then there is one of length at most C. Let $n_0 > \max\{n_*,\ 2C/\varepsilon\}$ be such that $|\text{NED}(w_1[..n_0], w_2[..n_0]) - d| < \varepsilon/2$. Let ρ be the corresponding edit path. Let $n_{**} \ge n_0$ be the first index after n_0 such that $r_1[n_{**}] = q_1$ and $r_2[n_{**}] = q_2$ and accepting states of both NBAs have been visited along the way. Let $u_i = w_i[..n_*]$, $v_i = w_i[n_*{+}1..n_0]$. Let ρ_z be a path from $(r_1[n_0], r_2[n_0])$ to $(r_1[n_{**}], r_2[n_{**}])$ and z_i the words obtained by projecting ρ_z on the NBAs $\mathcal{B}_i$ for $i \in \{1,2\}$. Note that the length of ρ_z is bounded by C and thus also its weight. Let ρ_v be an optimal edit path for v_1, v_2. Let i be the smallest integer such that both coordinates of $\mathit{endpoint}(\rho[..i])$ are greater or equal to n_* (see Figure 1 (e)). Assume w.l.o.g. that $\mathit{endpoint}(\rho[..i]) = \langle n_*, n_*{+}\delta\rangle$ where $\delta \ge 0$. Because $\langle n_*, n_*\rangle$ is on the diagonal of $\mathcal{G}(v_1z_1, v_2z_2)$, the number of south edges in ρ is at least δ, thus $\mathit{wgt}(\rho[..i]) \ge \delta$. Also $\mathit{wgt}(\rho_v) \le \delta + \mathit{wgt}(\rho[i{+}1..n_0])$ and a similar inequality holds for the respective lengths. Further $n_* + \delta \ge \mathit{len}(\rho[..i]) \ge \mathit{wgt}(\rho[i..])$. Thus

$$\begin{aligned}
\overline{\omega}\text{-NED}(u_1(v_1z_1)^\omega, u_2(v_2z_2)^\omega) - d &\le \text{NED}(v_1z_1, v_2z_2) - d \\
&\le \frac{\mathit{wgt}(\rho_v) + \mathit{wgt}(\rho_z)}{\mathit{len}(\rho_v) + \mathit{len}(\rho_z)} - d \\
&\le \frac{\delta + \mathit{wgt}(\rho[i{+}1..n_0]) + \mathit{wgt}(\rho_z)}{\delta + \mathit{len}(\rho[i{+}1..n_0]) + \mathit{len}(\rho_z)} - d \\
&\le \frac{n_* + \delta + \mathit{wgt}(\rho[i{+}1..n_0]) + \mathit{wgt}(\rho_z)}{n_* + \delta + \mathit{len}(\rho[i{+}1..n_0]) + \mathit{len}(\rho_z)} - d \\
&\le \frac{\mathit{wgt}(\rho[..i]) + \mathit{wgt}(\rho[i{+}1..n_0]) + \mathit{wgt}(\rho_z)}{\mathit{len}(\rho[..i]) + \mathit{len}(\rho[i{+}1..n_0]) + \mathit{len}(\rho_z)} - d \\
&\le \frac{\mathit{wgt}(\rho) + \mathit{wgt}(\rho_z)}{\mathit{len}(\rho) + \mathit{len}(\rho_z)} - d \\
&\le \frac{\mathit{wgt}(\rho) + C}{\mathit{len}(\rho)} - d\frac{\mathit{wgt}(\rho)}{\mathit{len}(\rho)} - d + \frac{C}{n_0} \le \varepsilon
\end{aligned}$$

The first inequality follows from Theorem 8. We use inequality because the concerned periods may not be the best rotation. The second follows since $\rho \cdot \rho_z$ is an edit path for (v_1z_1, v_2z_2). For the third inequality, recall that (n_*, n_*) is on the diagonal of $\mathcal{G}(v_1z_1, v_2z_2,)$

and Because the number of south edges in ρ is at least δ, $wgt(\rho[..i]) \geq \delta$. The third inequality follows since the optimal path from (n_*, n_*) to (n_{**}, n_{**}) is better than going δ steps to the south and then following ρ to (n_0, n_0) and then following ρ_z. The rest follows by applying arithmetic on our assumptions and noticing that if $\hat{\rho}$ is an edit path for w_1, w_2 then $cost(\hat{\rho}) \geq cost(\rho)$ by Fact 11. ◀

▶ **Corollary 31.** $\overline{\omega}\text{-}NED(L_1, L_2) = \mu_*$

▶ Remark 32 (Distance on Muller and Parity automata). In the full version we show that this construction can be extended to work with Muller and Parity automata.

6 Conclusion and Future Work

We have shown that a natural extension of the normalized edit distance (NED) from words to infinite words is a metric and explained its advantages over other measures of distance for infinite words. We have shown that it can be computed in PTIME, when the words are ultimately periodic. We have further shown that the distance between two regular ω-languages given by non-deterministic Büchi automata can be computed in PTIME.

While our choice of ignoring finite prefixes has been justified, in some cases one would like to distinguish for instance $z_1 = a^\omega$ and $z_2 = bbba^\omega$ and $z_3 = b^{100}a^\omega$ and require that the distance between z_1 and z_2 be smaller than the distance between z_1 and z_3. In particular, if the words have a common suffix (as is the case for z_1, z_2, z_3), we would like a measure that reflects the normalized number of edit operations required on the prefix. This can potentially be done by relaxing the requirement for a metric, and working with generalized metric spaces. But it also requires a formal definition of when the prefix ends. Challenges in achieving this and several suggestions are discussed in the 2nd author's master thesis [9].

References

1. Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, Georg Hofferek, Barbara Jobstmann, Bettina Könighofer, and Robert Könighofer. Synthesizing robust systems. *Acta Informatica*, 51(3-4):193–220, 2014. doi:10.1007/s00236-013-0191-5.
2. Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, and Barbara Jobstmann. Specification-centered robustness. In *Industrial Embedded Systems (SIES), 2011 6th IEEE International Symposium on, SIES 2011. Vasteras, Sweden, June 15-17, 2011*, pages 176–185, 2011. doi:10.1109/SIES.2011.5953660.
3. Pavol Cerný, Thomas A. Henzinger, and Arjun Radhakrishna. Simulation distances. *Theor. Comput. Sci.*, 413(1):21–35, 2012.
4. Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010. doi:10.1145/1805950.1805953.
5. Colin de la Higuera and Luisa Micó. A contextual normalised edit distance. In *Proceedings of the 24th International Conference on Data Engineering Workshops, ICDE 2008, April 7-12, 2008, Cancún, Mexico*, pages 354–361. IEEE Computer Society, 2008.
6. Emmanuel Filiot, Nicolas Mazzocchi, Jean-François Raskin, Sriram Sankaranarayanan, and Ashutosh Trivedi. Weighted transducers for robustness verification. In *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, pages 17:1–17:21, 2020.
7. Dana Fisman, Joshua Grogin, Oded Margalit, and Gera Weiss. The normalized edit distance with uniform operation costs is a metric. In *33rd Annual Symposium on Combinatorial Pattern Matching (CPM)*, 2022. To appear (meantime available on arxiv). arXiv:2201.06115.

8 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

9 Joshua Grogin. A normalized edit distance on finite and infinite words, Master Thesis, Ben-Gurion University of the Negev, March 2022. URL: https://jgrogin.github.io/A_Normalized_Edit_Distance_on_Finite_and_Infinite_Words_thesis.pdf.

10 H. J. Hoogeboom and G. Rozenberg. *Infinitary languages: Basic theory and applications to concurrent systems*, pages 266–342. Springer Berlin Heidelberg, Berlin, Heidelberg, 1986. doi:10.1007/BFb0027043.

11 Richard Johnsonbaugh. A Discrete Intermediate Value Theorem. https://www.maa.org/sites/default/files/0746834259610.di020780.02p0372v.pdf, 1998. The College Mathematical Journal.

12 Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discret. Math.*, 23(3):309–311, 1978. doi:10.1016/0012-365X(78)90011-0.

13 Vladimir Iosifovich Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, February 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

14 Yujian Li and Bi Liu. A normalized levenshtein distance metric. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1091–1095, 2007.

15 Andrés Marzal and Enrique Vidal. Computation of normalized edit distance and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):926–932, 1993.

16 Daniel Neider, Alexander Weinert, and Martin Zimmermann. Robust, expressive, and quantitative linear temporal logics: Pick any two for free. In *Proceedings Tenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2019, Bordeaux, France, 2-3rd September 2019*, pages 1–16, 2019. doi:10.4204/EPTCS.305.1.

17 Paulo Tabuada and Daniel Neider. Robust linear temporal logic. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 – September 1, 2016, Marseille, France*, pages 10:1–10:21, 2016. doi:10.4230/LIPIcs.CSL.2016.10.

Constructive and Synthetic Reducibility Degrees

Post's Problem for Many-One and Truth-Table Reducibility in Coq

Yannick Forster
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Inria, Gallinette Project-Team, Nantes, France

Felix Jahn
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We present a constructive analysis and machine-checked theory of one-one, many-one, and truth-table reductions based on synthetic computability theory in the Calculus of Inductive Constructions, the type theory underlying the proof assistant Coq. We give elegant, synthetic, and machine-checked proofs of Post's landmark results that a simple predicate exists, is enumerable, undecidable, but many-one incomplete (Post's problem for many-one reducibility), and a hypersimple predicate exists, is enumerable, undecidable, but truth-table incomplete (Post's problem for truth-table reducibility).

In synthetic computability, one assumes axioms allowing to carry out computability theory with all definitions and proofs purely in terms of functions of the type theory with no mention of a model of computation. Proofs can focus on the essence of the argument, without having to sacrifice formality. Synthetic computability also clears the lense for constructivisation.

Our constructively careful definition of simple and hypersimple predicates allows us to not assume classical axioms, not even Markov's principle, still yielding the expected strong results.

2012 ACM Subject Classification Theory of computation → Constructive mathematics; Theory of computation → Type theory

Keywords and phrases type theory, computability theory, constructive mathematics, Coq

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.21

Supplementary Material *Software (Proofs)*:
`https://github.com/uds-psl/coq-synthetic-computability/tree/reddegrees`
archived at `swh:1:dir:94351081a9b85bf57af18968712e4e22afea157d`

Funding *Yannick Forster*: received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101024493.

Acknowledgements We thank Dominik Kirst, Gert Smolka, Lennard Gäher, and Andrej Dudenhefner for discussions and feedback on the drafts of this paper, as well as the reviewers for their comments.

1 Introduction

The founding moment of "computability theory" deserving of the suffix "theory" was maybe Emil L. Post's 1944 paper [24]. Post introduced the concepts of one-one, many-one, and truth-table reducibility and identified and answered important questions on the structure of the reducibility degrees induced by these relations. Centrally, Post was interested in the question whether there are enumerable but undecidable degrees such that an undecidability proof cannot be done by reduction from the halting problem. For many-one and truth-table reducibility, Post was able to construct such degrees by introducing simple and hypersimple sets, which are still taught in modern textbook presentations of the field. The question whether an enumerable, undecidable problem which is not Turing-reducible from the halting problem exists became known as *Post's problem*, and we reuse the terminology for *Post's problem for many-one reducibility* ($\preceq_{\mathsf{m}}$) and *Post's problem for truth-table reducibility* ($\preceq_{\mathsf{tt}}$).

Early in his paper, Post remarks *'That mathematicians generally are oblivious to the importance of this work of Gödel, Church, Turing, Kleene, Rosser and others as it affects the subject of their own interest is in part due to the forbidding, diverse and alien formalisms in which this work is embodied.'* The evolution of "computability" to "computability theory" started by Post was enabled by a presentation of the lead questions in a more appealing, intuitive way, abstracting away from the "forbidding, alien formalisms" constituted by general recursive functions, Turing machines, the λ-calculus, or Post's own tag systems. Like in modern textbook presentations and research papers on computability (and complexity) theory, concrete formalisations are avoided by universal application of Church's thesis already to prove basic results such as the *s-m-n* theorem, fixpoint theorems, etc.

This circumstance is remarkable because it turns computability theory into a field not directly applicable to machine-checked proofs. While the use of proof assistants for cutting-edge research in e.g. programming language theory is already omnipresent and is even entering mainstream mathematics with the Lean proof assistant's `mathlib` [19], computability is unlikely to just catch up: Manually filling in machine-checked proofs for every use of (informal) Church's thesis is just infeasible.

We thus work in a different setting which renders computability theory subject to machine-checked proofs: synthetic computability using as logical foundation the Calculus of Inductive Constructions (CIC, the type theory underlying the Coq proof assistant) as set up by the first author of the present paper [8]. The setting has several historic predecessors: In the most basic form of synthetic computability, morally present in the Russian school of constructivism due to Markov [18], one assumes an axiom stating that for every function $\mathbb{N} \to \mathbb{N}$ there exists a μ-recursive function computing it, known as CT (Church's thesis, [16]). In synthetic computability due to Richman [26], a function $\phi\colon \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N})$ is *abstractly* assumed to be universal for all partial functions $\mathbb{N} \rightharpoonup \mathbb{N}$ (the axiom EPF, stating that $\forall f : \mathbb{N} \rightharpoonup \mathbb{N}.\exists c.\ \phi_c \equiv f$). Richman assumes the axiom of countable choice to be able to prove the *s-m-n* theorem, and in his book with Bridges [3] they remark that assuming just an *s-m-n* operator abstractly would also suffice, without elaborating further. Abstracting even more, Bauer [1] just works with enumerable sets and assumes that the set of enumerable sets is enumerable, together with Markov's principle and the axiom of countable choice.

As a consequence of assuming axioms of synthetic computability together with the axioms of countable choice, the law of excluded middle becomes disprovable since it entails the existence of a non-computable function. This renders synthetic computability in these settings a constructivist, anti-classical endeavour. Textbook presentations of computability theory are however explicit in that they use classical logic freely. Thus, to translate textbook presentations into an anti-classical synthetic setting, one has to constructivise proofs on the go. However, this is not always completely possible: It is well-known that Post's theorem that a set is decidable if and only if both it and its complement are enumerable is equivalent to MP (*Markov's Principle*) [31]. For other results, e.g. the ones covered in this paper, to the best of our knowledge it was unknown which logical assumptions such as LEM (*Excluded Middle*) or MP are necessary.

The first author [8, 7] has observed that in CIC, assuming a universal function ϕ abstractly together with an abstract *s-m-n* operator suffices to carry out synthetic computability theory. Equivalently, several axioms can be used: The axiom EPF, concerned with partial functions and more akin to the one used by Richman, and the axiom EA, concerned with enumerable predicates and more akin to Bauer's axiom. Since in CIC – contrary to other constructive foundations – the axiom of countable choice is not provable, it seems that one can consistently assume classical logical axioms even as strong as the law of excluded middle if needed.

The setting is fully in the spirit of Post's abstraction away from the "forbidding, alien formalisms" of machine models and forms a suitable setting for the analysis of the constructive status of theorems in computability theory. The present paper

1. presents a fully synthetic development of most results from Post's 1944 paper [24], with no reference to any model of computation,
2. uses intuitionistic logic enriched with the assumption of a (parametrically) universal enumerator but *no* additional axioms, and
3. is fully mechanised in the Coq proof assistant as a further step in the overarching goal of mechanising the pillar-stones of mathematics and computer science. The results of the paper are hyperlinked with Coq code as indicated by the  -symbols.

We believe that machine-checked proofs are also on its own a contribution to "classical" computability theory, i.e. without considering synthetic and constructive aspects: It enables the creation of a machine-checked library of the central theorems of computability theory, ensuring that no odd corner cases are left out in proofs or are hidden under uses of the Church Turing thesis. For the present paper, converting the well-known proofs from a set-theoretic foundation present in textbooks to type theory was not problematic, and modelling choices did not play a crucial role.

For Post's problem for $\preceq_{\mathsf{m}}$, i.e. that an enumerable, undecidable, many-one incomplete, simple predicate exists, the definition of simple in the synthetic setting is most interesting. The complement of a simple predicate has to be infinite, but is not allowed to have an infinite, enumerable subpredicate. In classical mathematics, p is infinite if and only if it is Cantor-infinite, i.e. if there is an injective function $\mathbb{N} \hookrightarrow p$. However, Cantor-infinite predicates (defined via functions) have a (synthetically) enumerable, infinite subpredicate – enumerated by said function. In constructive mathematics, a predicate p is called infinite if for any sequence $[x_1, \dots, x_n]$ there exists a y different from all x_i but s.t. py. However, any fully constructive proof of infinity in this sense can be turned into a proof of Cantor infinity, meaning there can be no such proof for the complement of a simple predicate. It is thus crucial that infinity is defined as non-finiteness free and thereby void of any computational content. The complement of a simple predicate is then non-finite, but not (synthetically) Cantor-infinite. Only with this definition of infinity Post's problem for $\preceq_{\mathsf{m}}$ can be settled constructively.

For Post's problem for $\preceq_{\mathsf{tt}}$, i.e. that an enumerable, undecidable, truth-table incomplete, hypersimple predicate exists, several interesting aspects appear: First, the definition of hypersimple predicates has to be chosen carefully to ensure that hypersimple predicates are tt-incomplete. The construction of a hypersimple predicate H is then easier. But since conventional proofs of its undecidability factor via simpleness (and since the textbook proofs showing that hypersimple predicates are simple seem to be inherently classical and we only manage to weaken the assumption to MP), we give a direct, fully constructive undecidability proof for H, which however does not generalise to arbitrary hypersimple predicates.

We try to give intuitive conceptual outlines in the paper, but focus on the interesting synthetic and constructive aspects more than the proof ideas. We largely follow the excellent book by Rogers [27], supplemented by the books by Cutland [4], Soare [28], and Odifreddi [21]. The Bachelor's thesis of the second author [14], containing preliminary results covered in this paper, discusses some aspects in more detail. The PhD thesis of the first author also discusses the material presented in this paper [7]. In general, it might be helpful for non-experts in computability theory to consult one of the books in cases where the presented intuition is not sufficient.

2 The Calculus of Inductive Constructions

We work in the calculus of inductive constructions (CIC) as implemented by the Coq proof assistant [29]. The calculus is a constructive type theory with a (cumulative hierarchy of) type universe(s) $\mathbb{T}$ and an impredicative universe of propositions $\mathbb{P} \subseteq \mathbb{T}$. The universe of propositions $\mathbb{P}$ is impredicative, so e.g. $(\forall X : \mathbb{P}.\ X) : \mathbb{P}$, and a sub-universe of $\mathbb{T}$, i.e. whenever $P : \mathbb{P}$ we also have $P : \mathbb{T}$. Both aspects however only play a minor role in this paper, important is that the universe $\mathbb{P}$ is separated from the universe $\mathbb{T}$. That means that in general, computations cannot inspect proofs to return a computational value.

Most instructively, the difference can be explained by looking at dependent pairs $\Sigma x.\ Ax$ – which we verbalise as "one can construct x s.t. Ax" – and existential quantification $\exists x.\ Ax$ – "there exists x s.t. Ax". Both are implemented as an inductive type of (dependent) pairs (x, y), with the only difference that $(\exists x.\ Ax) : \mathbb{P}$ but $(\Sigma x.\ Ax) : \mathbb{T}$. Dependent pairs can be eliminated in arbitrary contexts, i.e. there is an elimination function of the following type[1]

$$\forall p : (\Sigma x.\ Ax) \to \mathbb{T}.\ (\forall xy.\ p(x, y)) \to \forall (s : \Sigma x.\ Ax).\ ps.$$

We call principles eliminating a proposition into types *large elimination principles*, following the terminology "large elimination" for Coq's case analysis construct `match` [22]. Crucially, CIC proves a large elimination principle for the falsity proposition $\bot$, i.e. explosion applies to arbitrary types: $\forall A : \mathbb{T}.\ \bot \to A$. For existential quantification, a large elimination principle is not provable, the only principle one can obtain is the following, where the return type of p is $\mathbb{P}$: $\forall p : (\exists x.\ Ax) \to \mathbb{P}.\ (\forall xy.\ p(x, y)) \to \forall (s : \exists x.\ Ax).\ ps$.

There are two exceptions to this observation: First, whenever $\exists x.\ Ax$ (also in the relational form $\forall x_1.\exists x_2.\ Rx_1x_2$), can be proved *without assumptions*, one could instead prove the stronger result that $\Sigma x.\ Ax$ (for the relational form $\forall x_1.\Sigma x_2.\ Rx_1x_2$ or equivalently $\Sigma f.\forall x_1.Rx_1(fx_1)$). Secondly, for the case where p has a boolean decision function, an elimination principle is provable, reminiscent of the μ operator of general recursive functions.

Fact 1. *There is a guarded minimisation function*

$$\mu_{\mathbb{N}} : \forall f : \mathbb{N} \to \mathbb{B}.\ (\exists n.\ fn = \mathsf{true}) \to \Sigma n.\ fn = \mathsf{true} \wedge \forall m.\ fm = \mathsf{true} \to m \geq n.$$

The inductive types of interest in this paper are all standard, but listed for reference:

$n : \mathbb{N} ::= 0 \mid \mathsf{S}\, n$	(natural numbers)	$A + B := \mathsf{inl}\, a \mid \mathsf{inr}\, b$ *with* $a\colon A,\ b\colon B$	(sums)
$b : \mathbb{B} ::= \mathsf{false} \mid \mathsf{true}$	(booleans)	$A \times B := (a, b)$ *with* $a\colon A,\ b\colon B$	(pairs)
$l\colon \mathbb{L}A ::= [] \mid a :: l$ *with* $a\colon A$	(lists)	$\Sigma x\colon X.Ax := (x_0, a)$ *with* $A\colon X \to \mathbb{T},\ x_0\colon X, a\colon Ax_0$	(dependent pairs)
$o\colon \mathbb{O}A ::= \mathsf{None} \mid \mathsf{Some}\, a$ *w/* $a\colon A$	(options)		

We mention the following notations here: We use pairing function on natural numbers written as $\langle \_, \_ \rangle : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ and for all $f : \mathbb{N} \to \mathbb{N} \to X$ an inverse construction $\lambda\langle n, m\rangle.\ fnm$ of type $\mathbb{N} \to X$ s.t. $(\lambda\langle n, m\rangle.fnm)\langle n, m\rangle = fnm$. We use the list operations $\mathsf{map}\colon (X \to Y) \to \mathbb{L}X \to \mathbb{L}Y$, $\_ +\!\!+ \_ : \mathbb{L}X \to \mathbb{L}X \to \mathbb{L}X$, $\mathsf{filter}\colon (X \to \mathbb{B}) \to \mathbb{L}X \to \mathbb{L}X$, and $\_[\_] : \mathbb{L}X \to \mathbb{N} \to \mathbb{O}X$. We write $|l|$ for the length of $l : \mathbb{L}X$. For $p\colon X \to \mathbb{P}$, we denote the complement as $\overline{p}$ and the Cartesian product with $q\colon Y \to \mathbb{P}$ as $p \times q$. p is called *inhabited* if $\exists x.px$. When defining a predicate we identify a type X with $\lambda(x : X).\top$. We write $\mathsf{Forall}_2\, p\, l_1\, l_2$ for $l_1 : \mathbb{L}X$, $l_2 : \mathbb{L}Y$, and $p\colon X \to Y \to \mathbb{P}$ if p holds pointwise on the lists l_1 and l_2, i.e. for the inductive predicate with constructors of type $\mathsf{Forall}_2\ []\ []$ and $\forall x\ y\ l_1\ l_2.\ pxy \to \mathsf{Forall}_2\ l_1\ l_2 \to \mathsf{Forall}_2\ (x :: l_1)\ (y :: l_2)$.

[1] Note that, as customary in type theory, quantifications range over the rest of the expression, i.e. the above formula is equivalent to $\forall (p : (\Sigma x.\ Ax) \to \mathbb{T}).\ ((\forall xy.\ (p(x, y))) \to \forall (s : \Sigma x.\ Ax).\ (ps))$.

3 Constructive proofs

A proposition $P : \mathbb{P}$ is *stable* if it is unchanged under double negation, i.e. $\neg\neg P \to P$. Furthermore, we say that P is *logically decidable*, if $P \vee \neg P$ holds.

Fact 2. *Logically decidable propositions are stable.*

Stability properties are useful if they apply to a goal, whereas logically decidability is also useful to establish assumptions. The *law of excluded middle* LEM states that every proposition is logically decidable, and thus enables strengthening assumptions. *Markov's Principle* (MP) accepted for instance in Russian constructivism states that satisfiability of a boolean test on natural numbers is stable and is a consequence of LEM:

$$\mathsf{LEM} := \forall P : \mathbb{P}.\ P \vee \neg P \qquad \mathsf{MP} := \forall f : \mathbb{N} \to \mathbb{B}.\ \neg\neg(\exists n.\ fn = \mathsf{true}) \to (\exists n.\ fn = \mathsf{true})$$

LEM is routinely used in textbooks, often in the form of double negation elimination:

Fact 3. $\mathsf{LEM} \leftrightarrow \forall P : \mathbb{P}.\ \neg\neg P \to P$

It is folklore that LEM is independent in CIC: It can be consistently assumed, but not proved. However, when the conclusion is stable, LEM is not needed for case analysis:

Fact 4. *Let Q be stable. We have that*

1. $P \to \neg\neg P$ **2.** $\neg\neg P \to (P \to Q) \to Q$ **3.** $(P \vee \neg P \to Q) \to Q$.

Note that negated propositions are always stable, and in fact the following holds:

Fact 5. *P is stable if and only if P is equivalent to $\neg Q$ for some Q.*

MP is also independent in CIC [23, 17]. For classical theorems on enumerable predicates, often full LEM is not needed and MP suffices, see Fact 9 in the next section.

4 Synthetic Computability Theory

The key ingredients for synthetic computability theory are synthetic definitions of central notions and suitable synthetic axioms. We define synthetic decidability, enumerability, semi-decidability, and many-one reducibility [10, 6], mirroring the textbook definitions without the requirement that the witnessing function is computable in a model of computation. We recall the synthetic setting due to Forster [8] in which we work. A predicate $p : X \to \mathbb{P}$ is

- *decidable* if there exists a decider: $\mathcal{D}p := \exists f : X \to \mathbb{B}.\forall x.\ px \leftrightarrow fx = \mathsf{true}$
- *semi-decidable* if there exists a semi-decider: $\mathcal{S}p := \exists f.\forall x.\ px \leftrightarrow \exists n.fxn = \mathsf{true}$
- *enumerable* if there exists an enumerator: $\mathcal{E}p := \exists f : \mathbb{N} \to \mathbb{O}X.\forall x.\ px \leftrightarrow \exists n.\ fn = \mathsf{Some}\, x$
- *strongly enumerable* if the following holds: $\mathcal{E}_+p := \exists f : \mathbb{N} \to X.\forall x.\ px \leftrightarrow \exists n.\ fn = x$

We treat n-ary predicates as unary via (implicit) uncurrying. A type X is *discrete* if $\mathcal{D}(\lambda xy : X.x = y)$ and *enumerable* if $\mathcal{E}(\lambda x : X.\top)$. We sum up the connections of the notions:

Fact 6.
1. *Decidable predicates are semi-decidable.*
2. *Decidability is closed under (pointwise) $\wedge$, $\vee$, and $\neg$.*
3. *Finite predicates are decidable.*
4. *Semi-decidable predicates on enumerable types are enumerable.*
5. *Semi-decidability is closed under (pointwise) $\wedge$ and $\vee$.*
6. *The projections $\lambda y.\exists x.\, pxy$ and $\lambda x.\exists y.\, pxy$ of an enumerable predicate $p\colon X\to Y\to\mathbb{P}$ are enumerable.*
7. *Enumerable predicates on discrete types are semi-decidable.*
8. *Enumerability is closed under (pointwise) $\wedge$ and $\vee$.*
9. *Strongly enumerable predicates are enumerable.*
10. *Enumerable inhabited predicates are strongly enumerable.*

Fact 7.
1. *$\mathbb{N}$ and $\mathbb{B}$ are discrete and enumerable.*
2. *Both discrete and enumerable types are closed under pairs, sums, options, and lists.*

A *retraction* is an injective function with explicit inverse. X is a retract of Y if there are $I\colon X\to Y$ and $R : Y \to \mathbb{O}X$ s.t. $R(Ix) = \mathsf{Some}\, x$ and $\forall xy.\ Ry = \mathsf{Some}\, x \to y = Ix$.

Fact 8. *A type is discrete and enumerable if and only if it is a retract of $\mathbb{N}$.*

This fact enables most of our results to only mention predicates $\mathbb{N}\to\mathbb{P}$, and then transport for free to predicates on infinite, enumerable, discrete types (such as $\mathbb{L}\mathbb{B}\times\mathbb{O}\mathbb{N}$).

We can characterise MP in terms of enumerable and semi-decidable predicates:

Fact 9 ([10] 2.17, [6] 41)**.** *The following are equivalent:*

1. MP **2.** $\forall p\colon\mathbb{N}\to\mathbb{P}.\ \mathcal{S}p \to \forall x.\ \neg\neg px \to px$ **3.** $\forall p\colon X\to\mathbb{P}.\ \mathcal{E}p \to \neg\neg(\exists n.\ pn) \to \exists n.\ pn$

The following strengthening of $\mu_{\mathbb{N}}$ (Fact 1) will be crucial later:

Fact 10. *Let $p\colon\mathbb{N}\to X\to\mathbb{P}$ be enumerable and X discrete. Then there is $\mu_{\mathcal{E}}\colon\forall n.(\exists x.pnx)\to\Sigma x.pnx$.*

Note that the result of $\mu_{\mathcal{E}}$ is independent of the proof $H\colon\exists x.\, pnx$, even without assumptions.

In this paper we will define many-one, one-one, and truth-table reducibility ($\preceq_{\mathsf{m}}, \preceq_1, \preceq_{\mathsf{tt}}$). In general, for a reducibility notion $\preceq_{\mathsf{r}}$ we define predicates $p\colon X\to\mathbb{P}$ and $q\colon Y\to\mathbb{P}$ to be r-*equivalent* if $p \equiv_{\mathsf{r}} q := p \preceq_{\mathsf{r}} q \wedge q \preceq_{\mathsf{r}} p$. Informally, we might also refer to the class of predicates q s.t. $p \equiv_{\mathsf{r}} q$ as the r-degree of p, but will not make degrees formal to avoid extensionality assumptions. A predicate $p\colon\mathbb{N}\to\mathbb{P}$ is called r-*complete* if $\mathcal{E}p \wedge \forall q\colon\mathbb{N}\to\mathbb{P}.\ \mathcal{E}q \to q \preceq_r p$. We freely use the notion as well for predicates $p\colon X\to\mathbb{P}$ if X is in bijection to $\mathbb{N}$.

All notions of reducibility have in common that they form pre-orders (i.e. are reflexive and transitive) and that they transport decidability backwards, i.e. $p \preceq_r q \to \mathcal{D}q \to \mathcal{D}p$.

We now start with the first notion of synthetic reducibility: A function $f : X \to Y$ is a *many-one reduction* from $p\colon X\to\mathbb{P}$ to $q\colon Y\to\mathbb{P}$ if it expresses p in terms of q:

$$p \preceq_{\mathsf{m}} q := \exists f : X \to Y.\forall x.\ px \leftrightarrow q(fx)$$

Fact 11.
1. *Many-one reducibility forms a pre-order.*
2. *If $p \preceq_{\mathsf{m}} q$ and q is decidable then p is decidable, resp. for semi-decidability and stability.*
3. *$p \preceq_{\mathsf{m}} q \to \overline{p} \preceq_{\mathsf{m}} \overline{q}$.*

Regarding the order structure of $\preceq_{\mathsf{m}}$, one can prove that decidable predicates constitute minima for the class of non-trivial predicates, and that $\preceq_{\mathsf{m}}$ forms an upper semi-lattice:

Fact 12. *Let p be decidable. If $\exists x_1 x_2.\, qx_1 \land \neg qx_2$, then $p \preceq_{\mathsf{m}} q$.*

Fact 13. *Let $p\colon X \to \mathbb{P}$ and $q\colon Y \to \mathbb{P}$. Then there is a lowest upper bound $p+q\colon X+Y\to\mathbb{P}$ w.r.t. $\preceq_{\mathsf{m}}$: If $(p+q)(\mathsf{inl}\, x) := px$ and $(p+q)(\mathsf{inr}\, y) := qy$, then $p+q$ is the join of p and q w.r.t $\preceq_{\mathsf{m}}$, i.e. $p \preceq_{\mathsf{m}} p+q$, $q \preceq_{\mathsf{m}} p+q$, and for all r if $p \preceq_{\mathsf{m}} r$ and $q \preceq_{\mathsf{m}} r$ then $p+q \preceq_{\mathsf{m}} r$.*

In traditional computability theory, a universal machine for the chosen model of computation is central to almost all interesting results. In a synthetic approach to computability where there is no explicit model of computation and the notion of function and computable function are identified, a universal function cannot be defined. Instead, its existence has to be axiomatically assumed.

To develop synthetic computability theory agnostic towards classical axioms like LEM, we assume the enumerability axiom EA [8, 7]. The axiom EA postulates a *parametrically universal enumerator* $\varphi\colon \mathbb{N}\to(\mathbb{N}\to\mathbb{ON})$, i.e. that for all $p\colon \mathbb{N}\to\mathbb{N}\to\mathbb{P}$

$$(\exists f\colon \mathbb{N}\to(\mathbb{N}\to\mathbb{ON}).\forall i.\ f_i \text{ enumerates } p_i) \to \exists \gamma\colon \mathbb{N}\to\mathbb{N}.\forall i.\ \varphi_{\gamma i} \text{ enumerates } p_i.$$

To ease language, we often refer to just the universal property as EA in this paper.

Fact 14. $\forall p\colon \mathbb{N}\to\mathbb{P}.\ \mathcal{E}p \to \exists c.\ \varphi_c \text{ enumerates } p$

Fact 15. *Let X be enumerable and discrete and $p\colon X\times\mathbb{N}\to\mathbb{P}$ be enumerable. Then*

$$\exists \gamma\colon X\to\mathbb{N}.\forall x.\ \varphi_{\gamma x} \text{ enumerates } \lambda y.p(x,y).$$

Fact 16. *Let I be enumerable and discrete and $p\colon I\to\mathbb{N}\to\mathbb{P}$. We have*

$$(\exists f\colon I\to(\mathbb{N}\to\mathbb{ON}).\forall i.\ f_i \text{ enumerates } p_i) \to \exists \gamma\colon I\to\mathbb{N}.\forall i.\ \varphi_{\gamma i} \text{ enumerates } p_i$$

As is common in developments of computability, we start by defining an enumerable, undecidable, m-complete predicate and its diagonal:

$$\mathcal{W}_c x := \exists n.\ \varphi_c n = \mathsf{Some}\, x \qquad \mathcal{K}c := \mathcal{W}_c c$$

We call $\mathcal{W}$ the *universal table* of enumerable predicates, justified by the following property:

Fact 17. $\forall p\colon \mathbb{N}\to\mathbb{P}.\ \mathcal{E}p \leftrightarrow \exists c.\ \forall x.\ \mathcal{W}_c x \leftrightarrow px$

The universal table $\mathcal{W}$ is enumerable, undecidable and m-complete, i.e. takes the role of the halting problem from textbook computability. $\mathcal{K}$ plays a similar role as the self-halting problem, instead of the codes halting on themselves, $\mathcal{K}c$ holds if c is in the range of φ_c.

We start by showing that the complement of $\mathcal{K}$ is not enumerable. Thus $\mathcal{K}$ is undecidable, and undecidability can be transported to $\mathcal{W}$ via a many-one reduction.

Fact 18. $\neg\mathcal{E}\overline{\mathcal{K}}$ *and thus* $\neg\mathcal{D}\overline{\mathcal{K}}$ *and* $\neg\mathcal{D}\mathcal{K}$

Fact 19. $\mathcal{K} \preceq_{\mathsf{m}} \mathcal{W}$, *and thus* $\neg\mathcal{E}\overline{\mathcal{W}}$, $\neg\mathcal{D}\overline{\mathcal{W}}$, $\neg\mathcal{D}\mathcal{W}$.

To show the enumerability of both $\mathcal{K}$ and $\mathcal{W}$, we show the enumerability of $\mathcal{W}$ and again transport via the above many-one reduction, this time positively.

Fact 20. $\mathcal{E}\mathcal{W}$ *and thus* $\mathcal{E}\mathcal{K}$.

We now turn towards m-completeness of $\mathcal{W}$ and $\mathcal{K}$, i.e. that all $p\colon X\to\mathbb{P}$ for enumerable, discrete X many-one reduce to $\mathcal{W}$ and $\mathcal{K}$. To establish this for $\mathcal{K}$ for the first time requires the full strength of EA, whereas before the non-parametric Fact 14 would have sufficed.

Lemma 21. *$\mathcal{W}$ is m-complete.*

Proof. Let p be enumerable by φ_c via Fact 14. Then $\lambda x.(c,x)$ reduces p to $\mathcal{W}$. ◀

Lemma 22. $\mathcal{W} \preceq_{\mathsf{m}} \mathcal{K}$

Proof. We obtain the reduction function γ from Fact 16 with $p(x,y)z := \mathcal{W}_x y$. Since $\forall xyz.\, \mathcal{W}_{\gamma(x,y)} z \leftrightarrow \mathcal{W}_x y$ we have $\mathcal{W}_x y \leftrightarrow \mathcal{W}_{\gamma(x,y)}(\gamma(x,y)) \leftrightarrow \mathcal{K}(\gamma(x,y))$. ◀

Fact 23. *$\mathcal{W} \equiv_{\mathsf{m}} \mathcal{K}$ and $\mathcal{K}$ is m-complete.*

We can recover the characterisation of MP as stability of halting:

Fact 24. MP *is equivalent to the stability of* **1.** *$\mathcal{W}$,* **2.** *$\mathcal{K}$,* **3.** *any enumerable, m-complete predicate on $\mathbb{N}$, and* **4.** *every enumerable, m-complete predicate on $\mathbb{N}$.*

5 Finite predicates

We discuss two notions of finiteness: Finite (often Bishop- or $\mathcal{B}$-finite) and subfinite (also $\widetilde{\mathcal{B}}$-finite) predicates. We say that a list $l : \mathbb{L}X$ *lists* a predicate $p\colon X\to\mathbb{P}$ if $px \leftrightarrow x \in l$. Note that l is not allowed to contain elements not fulfilled by the predicate. A list $l : \mathbb{L}X$ *exhausts* a predicate $p\colon X\to\mathbb{P}$ if $px \to x \in l$: We write $\mathcal{L}p$ for finite and $\mathcal{X}p$ for subfinite predicates p.

$$\mathcal{L}p := \exists l : \mathbb{L}X.\forall x : X.\ px \leftrightarrow x \in l \qquad\qquad \mathcal{X}p := \exists l : \mathbb{L}X.\ \forall x : X.\ px \to x \in l$$

Clearly, finiteness implies subfiniteness, but the converse is equivalent to LEM:

Fact 25. *Finite predicates are subfinite.*

Lemma 26. *Every subfinite predicate is not not finite.*

Proof. We first prove the lemma that $\forall l_0 : \mathbb{L}X.\forall p\colon X\to\mathbb{P}.\neg\neg\exists l'.\forall x.\ x \in l' \leftrightarrow x \in l_0 \wedge px$ which follows by induction on l_0. Now, let l exhaust p and let p be not finite. We have to prove falsity. Using the lemma, we obtain l' s.t. $\forall x.\ x \in l' \leftrightarrow x \in l \wedge px$ and still have to prove falsity. Now l' lists p. Contradiction. ◀

Lemma 27. *If every subfinite predicate is finite,* LEM *holds.*

Proof. For $P : \mathbb{P}$ we define $p(x : \mathbb{B}) := P$. p is subfinite because it is exhausted by $[\mathsf{true}, \mathsf{false}]$. If p is listed by l, case analysis on l allows proving $P \vee \neg P$. ◀

Corollary 28. LEM *holds if and only if every subfinite predicate is finite.*

6 Pigeonhole principles

Pigeonhole principles are omnipresent in discrete mathematics. We will prove three variants of the pigeonhole principle based on duplicate-free lists: Our formulation of the principle is that for any duplicate-free list l_1 longer than a list l_2 one can obtain an element x which is in l_1 but not in l_2 – for different formalisations of "obtain" in CIC[2]: **1.** x is computable

[2] Admittedly, this formulation is more a "pigeon-less hole principle", but we use the well-known terminology.

$(\forall l_1 l_2.\ \cdots \to \Sigma x.\ \ldots)$, **2.** x constructively exists $(\forall l_1 l_2.\ \cdots \to \exists x.\ \ldots)$, **3.** x classically exists $(\forall l_1 l_2.\ \cdots \to \neg\neg\exists x.\ \ldots)$. These are exactly the three possible formalisations of "obtain" in CIC: A function returning a dependent pair (Σ), a proof of an existential proposition ($\exists$), or of a double-negated existential proposition ($\neg\neg\exists$, equivalently $\neg\neg\Sigma$). Formulating existence as Σ inherently means that the result has to be *computable*, a property unchanged by the assumption of logical axioms like LEM. $\exists$ has to be proved using a (computable) function, but the function cannot be used computationally after the proof. This means that any constructive proof of $\forall x.\exists y.\ldots$ not using assumptions could always be turned into a proof of $\forall x.\Sigma y$ and vice versa, but under assumptions and as assumptions the two behave differently, see Section 2 for more details.

We now turn towards proving the principles, which will vary in the requirements on the underlying type X. For the formalisation, we define $\#l : \mathbb{P}$ for a list $l : \mathbb{L}X$ inductively in the expected way to state that l does not contain any duplicates.

Given an equality decider for the base type X it is straightforward to prove the $\forall\Sigma$-version of the pigeonhole principle:

Lemma 29. *Let d decide equality on X and $l_1, l_2 : \mathbb{L}X$. If $\#l_1$ and $|l_1| > |l_2|$, then $\Sigma x.\ x \in l_1 \land x \notin l_2$.*

Proof. By induction on $\#l_1$, with l_2 generalised. The first case is contradictory, since $|[]| = 0 > |l_2|$ is impossible. Let $x \notin l_1$ and $\#l_1$. Using d allows a case analysis whether $x \in l_2$ or $x \notin l_2$: If $x \notin l_2$, the claim is immediate. If $x \in l_2$, the claim follows from the induction hypothesis for $l_2 := \mathsf{filter}(\lambda y.\textbf{if}\ dxy\ \textbf{then}\ \mathsf{false}\ \textbf{else}\ \mathsf{true})\ l_2$ (i.e. for l_2 with x removed). ◀

Note how the $\forall\Sigma$ version depends on computationally removing an element from a list, and thus on an equality decider d. If no such d is available, a removal function is not definable. However, for the $\forall\exists$ and $\forall\neg\neg\exists$ forms of the pigeonhole principle, a removal *function* is not needed. Instead, for the $\forall\exists$ version it suffices to prove that for any list l_0 and any element x_0, there exists ($\exists$) a list with the same elements of l_0, just x_0 removed. This becomes possible provided $x_1 \neq x_2$ is logically decidable for all x_1, x_2.For the $\forall\neg\neg\exists$ version of the pigeonhole principle, consequently a removal principle of the form $\neg\neg\exists l\ldots$ suffices, which can be proved fully constructively without assumptions.

To prove the two removal principles, we define a generalised filter predicate $l_0 \supseteq_p l$ w.r.t. a predicate $p\colon X \to \mathbb{P}$ stating that l is exactly the sublist of l_0 with all elements which fulfil p:

$$\frac{}{[] \supseteq_p []} \qquad \frac{px \qquad l_0 \supseteq_p l}{(x :: l_0) \supseteq_p (x :: l)} \qquad \frac{\neg px \qquad l_0 \supseteq_p l}{(x :: l_0) \supseteq_p l}$$

We can prove two existence principles, one assuming that p is logically decidable, and one proving a double negation:

Fact 30. *Let $l_0 : \mathbb{L}X$. Then (1) $\neg\neg\exists l.\ l_0 \supseteq_p l$ and (2) $(\forall x.px \lor \neg px) \to \exists l.\ l_0 \supseteq_p l$.*

The $\forall\exists$ and $\forall\neg\neg\exists$ forms of the pigeonhole principle follow:

Lemma 31. *If $\#l_1$ and $|l_1| > |l_2|$, then $\neg\neg\exists x.x \in l_1 \land x \notin l_2$.*

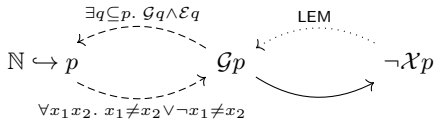
Proof. By induction on $\#l_1$, with l_2 generalised. The case $l_1 = []$ is contradictory since then $|[]| = 0 > |l_2|$. Thus let $x \notin l_1$ and $\#l_1$. Since the claim is negative, we can do a case analysis on $x \in l_2$ by Fact 4 **3**. If $x \notin l_2$, the claim is immediate. If $x \in l_2$, we obtain l s.t. $l_2 \supseteq_{(\lambda y.x \neq y)} l$ from Fact 30 (1). The claim follows by induction for l. ◀

Lemma 32. *If $\forall x_1 x_2\colon X.x_1 \neq x_2 \lor \neg x_1 \neq x_2$, $\#l_1$ and $|l_1| > |l_2|$, then $\exists x.\ x \in l_1 \land x \notin l_2$.*

Proof. Similar to the last proof, using Fact 30 (2) and the assumption. ◀

7 Infinite Predicates

The central notions of simple and hypersimple predicates depend on a formalisation of infinity. Similar to finite predicates, there are several classically equivalent but constructively different definitions of infinite predicates. We discuss three possible definitions of infinity, where a predicate $p\colon X\to\mathbb{P}$ is non-finite if it is not finite ($\neg\mathcal{L}p$), or equivalently not subfinite ($\neg\mathcal{X}p$), both stable propositions, generative ($\mathcal{G}p$) if for every list there exists a further element fulfilling p not in the list (a $\forall\exists$ proposition), and Cantor-infinite ($\mathbb{N}\hookrightarrow p$) if there exists an injection returning only elements in p (equivalent to a $\forall\Sigma$ proposition). Those three notions of infinite predicates mirror exactly the three different formulations of pigeonhole principles in the previous chapter. We obtain the following graph, where the dashed (dotted) lines are annotated with sufficient (and necessary) conditions:



Formally, a predicate $p\colon X\to\mathbb{P}$ is called *non-finite* if is not finite. A predicate $p\colon X\to\mathbb{P}$ is *finite* (written $\mathcal{L}p$) if $\exists l\colon\mathbb{L}X.\ \forall x.\ px\leftrightarrow x\in l$.

Due to the negation, non-finiteness is equivalent to non-sub-finiteness. A predicate $p\colon X\to\mathbb{P}$ is *subfinite* (written $\mathcal{X}p$) if $\exists l\colon\mathbb{L}X.\ \forall x.\ px\to x\in l$. We discuss more properties on finite predicates in Section 5.

Fact 33. $\neg\mathcal{X}p\leftrightarrow\neg\mathcal{L}p$.

A predicate $p\colon X\to\mathbb{P}$ is called *generative* if for every list there exists a further element fulfilling p which is not in the list:

$$\mathcal{G}p:=\forall l:\mathbb{L}X.\exists x.\ px\wedge x\notin l$$

Fact 34. *Generative predicates are inhabited, and non-finite predicates not not inhabited.*

Generative predicates on $\mathbb{N}$ can be characterised as follows:

Fact 35. $\forall p\colon\mathbb{N}\to\mathbb{P}.\ \mathcal{G}p\leftrightarrow\forall n.\exists m\geq n.\ pm$

The following characterisation of non-finiteness as $\forall\neg\neg\exists$-formula connects the two notions:

Fact 36. *If* $\forall x_1x_2:X.\ x_1=x_2\vee x_1\neq x_2$, *then* $\neg\mathcal{X}p\leftrightarrow\forall l:\mathbb{L}X.\neg\neg\exists x.\ px\wedge x\notin l$.

Corollary 37. $\forall p\colon\mathbb{N}\to\mathbb{P}.\ \neg\mathcal{X}p\leftrightarrow\forall n.\neg\neg\exists m\geq n.pm$

Since the proof of the direction from right to left of Fact 36 does not depend on logical decidability of equality we can prove the following:

Fact 38. *Generative predicates are non-finite.*

Fact 39. *If* LEM *holds and p is non-finite, p is generative.*

We introduce a second $\forall\exists$-notion of infinite predicates: $p\colon X\to\mathbb{P}$ is called *unbounded* if there exist duplicate-free lists of arbitrary length containing only elements from p:

$$\mathcal{U}p:=\forall n:\mathbb{N}.\exists l.\ |l|=n\wedge\#l\wedge\forall x\in l.\ px$$

Fact 40. $\neg\mathcal{X}p \leftrightarrow \forall n : \mathbb{N}.\neg\neg\exists l.\ |l| = n \wedge \#l \wedge \forall x \in l.\ px$

Lemma 41. *Generative predicates are unbounded.*

Proof. Let p be generative and $n : \mathbb{N}$. We construct l by induction on n. For $n = 0$ we pick $l = []$. For $n = \mathsf{S}n'$ we use the inductive hypothesis to obtain l, and use generativity to obtain x s.t. px and $x \notin l$. Now $x :: l$ is the wanted list. ◀

We then can prove the following using Lemmas 31 and 32.

Fact 42. *Unbounded predicates are non-finite, and are generative for types X with $\forall x_1 x_2 : X.\ x_1 \neq x_2 \vee \neg x_1 \neq x_2$.*

LEM is necessary for non-finite predicates to be unbounded:

Lemma 43. *If all non-finite $p : \mathbb{N} \to \mathbb{P}$ are unbounded,* LEM *holds.*

Proof. Let $P : \mathbb{P}$ and $pn := P \leftrightarrow n$ *is even*. If p is inhabited, LEM holds: Given pn we can analysing whether n is even. Since unbounded predicates are inhabited, by assumption it suffices to prove that p is non-finite. So let p be exhaustible, we have to prove falsity. We can thus decide P and it suffices to prove that p is generative to obtain a contradiction.

If P holds, $pn \leftrightarrow n$ *is even* and if $\neg P$, $pn \leftrightarrow n$ *is odd*. Both are generative. ◀

Corollary 44. *Non-finite predicates are unbounded or generative if and only if* LEM *holds.*

Lastly, we introduce Cantor infinity, often used in textbooks. A predicate $p : X \to \mathbb{P}$ is *Cantor-infinite* if there exists an injection $f : \mathbb{N} \to X$ only returning elements in p:

$$\mathbb{N} \hookrightarrow p := \exists f : \mathbb{N} \to X.\ f \text{ is injective} \wedge \forall n.\ p(fn),$$

whereby we define as also in the remainder of the paper a function $f : A \to B$ to be injective if $\forall a_1, a_2.\ f(a_1) = f(a_2) \to a_1 = a_2$.

Fact 45. *Cantor-infinite predicates are unbounded.*

Lemma 46. *Let X be discrete and $p : X \to \mathbb{P}$. Then $\mathbb{N} \hookrightarrow p \leftrightarrow \forall l : \mathbb{L}X.\Sigma x.\ px \wedge x \notin l$.*

Proof. The forward direction is easy using Lemma 29. Conversely, let $F : \forall l : \mathbb{L}X.\Sigma x.\ px \wedge x \notin l$ and $fl := \pi_1(Fl)$. Let $g0 := []$ and $g(\mathsf{S}n) := gn +\!\!+ [f(gn)]$. Then pick $\lambda n.f(gn)$. ◀

We have proven that non-finite predicates are generative if and only if LEM holds. However, non-finite, enumerable predicates on $\mathbb{N}$ are generative already given MP:

Lemma 47. *Assume* MP *and let $p : \mathbb{N} \to \mathbb{P}$. If p is enumerable and non-finite, p is generative.*

Proof. Let p be enumerable and non-finite, and let l be given. We have to prove $\exists x.\ x \notin l \wedge px$. Since p is enumerable, so is $\lambda x.\ x \notin l \wedge px$. Using Fact 9 it suffices to prove $\neg\neg\exists x.\ x \notin l \wedge px$, which holds by Fact 36. ◀

We adapt [10, Th. 2.28] to prove that generative enumerable predicates are Cantor-infinite:

Fact 48. *Let X be discrete. Generative, enumerable predicates over X have a strong, injective enumerator: $\forall p : X \to \mathbb{P}.\ \mathcal{G}p \to \mathcal{E}p \to \exists f.\ f \text{injective} \wedge \forall x.\ px \leftrightarrow \exists n.\ fn = x$.*

Corollary 49. *Generative, enumerable predicates over discrete types are Cantor-infinite.*

In synthetic computability, Cantor-infinite predicates are problematic because the function $f\colon \mathbb{N}\to X$ can be turned into an enumerator. From $\mathbb{N} \hookrightarrow p$ we cannot conclude that p is enumerable, but that p has a Cantor-infinite, enumerable subpredicate:

Lemma 50. $\mathbb{N} \hookrightarrow p \to \exists q.\ \mathcal{E}q \wedge (\forall x.\ qx \to px) \wedge \mathbb{N} \hookrightarrow q$

Proof. Given p and an injection f witnessing $\mathbb{N} \hookrightarrow p$, define $qx := \exists n.\ fn = x$. Clearly, $\forall x.\ qx \to px$ since $\forall n.\ p(fn)$, and $\lambda n.\ \mathsf{Some}\,(fn)$ enumerates q. f still proves $\mathbb{N} \hookrightarrow q$. ◀

Recall that a predicate is simple if it is enumerable, and its complement is infinite but does not have an infinite enumerable subpredicate. Given the last lemma, we have that under the assumption of a universal enumerator φ, defining infinity as...

1. Cantor infinity proves there is no simple predicate.
2. generativity makes the existence of simple predicates logically independent, since any constructive proof of generativity could be turned into a proof of Cantor-infinity by Lemma 46, but this is not provable,
3. non-finiteness allows to construct a simple predicate.

8 One-one reducibility

We now introduce our second notion of reducibility: One-one reducibility is a special case of many-one reducibility. A function $f\colon X\to Y$ is a *one-one reduction* from p to q if f is an injective many-one reduction from p to q:

$$p \preceq_1 q := \exists f : X \to Y.\ f \text{ is injective} \wedge \forall x.\ px \leftrightarrow q(fx)$$

Fact 51. 1. *One-one reducibility forms a pre-order,* 2. *implies many-one reducibility, and* 3. *transports decidability, semi-decidability, and stability backwards.*

It is easy to prove that one-one and many-one reducibility are not equivalent:

Fact 52. *If* $px := \top$ *and* $qx := x = 0$, $p \preceq_{\mathsf{m}} q$ *but* $q \not\preceq_1 p$.

Note that the lemma leaves open whether there are two enumerable but *undecidable* predicates p and q s.t. $p \preceq_{\mathsf{m}} q$ but $p \not\preceq_1 q$. We settle this more interesting case in Corollary 68 via simple predicates. In Appendix A, we characterise m-reducibility in terms of 1-reducibility via so-called cylindrification, adapting the proof by Rogers [27, §7.6 Th. VIII].

9 Simple predicates

We now turn to Post's problem for $\preceq_{\mathsf{m}}$, i.e. to finding an enumerable, undecidable, m-incomplete predicate S i.e. $\mathcal{W} \not\preceq_{\mathsf{m}} S$. Post's observation was that the complements of m-complete predicates are productive. It thus suffices to find an enumerable predicate with non-productive complement. A predicate p is productive if its non-enumerability is witnessed by a function f s.t. for every enumerable subpredicate $\mathcal{W}_c$ of p, p and $\mathcal{W}_c$ differ on fc.

$$\mathsf{productive}\,(p\colon \mathbb{N}\to\mathbb{P}) := \exists f : \mathbb{N} \to \mathbb{N}.\forall c.\ (\forall x.\ \mathcal{W}_c x \to px) \to p(fc) \wedge \neg\mathcal{W}_c(fc)$$

Lemma 53. *Productive predicates are not enumerable.*

Proof. Let p be enumerable, i.e. $px \leftrightarrow \mathcal{W}_c x$ by Fact 17. Let f be a productive function for p, i.e. $p(fc)$ and thus $\mathcal{W}_c(fc)$, and $\neg\mathcal{W}_c(fc)$. Contradiction. ◀

Lemma 54. *$\overline{\mathcal{K}}$ is productive.*

Proof. Pick $\lambda n.n$. Let c be s.t. $\forall x.\ \mathcal{W}_c x \to \overline{\mathcal{K}} x$. Then $\mathcal{W}_c c \to \neg \mathcal{W}_c c$. Contradiction. ◀

Every productive predicate is Cantor-infinite and thus has an enumerable, Cantor-infinite subpredicate.

Lemma 55. *Every productive predicate is Cantor-infinite.*

Proof. From Fact 16 and since $x \in l$ is decidable, we obtain $c : \mathbb{L}\mathbb{N} \to \mathbb{N}$ s.t. $\mathcal{W}_{cl} x \leftrightarrow x \in l$ for every $l : \mathbb{L}\mathbb{N}$. Let p have a productive function f. We prove $\forall l : \mathbb{L}\mathbb{N}.\Sigma x.\ (\forall x_0 \in l. p x_0) \to p x \wedge x \notin l$, which suffices by a slight adaption of Lemma 46. Given $l : \mathbb{L}\mathbb{N}$, pick $x := f(cl)$. ◀

Corollary 56. *Every productive predicate has an enumerable, Cantor-infinite subpredicate.*

We can now show that the complement of m-complete predicates contains an enumerable, Cantor-infinite subpredicate since productiveness transports along many-one reductions:

Lemma 57. *Let $p \preceq_{\mathsf{m}} q$. If p is productive, q is productive.*

Proof. Let f many-one reduce p to q, and let g be a productive function for p. Fact 16 yields k s.t. $\mathcal{W}_{(kc)} x \leftrightarrow \mathcal{W}_c (f x)$. Then $\lambda c.\ f(g(kc))$ is a productive function for q. ◀

Lemma 58. *Let p be m-complete. Then $\overline{p}$ has an enumerable, Cantor-infinite subpredicate.*

Proof. Since productiveness of $\overline{p}$ follows from $\overline{\mathcal{K}} \preceq_{\mathsf{m}} \overline{p}$ and productiveness of $\overline{\mathcal{K}}$. ◀

In particular, then the complement of an m-complete predicate has a non-finite, enumerable subpredicate. Post's idea to find an enumerable, but undecidable, many-one *incomplete* predicate hinges on exactly this observation. We follow Post and define a predicate $p : X \to \mathbb{P}$ to be *simple* if it is enumerable, and its complement is both non-finite and does *not* contain a non-finite, enumerable predicate:

$$\mathsf{simple}\, p := \mathcal{E} p \wedge \neg \mathcal{X} \overline{p} \wedge \neg \exists q.\ (\forall x.\ q x \to \overline{p} x) \wedge \neg \mathcal{X} q \wedge \mathcal{E} q$$

Fact 59. *Complements of simple predicates are not enumerable.*

Corollary 60. *Simple predicates are undecidable.*

Lemma 61. *Simple predicates are m-incomplete.*

We follow the presentation of Rogers [27, §8.1 Th. II] to construct the same simple predicate as Post [24, §5]. The latter two properties of a simple predicate (non-finite complement and may not contain a non-finite, enumerable subpredicate) will drive the construction, since either one is easy to establish on their own, but the combination needs care. Post's idea was to construct a predicate S containing an element from every non-finite (enumerable) predicate $\mathcal{W}_c$. Thus, $\overline{S}$ cannot have a non-finite, enumerable subpredicate. To ensure that $\overline{S}$ is still non-finite, S contains only a *unique* $x > 2c$ with $\mathcal{W}_c x$ for every large enough $\mathcal{W}_c$. The condition $x > 2c$ ensures that there are at least n elements less or equal $2n$ in $\overline{S}$, and thus $\overline{S}$ is non-finite.

The only technical difficulty in the definition of S is to obtain a unique x satisfying $x > 2c$ and $\mathcal{W}_c x$ for every large enough $\mathcal{W}_c$. Post ensures this by choosing the x enumerated first by φ_c (i.e. the x with the least index n s.t. $\varphi_c n = \mathsf{Some}\, x$) satisfying $x > 2c$. We abstract away from this property, and observe that any function mapping c to an $x > 2c$ does the job.

We fix a function $\psi : \forall c.\ (\exists x.\ \mathcal{W}_c x \land x > 2c) \to \mathbb{N}$ s.t. $\psi c H = x \to \mathcal{W}_c x \land x > 2c$ and $\psi c H_1 = \psi c H_2$, i.e. a proof-irrelevant choice function for the predicate $\lambda x.\ \mathcal{W}_c x \land x > 2c$. Such a choice function ψ can be constructed by using for instance $\mu_{\mathcal{E}}$ (Fact 10).

We then define S as the image of ψ and prove that S is a simple predicate:

$$Sx := \exists c{:}\mathbb{N}.\exists H : (\exists y.\ \mathcal{W}_c y \land y > 2c).\ \psi c H = x$$

Lemma 62. *S is enumerable.*

Proof. There is a strong enumerator $E : \mathbb{N} \to \mathbb{N}$ for $\lambda c.\exists x.\mathcal{W}_c x \land x > 2c$ and thus we have $H{:}\forall n.\exists x.\mathcal{W}_{(En)} x \land x > 2 \cdot (En)$. Then, $\lambda n.\psi(En)(Hn)$ strongly enumerates S. ◀

Lemma 63. *$\overline{S}$ is non-finite.*

Proof. By Fact 40 it suffices to prove $\forall n.\neg\neg\exists L.|L| = n \land \#L \land \forall x \in L.\overline{S}x$. Given n, let $px := Sx \land x \leq 2n$. p is exhausted by $[0, \ldots, 2n]$, thus it is not not finite. Since the claim is negative, we can assume some duplicate-free L_p listing p.

Now $|L_p| \leq n$: by decomposing Sx for every $x \in L_p$, we obtain L'_p with $\#L'_p$, $|L'_p| = |L_p|$, and $\forall c \in L'_p.\ c < n \land \exists H.\ \psi c H \in L_p$. Hence, $|L_p| \leq n$. Now the first n elements of $\mathsf{filter}(\lambda x.\ x \notin L_p)\,[0, \ldots, 2n]$ form the wanted list. ◀

Lemma 64. *$\overline{S}$ contains no non-finite, enumerable subset.*

Proof. Let q be non-finite, contained in $\overline{S}$ and enumerated by φ_c via Fact 10. We derive a contradiction by showing $[0, \ldots, 2c]$ to exhaust q: Assume qx. Then $\mathcal{W}_c x$ since φ_c enumerates q. We have to prove $x \in [0, \ldots, 2c]$, which is stable. So let $x \notin [0, \ldots, 2c]$ and derive a contradiction. Since $\mathcal{W}_c x \land x > 2c$ holds, there is in particular a proof $H : \exists x.\ \mathcal{W}_c x \land x > 2c$. By definition of ψ, we have $\mathcal{W}_c(\psi c H)$. By definition of φ_c, $q(\psi c H)$, and thus $\neg S(\psi c H)$, i.e. $\neg\exists c' H'.\psi c' H' = \psi c H$ – contradiction. ◀

Theorem 65. *S is a simple predicate.*

Proof. Direct by Lemmas 62, 63, and 64. ◀

Based on our definitions and work, Forster, Kunze, and Lauermann [12] prove that nonrandom numbers, defined via Kolmogorov complexity, also are simple. The construction of any of these simple predicates settles Post's problem for $\preceq_{\mathsf{m}}$:

Theorem 66. *There exists an enumerable, undecidable and m-incomplete predicate.*

Proof. Direct by Corollary 60, Lemma 61, and Theorem 65. ◀

In the following we write $p \times \mathbb{N}$ for the predicate $\lambda(x, n){:}\mathbb{N} \times \mathbb{N}.px$ and use the following observation to prove that $\preceq_{\mathsf{m}}$ and $\preceq_1$ differ on enumerable, undecidable predicates.

Lemma 67. *Given $p{:}\mathbb{N}{\to}\mathbb{P}$ and $p \times \mathbb{N} \preceq_1 p$, p is not simple.*

Proof. Let p be simple and f a one-one reduction from $p \times (\lambda x : \mathbb{N}.\top)$ to p. We have to prove falsity, thus we can assume an element x_0 s.t. $\overline{p}x_0$ since $\overline{p}$ is non-finite by Fact 34. Now $\lambda x.\exists n.f(x_0, n) = x$ is a non-finite, enumerable subpredicate of $\overline{p}$. ◀

Corollary 68. *$\preceq_1$ and $\preceq_{\mathsf{m}}$ differ on enumerable, undecidable predicates.*

Proof. Let S be a simple predicate. Now $S \times \mathbb{N} \preceq_{\mathsf{m}} S$ with $\lambda(x, n).\ x$, but $S \times \mathbb{N} \not\preceq_1 S$ by Lemma 67. ◀

10 Truth-table reducibility

Recall that p is many-one reducible to q if a decision for px can be computed from one instance of q, namely $q(fx)$. Truth-table reducibility generalises this intuition: A predicate p is truth-table reducible to q if a decision for px can be computed by evaluating a boolean formula with atoms of the form qy_i for finitely many queries y_i. Equivalently, boolean formulas with n inputs can also be expressed as truth-tables with 2^n rows, explaining the name "truth-table reducibility". We fix a canonical listing function $\mathsf{gen}\colon \mathbb{N}\to\mathbb{L}(\mathbb{LB})$of boolean lists of length n, with $|\mathsf{gen}\, n| = 2^n$ and $\forall l\colon \mathbb{LB}.\ l \in \mathsf{gen}\, n \leftrightarrow |l| = n$.

We model truth-tables T with n inputs as boolean lists and define the value of T on input l as the i-th element of T if l is the i-th element of $\mathsf{gen}\, n$. Formally, $\mathsf{truthtable} := \mathbb{LB}$ and for $l\colon \mathbb{LB}$ of length n:

$$l \vDash T := \exists i.\ T[i] = \mathsf{Some}\,\mathsf{true} \wedge (\mathsf{gen}\,|l|)[i] = \mathsf{Some}\, l$$

When convenient, we assume $(l \vDash T)\colon \mathbb{B}$ since $\lambda lT.\ l \vDash T$ is decidable. Any $f\colon \mathbb{LB}\to\mathbb{B}$ can be converted into the truth-table $\mathsf{map} f(\mathsf{gen}\, n)$ with n inputs s.t. $l \vDash \mathsf{map} f(\mathsf{gen}\, n) \leftrightarrow fl = \mathsf{true}$ provided $|l| = n$. On paper, we abuse notation and treat any function $f\colon \mathbb{LB}\to\mathbb{B}$ as truth-table.

A function $f\colon X\to\mathbb{L}Y \times \mathsf{truthtable}$ is a *truth-table reduction* from p to q if $\forall x.px \leftrightarrow l \vDash \pi_2(fx)$ for all lists l which reflect q pointwise on the query list $\pi_1(fx)$. A predicate $p\colon X\to\mathbb{P}$ is *truth-table reducible* to a predicate $q\colon Y\to\mathbb{P}$ if there is a truth-table reduction:

$$p \preceq_{\mathsf{tt}} q := \exists f\colon X\to\mathbb{L}Y \times \mathsf{truthtable}.\forall x\ l.\ \mathsf{Forall}_2\,(\lambda yb.\ qy \leftrightarrow b = \mathsf{true})\,(\pi_1(fx))\,l \to px \leftrightarrow l \vDash \pi_2(fx)$$

Lemma 69.
1. *Truth-table reducibility forms a pre-order.*
2. *If $p \preceq_{\mathsf{tt}} q$ and q is decidable then p is decidable.*
3. *If $p \preceq_{\mathsf{m}} q$ then $p \preceq_{\mathsf{tt}} q$.*

Truth-table reducibility can employ negation and thus does not transport enumerability.

Fact 70. $\overline{p} \preceq_{\mathsf{tt}} p$

Fact 71. *Truth-table reducibility is an upper semi-lattice.*

11 Hypersimple predicates

For settling Post's problem w.r.t. $\preceq_{\mathsf{tt}}$ in our synthetic setting we once more follow Rogers [27, §9.5] and introduce majorising functions: $f\colon \mathbb{N}\to\mathbb{N}$ *majorises* a predicate $p\colon \mathbb{N}\to\mathbb{P}$ if

$$\forall n.\neg\neg\exists l\colon \mathbb{LN}.\ \#l \wedge |l| = n \wedge \forall m \in l.pm \wedge m \le fn.$$

Note that we slightly adapted the definition of majorising in order to be more suitable for constructive proofs. We immediately introduce a strengthening of the notion following Odifreddi [21]: A function $f\colon \mathbb{N}\to\mathbb{N}$ *exceeds* a predicate $p\colon \mathbb{N}\to\mathbb{P}$ if $\forall n.\neg\neg\exists i.\ n < i \le fn \wedge pi$.

Fact 72. *If f exceeds p, $\lambda n.f^n 0$ majorises p.*

Lemma 73. *If $\forall x.qx \to px$ and q is Cantor-infinite, there exists f exceeding p.*

Proof. Take $\lambda n.\ \pi_1(F[0,\dots,n])$, where F is obtained from the $\forall\Sigma$ formula in Lemma 46. ◀

Corollary 74. *Given* MP*, if there is no f majorising p, then $\overline{p}$ does not have a non-finite, enumerable subpredicate.*

Proof. Let no f majorise p and let q be a non-finite, enumerable subpredicate of $\overline{p}$. By MP and Lemma 47, q is generative. By Corollary 49, q is Cantor-infinite. By Lemma 73 there is f exceeding p. By Fact 72, p is majorised – contradiction. ◀

The following is an adaption of Th. III.3.10 in the book by Odifreddi [21], with

$$p \vDash (P,T) := \forall l.\ \mathsf{Forall}_2\,(\lambda x b.\ px \leftrightarrow b = \mathsf{true})\,P\,l \to l \vDash T.$$

Lemma 75. *If $\mathcal{K} \preceq_{\mathsf{tt}} p$ there exists a function exceeding $\overline{p}$.*

Proof. Let g be a tt-reduction from $\mathcal{K}$ to p. By Fact 16 there is $c\colon \mathbb{L}\mathbb{N}\to\mathbb{N}$ s.t. $\forall lx.\ \mathcal{W}(cl)x \leftrightarrow \neg((\lambda y.y \notin l) \vDash gx)$. Let $\mathsf{gen}_n : \mathbb{L}(\mathbb{L}\mathbb{N})$ contain exactly all duplicate-free lists l s.t. $\mathsf{max}\ l \leq n$. We define $a_{n,i} := c(\mathsf{gen}_n[i])$ for $i < 2^n$ and $a_{n,i} := c[]$ otherwise. Then $\lambda n.\ 1 + \mathsf{max}\,[\pi_1(ga_{n,i}) \mid i < 2^n]$ exceeds $\overline{p}$. To show this, let n be given and assume

$$(*) : \forall j.\ n < j < \mathsf{max}\,[\pi_1(ga_{n,i}) \mid i < 2^n] \to pj$$

We have to prove falsity. Note that $\neg\neg\exists.i < 2^n.\ \forall z.\ z \notin \mathsf{gen}_n[i] \leftrightarrow p^*z$ where $p^*z := (\neg\neg pz \wedge z \leq n) \vee z > n$. Since we have to prove falsity, we can assume such an i. Now by $(*)$ we have $\forall j.n < j \in \pi_1(ga_{n,i}) \to pj$ since $\pi_1(ga_{n,i}) < \mathsf{max}\,[\pi_1(ga_{n,i}) \mid i < 2^n]$ follows from $i < 2^n$. Since for $x \leq n$, $px \leftrightarrow p^*x$ by definition, we have $\forall x \in \pi_1(ga_{n,i}).\ px \leftrightarrow p^*x$. But now we obtain the following contradiction:

$$\begin{aligned} p \vDash ga_{n,i} &\leftrightarrow \mathcal{W}\,a_{n,i}\,a_{n,i} \leftrightarrow \mathcal{W}\,(c(\mathsf{gen}_n[i]))\,a_{n,i} \\ &\leftrightarrow \neg(\overline{\mathsf{gen}_n[i]} \vDash ga_{n,i}) \leftrightarrow \neg(p^* \vDash ga_{n,i}) \\ &\leftrightarrow \neg(p \vDash ga_{n,i}) \end{aligned}$$

□

A predicate p is *hypersimple* if it is enumerable and $\overline{p}$ is non-finite and not majorised:

$$\mathsf{hypersimple}\,(p\colon\mathbb{N}\to\mathbb{P}) := \mathcal{E}p \wedge \neg\mathcal{X}\overline{p} \wedge \neg\exists f.f \text{ majorises } \overline{p}$$

Fact 76. *Hypersimple predicates are not* tt*-complete.*

Fact 77. *Given* MP*, hypersimple predicates are simple.*

12 Construction of a hypersimple predicate

We now construct and verify a hypersimple predicate, a result due to Post [24, §9]. We however follow Rogers [27, §8.1 Th. II], who presents a (more general) construction due to Dekker [5], defining a hypersimple predicate H_I for an arbitrary undecidable $I\colon\mathbb{N}\to\mathbb{P}$ with a strong, injective enumerator $E_I\colon\mathbb{N}\to\mathbb{N}$. By instantiating with e.g. $\mathcal{W}' := \lambda\langle c,x\rangle.\mathcal{W}_c x$, we obtain that $H_{\mathcal{W}'}$ is hypersimple, and thus enumerable, undecidable and tt-incomplete. The predicate $H_I\colon\mathbb{N}\to\mathbb{P}$ is defined as the so-called "deficiency predicate" of I:

$$H_I x := \exists x_0 > x.\ E_I x_0 < E_I x$$

Lemma 78. *$\overline{H_I}$ is non-finite.*

Proof. By Corollary 37 it suffices to prove $\forall x.\neg\neg\exists y \geq x.\overline{H_I}y$. We use complete induction on $E_I x$. Given $x\colon\mathbb{N}$ assume $(*)\colon \neg\exists y{\geq}x.\overline{H_I}y$. The claim is negative. Case analysis:

1. If $H_I x$, there exists $x_0 > x$ with $E_I x_0 < E_I x$. Hence, induction for x_0 yields $\neg\neg\exists y \geq x_0.\overline{H_I}y$ contradicting $(*)$.
2. If $\overline{H_I}x$ holds, x is an element as required. ◀

Lemma 79. *If f majorises $\overline{H_I}$, I is decidable.*

Proof. We prove that $g := \lambda x.x \in_{\mathbb{B}} \mathsf{map}\ E_I\ [0,\dots,f(\mathsf{S}\,x)]$ decides I if f majorises $\overline{H_I}$, i.e. $\forall x.\ Ix \leftrightarrow gx = \mathsf{true}$. The direction from right to left is easy, since E_I enumerates I. For the converse direction let $E_I n = x$ for some n. We show $n \in [0,\dots,f(\mathsf{S}\,x)]$, which is stable. Thus let $n \notin [0,\dots,f(\mathsf{S}\,x)]$, i.e. $n > f(\mathsf{S}\,x)$. Since f majorises $\overline{H_I}$ and the goal is $\bot$, we can assume l with $\#l$, $|l| = \mathsf{S}\,x$ and $\forall y \in l.\overline{H_I}y \wedge y \leq f(\mathsf{S}\,x)$. Let $m := \mathsf{max}(\mathsf{map}E_I\,l)$. We have that $m \geq x$, since $|\mathsf{map}E_I l| = |l| > x$ and $\#(\mathsf{map}E_I l)$, and that $m = E_I m_0$ for some $m_0 \in l$ and therefore $m_0 \leq f(\mathsf{S}\,x)$ and $\overline{H_I}m_0$ by the properties of l. We now prove $H_I m_0$ to obtain a contradiction. We have $n > f(\mathsf{S}\,x) \geq m_0$ and $E_I n = x \leq m = E_I m_0$. By the injectivity of E_I, $E_I n = E_I m_0$ implies $n = m_0$ (a contradiction), i.e. $E_I n < E_I m_0$ as required. ◀

Theorem 80. *H_I is a hypersimple predicate, and in particular $H_{\mathcal{W}'}$ is.*

Proof. $\mathcal{E}H_I$ follows by (6) of Fact 6. ◀

We have not yet proved H_I to be undecidable. A proof that H_I is simple seems to require MP. We instead constructively prove $\mathcal{D}H_I{\to}\mathcal{D}I$ and thus H_I undecidable for undecidable I:

To do so, we define $\mathcal{B}\colon\mathbb{N}{\to}\mathbb{T}$ describing a certain kind of guardedness with guards in $\overline{H_I}$:

$$\mathcal{B}n := \Sigma x.E_I x \geq n \wedge \overline{H_I}x$$

Lemma 81. $\forall n x.\Sigma x' \geq x.E_I x' > n$.

Proof. By the injectivity of E_I using Lemma 29. ◀

Lemma 82. *Let H_I be decidable by some f. Then, $\forall n x'.(\mathcal{B}n) + (\Sigma x > x'.E_I x < n)$.*

Proof. Let H_I be decidable. Given n and x', we show

$$\forall n_0 \geq x'.E_I n_0 > n \to (\mathcal{B}n) + (\Sigma x > x'.E_I x < n)$$

by complete induction on $E_I n_0$:

Let $n_0 \geq x'$ with $E_I n_0 > n$. If $\overline{H_I}n_0$, $\mathcal{B}n$ holds and we are done. If $H_I n_0$ holds, we have $\exists n_1 > n_0.\ E_I n_1 < E_I n_0$. Applying $\mu_{\mathbb{N}}$ gives the stronger assumption $\Sigma n_1 > n_0.\ E_I n_1 < E_I n_0$. Then, induction for $E_I n_1$ implies the claim.

Finally, Lemma 81 yields $n_0 \geq x'$ as assumed in the claim. ◀

Lemma 83. *Let H_I be decidable by some f. Then $\forall n.\mathcal{B}n$.*

Proof. Let H_I be decidable. Given n, we first show

$$\forall d.(\mathcal{B}n) + (\Sigma L.\#L \wedge |L| = d \wedge \forall x \in L.E_I x \leq n)$$

by induction on d:

The base case is immediate by picking $L := []$. In the step case, the inductive hypothesis either yields $\mathcal{B}n$ which shows the claim or a duplicate-free list L with $|L| = d$ and $\forall x \in L.E_I x \leq n$ and an element $x > \mathsf{max}\,L$. Now, Lemma 82 for $x' := \mathsf{max}\,L$ yields again either $\mathcal{B}n$ or an element $x > \mathsf{max}\,L$ with $E_I x \leq n$. Then, $(x :: L)$ is a list as required which.

For $d := 2 + n$ however, $\Sigma L.\#L \land |L| = 2 + n \land \forall x \in L.E_I x \leq n$ is contradictory: By the injectivity of E_I, $\mathsf{map}\, E_I\, L$ is duplicate-free, $|\mathsf{map}\, E_I\, L| = 2 + n > |[0, \dots, n]$ but $\forall x \in \mathsf{map}\, E_I\, L.x \in [0, \dots, n]$. Lemma 29 implies the contradiction. Hence, we have $\mathcal{B}n$. ◀

Lemma 84. $(\forall n.\mathcal{B}n) \to \mathcal{D}I$.

Proof. $\forall n.\mathcal{B}n$ yields $\forall n.\Sigma x.E_I x \geq n \land \forall x_0 > x.E_I x_0 > E_I x$ which implies $H : \forall n.\Sigma x.\forall x_0 > x.E_I x_0 > n$.

Then, $\lambda n.n \in_{\mathbb{B}} \mathsf{map}\, E_I\, [0, \dots, \pi_1(Hn)]$ decides I. ◀

Corollary 85. $\mathcal{D}H_I \to \mathcal{D}I$.

Proof. Directly by Lemma 83 and Lemma 84. ◀

Corollary 86. *H_I is undecidable, and in particular $H_{\mathcal{W}'}$ is.*

It seems that this fully constructive proof of $\mathcal{D}H_I \to \mathcal{D}I$ can be turned into an again fully constructive Turing reduction from I to H_I, provided a reasonable notion of synthetic Turing reductions: Instead of applying $\mu_{\mathbb{N}}$ to the proof of $H_I n_0$ in Lemma 82, partial functions (that must be crucially present in Turing reductions) would allow to find $n_1 > n_0$ with $E_I n_1 < E_I n_0$ via an unbounded search.

The above results settle Post's problem for $\preceq_{\mathsf{tt}}$:

Theorem 87. *There exists an enumerable, undecidable and* tt*-incomplete predicate.*

Proof. By Fact 76, Corollary 86 and Theorem 80. ◀

13 Discussion

This paper provides formalised, mechanised, constructive, synthetic proofs of major well-known results in the area of reducibility theory. The synthetic perspective tremendously helped in all three other aspects: Synthetic proofs are easier to formalise, easier to mechanise, and easier to constructivise since it clears the view by avoiding a hard to handle concrete model of computation.

We would argue that a synthetic formalisation is an almost necessary pre-requisite for a mechanised proof of advanced computability-theoretic results: working in a model of computation in the phase of mechanisation where details are still unclear seems too tedious, since defining and verifying programs in a model of computation is time-consuming and should be delayed as much as possible. The concretisation of all synthetically developed results to a model of computation in a second step is then routine. Even in the mechanised proofs, no principal obstacles should occur. For Coq specifically, the automatic extraction of functions to a λ-calculus by Forster and Kunze [11] would simplify the task further.

Choosing CIC rather than other systems is crucial for projects in constructive reverse mathematics concerning synthetic computability theory: the textbook proofs this paper is based on are often heavily classical, not only in the regard that different, constructively non-equivalent definitions of infinity are used interchangeably. It was thus conceivable that some results would depend on a classical logical axiom like LEM or the limited principle of omniscience LPO. Due to the separated universe of propositions CIC features, CT $\land$ LEM seems to be consistent, which is however inconsistent in e.g. HoTT [6]. However, in the end all our results do not require any classical assumptions. This means that the results can be transported to other type theories. Compatibility with classical logic is however still desirable both for possible investigations of classical synthetic computability theory as well as for purposes of constructive reverse mathematics.

As follow-up work to the present paper, Forster, Kunze, and Lauermann [12] prove that nonrandom numbers, defined via Kolmogorov complexity, form a simple predicate. Crucially, they started by working in a classical setting under the assumption of LEM, and gradually constructivise proofs until not even MP is required.

For future work, two questions seem most prominent: First, we would like to analyse whether our assumptions are minimal. It would be interesting to see whether MP is necessary to prove that hypersimple predicates are simple and undecidable. Secondly, Turing reducibility is the next natural step. The first author and Kirst propose a synthetic definition of Turing reducibility in CIC in [9], based on an idea by Bauer [2]. Several open questions remain, most prominently how to construct a universal oracle functional. We would like to develop synthetic versions of the Kleene-Post theorem [15], stating that there are incomparable Turing-reducibility degrees, Post's theorem [25], connecting Turing reducibility via the Turing jump to the arithmetical hierarchy, and directly subsequent to the present paper the Friedberg-Mučnik theorem [13, 20], settling Post's problem by proving that there exists an enumerable, undecidable, but Turing-reducibility incomplete predicate.

The Coq development accompanying the paper is sizeable due to the number of results covered, but due to the synthetic approach it is still concise. It is relatively elementary: No advanced dependent types are needed, we do not rely on complex hand-written automation.

The preparations regarding the axioms for synthetic computability, finiteness, Pigeonhole principles, and infinity, take 900 lines of code (LOC). Defining $\preceq_{\mathsf{m}}$, $\preceq_{1}$, and $\preceq_{\mathsf{tt}}$ and proving related facts needs 700 LOC. General results regarding simple and hypersimple predicates take 100 and 230 LOC respectively. The construction of S and S^* needs 700 LOC, of H_I only 250. The `std++` library [30] is used for several convenient auxiliary functions regarding lists missing in Coq's standard library.

References

1 Andrej Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006.

2 Andrej Bauer. Synthetic mathematics with an excursion into computability theory. University of Wisconsin Logic seminar, 2021. URL: `http://math.andrej.com/asset/data/madison-synthetic-computability-talk.pdf`.

3 Douglas Bridges and Fred Richman. *Varieties of constructive mathematics*, volume 97. Cambridge University Press, 1987.

4 Nigel Cutland. *Computability: An introduction to recursive function theory*. Cambridge university press, 1980.

5 James C. E. Dekker. A theorem on hypersimple sets. *Proceedings of the American Mathematical Society*, 5:791–796, 1954.

6 Yannick Forster. Church's Thesis and Related Axioms in Coq's Type Theory. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CSL.2021.21`.

7 Yannick Forster. *Computability in Constructive Type Theory*. PhD thesis, Saarland University, 2021. `doi:10.22028/D291-35758`.

8 Yannick Forster. Parametric Church's Thesis: Synthetic computability without choice. In Sergei Artemov and Anil Nerode, editors, *Logical Foundations of Computer Science*, pages 70–89, Cham, 2022. Springer International Publishing. `doi:10.1007/978-3-030-93100-1_6`.

9 Yannick Forster and Dominik Kirst. Synthetic Turing reducibility in constructive type theory. 28th International Conference on Types for Proofs and Programs (TYPES 2022), 2022.

10 Yannick Forster, Dominik Kirst, and Gert Smolka. On synthetic undecidability in Coq, with an application to the Entscheidungsproblem. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 38–51, 2019.

11 Yannick Forster and Fabian Kunze. A Certifying Extraction with Time Bounds from Coq to Call-By-Value Lambda Calculus. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITP.2019.17.

12 Yannick Forster, Fabian Kunze, and Nils Lauermann. Synthetic Kolmogorov Complexity in Coq. working paper or preprint, March 2022. URL: https://hal.inria.fr/hal-03596267.

13 Richard M Friedberg and Hartley Rogers Jr. Reducibility and completeness for sets of integers. *Mathematical Logic Quarterly*, 5(7-13):117–125, 1959. doi:10.1002/malq.19590050703.

14 Felix Jahn. *Synthetic One-One, Many-One, and Truth-Table Reducibility in Coq*. Bachelor's thesis, Saarland University, 2020.

15 Steven C. Kleene and Emil L. Post. The upper semi-lattice of degrees of recursive unsolvability. *The Annals of Mathematics*, 59(3):379, May 1954. doi:10.2307/1969708.

16 Georg Kreisel. Mathematical logic. *Lectures in modern mathematics*, 3:95–195, 1965. doi:10.2307/2315573.

17 Bassel Mannaa and Thierry Coquand. The independence of markov's principle in type theory. *Logical Methods in Computer Science*, 13, 2017.

18 Andrei Andreevich Markov. The theory of algorithms. *Trudy Matematicheskogo Instituta Imeni VA Steklova*, 42:3–375, 1954.

19 The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pages 367–381, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3372885.3373824.

20 Albert Abramovich Munik. On strong and weak reducibility of algorithmic problems. *Sibirskii Matematicheskii Zhurnal*, 4(6):1328–1341, 1963.

21 Piergiorgio Odifreddi. *Classical recursion theory: The theory of functions and sets of natural numbers*. Elsevier, 1992.

22 Christine Paulin-Mohring. Inductive definitions in the system Coq rules and properties. In *International Conference on Typed Lambda Calculi and Applications*, pages 328–345. Springer, 1993.

23 Pierre-Marie Pédrot and Nicolas Tabareau. Failure is not an option. In *European Symposium on Programming*, pages 245–271. Springer, 2018.

24 Emil L Post. Recursively enumerable sets of positive integers and their decision problems. *bulletin of the American Mathematical Society*, 50(5):284–316, 1944.

25 Emil L. Post. Degrees of recursive unsolvability – preliminary report. In *Bulletin of the American Mathematical Society*, volume 54, pages 641–642. American Mathematical Society (AMS), 1948.

26 Fred Richman. Church's thesis without tears. *The Journal of symbolic logic*, 48(3):797–803, 1983.

27 Hartley Rogers. *Theory of recursive functions and effective computability*. CUMINCAD, 1987.

28 Robert I Soare. *Recursively enumerable sets and degrees: A study of computable functions and computably generated sets*. Springer Science & Business Media, 1999.

29 The Coq Development Team. The Coq Proof Assistant, version 8.11.0. https://doi.org/10.5281/zenodo.3744225, January 2020. doi:10.5281/zenodo.3744225.

30 The Coq std++ Team. An extended "standard library" for Coq. https://gitlab.mpi-sws.org/iris/stdpp, 2020.

31 Anne Sjerp Troelstra and Dirk van Dalen. Constructivism in mathematics. vol. i, volume 121 of. *Studies in Logic and the Foundations of Mathematics*, 26, 1988.

A Cylindrification proofs

We characterise many-one reducibility and truth-table reducibility in terms of one-one reducibility.

For the former, the proofs by Rogers [27, §7.6 Th. VIII] work via so-called cylindrification and we generalise them slightly to apply to base types other than $\mathbb{N}$. In our setting, a cylindrification of a predicate $p : X \to \mathbb{P}$ is $p \times Z$ for an inhabited type Z, and predicates of the form $p \times Z$ in general are called cylinders.

We fix two predicates $p: X \to \mathbb{P}$, $q: Y \to \mathbb{P}$ and $z: Z$.

Lemma 88. *Let $f: Y \times Z \to Z$ be an injection. Then $q \preceq_{\mathsf{m}} p \leftrightarrow q \times Z \preceq_1 p \times Z$.*

Proof. We first prove:

1. $p \preceq_1 p \times Z$ given $z : Z$.
2. $p \times Z \preceq_{\mathsf{m}} p$.
3. If $q \preceq_{\mathsf{m}} p \times Z$ then $q \preceq_1 p \times Z$, provided an injection $g: Y \times X \to Z$.

The only interesting proof is (3): Let f reduce q to $p \times Z$. Then $\lambda y.(\pi_1(fy), g(y, \pi_1(fy)))$ is proves $q \preceq_1 p \times Z$. It is injective since g is injective and correct since f is correct.

Now, the direction from left to right follows from (1), (2), and (3). The converse direction from (1) and (2). ◀

Corollary 89. *Let $f: Y \to Z$ be injective. Then $p \preceq_1 p \times Z$ and for q s.t. $q \equiv_{\mathsf{m}} p$ and $p \preceq_1 q$, $q \preceq_1 p \times Z$.*

To characterise truth-table reducibility in terms of one-one reduciblity, we introduce so-called truth-table cylinders, following Rogers [27, §8.4 Th. IX]. Given a predicate $p: X \to \mathbb{P}$, we define the predicate $p^{\mathsf{tt}}: \mathbb{L}X \times \mathsf{truthtable} \to \mathbb{P}$:

$$p^{\mathsf{tt}} := \lambda z.\ \forall l.\ \mathsf{Forall}_2\ (\lambda x b.\ px \leftrightarrow b = \mathsf{true})(\pi_1 z)\ l \to l \vdash \pi_2 z$$

The characterisation seems to be inherently non-constructive, we thus assume p to be stable.

Lemma 90. *Let $p: X \to \mathbb{P}$, $q: Y \to \mathbb{P}$, $x_0: X, y_0: Y$, and $g: \mathbb{L}X \times \mathsf{truthtable} \to Y$ be an injection. If p is stable, then $p \preceq_{\mathsf{tt}} q \leftrightarrow p^{\mathsf{tt}} \preceq_1 q^{\mathsf{tt}}$.*

Proof. We loosely follow the proof by Rogers [27, §8.4 Th. IX] and prove the following: (1) If p is stable then $p \preceq_1 p^{\mathsf{tt}}$. (2) $p^{\mathsf{tt}} \preceq_{\mathsf{tt}} p$. (3) Every reduction $p \preceq_{\mathsf{tt}} q$ can be given via an injective reduction function, provided an injection $f: X \to Y$ exists. (4) If p is stable, $f: X \to Y$ injective then $p \preceq_{\mathsf{tt}} q \to p \preceq_1 q^{\mathsf{tt}}$.

(1) and (2) are straightforward. For (3) assume $p \preceq_{\mathsf{tt}} q$ via a reduction function g and construct $g'x := (fx :: \pi_1(gx), \lambda\, b :: L.\ \pi_2(gx)L)$. g' is injective since f is injective. For (4), let $p \preceq_{\mathsf{tt}} q$ via an injection f by (3). Then f 1-reduces p to q^{tt}. The claim from left to right follows using (1), (2), and (4), the direction from right to left needs (1) and (4). ◀

Corollary 91. *If $p, q: \mathbb{N} \to \mathbb{P}$ and p stable, $p \preceq_{\mathsf{tt}} q \leftrightarrow p^{\mathsf{tt}} \preceq_1 q^{\mathsf{tt}}$.*

Quantitative Hennessy-Milner Theorems via Notions of Density

Jonas Forster
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Sergey Goncharov
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Dirk Hofmann
CIDMA, University of Aveiro, Portugal

Pedro Nora
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Lutz Schröder
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Paul Wild
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

The classical *Hennessy-Milner theorem* is an important tool in the analysis of concurrent processes; it guarantees that any two non-bisimilar states in finitely branching labelled transition systems can be distinguished by a modal formula. Numerous variants of this theorem have since been established for a wide range of logics and system types, including quantitative versions where lower bounds on behavioural distance (e.g. in weighted, metric, or probabilistic transition systems) are witnessed by quantitative modal formulas. Both the qualitative and the quantitative versions have been accommodated within the framework of *coalgebraic logic*, with distances taking values in quantales, subject to certain restrictions, such as being so-called *value quantales*. While previous quantitative coalgebraic Hennessy-Milner theorems apply only to liftings of set functors to (pseudo)metric spaces, in the present work we provide a quantitative coalgebraic Hennessy-Milner theorem that applies more widely to functors native to metric spaces; notably, we thus cover, for the first time, the well-known Hennessy-Milner theorem for continuous probabilistic transition systems, where transitions are given by Borel measures on metric spaces, as an instance of such a general result. In the process, we also relax the restrictions imposed on the quantale, and additionally parametrize the technical account over notions of *closure* and, hence, *density*, providing associated variants of the Stone-Weierstraß theorem; this allows us to cover, for instance, behavioural ultrametrics.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Concurrency

Keywords and phrases Behavioural distances, coalgebra, characteristic modal logics, density, Hennessy-Milner theorems, quantale-enriched categories, Stone-Weierstraß theorems

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.22

Related Version *Full Version with Proofs*: `https://arxiv.org/abs/2207.09187`

Funding *Jonas Forster*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 434050016.
Dirk Hofmann: Funded by The Center for Research and Development in Mathematics and Applications (CIDMA) through the Portuguese Foundation for Science and Technology (FCT) – project numbers UIDB/04106/2020 and UIDP/04106/2020.
Pedro Nora: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 259234802.
Lutz Schröder: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 434050016.

1 Introduction

Modal logic in general is an established tool in the analysis of concurrent systems. One of its uses is as a means to distinguish non-equivalent states; for instance, the classical Hennessy-Milner theorem [18] guarantees that any two non-bisimilar states in finitely branching labelled transition systems can be distinguished by a formula in a modal logic naturally associated to labelled transition systems. Similar theorems have subsequently proliferated, having been established, for instance, for probabilistic transition systems [24], neighbourhood structures [17], and open bisimilarity in the π-calculus [3]. As a recent example application, the counterproof for unlinkability in the ICAO 9303 standard for e-passports [13] is based on providing a distinguishing modal formula in an intuitionistic modal logic.

For systems featuring quantitative data, such as probabilistic or weighted systems or metric transition systems, *behavioural distance* provides a more fine-grained measure of agreement between systems than two-valued bisimilarity (e.g. [15, 33, 32, 10]). In analogy to the classical Hennessy-Milner theorem, behavioural distances can often be characterized by quantitative modal logics, in the sense that the behavioural distance of any two states can be approximated by the difference in value of quantitative modal formulae on these states (that is, for states with distance $> r$ one can find a quantitative modal formula on which the states disagree by at least r). Such theorems, to which we refer as *quantitative Hennessy-Milner theorems*, have been proved, e.g., for probabilistic transition systems [33, 32] and for metric transition systems [10].

Universal coalgebra [29] serves as a generic framework for concurrent systems, based on the key abstraction of encapsulating the system type in a functor, whose coalgebras then correspond to the systems of interest. Both two-valued and quantitative Hennessy-Milner theorems have been established at the level of generality offered by *coalgebraic modal logic*. The two-valued coalgebraic Hennessy-Milner theorem [27, 30] covers all coalgebraic system types, under the assumption of having a *separating* set of modalities; instances include the mentioned Hennessy-Milner theorems for probabilistic systems [24] and neighbourhood structures [17]. Various quantitative coalgebraic Hennessy-Milner theorems have been established fairly recently [23, 34, 35, 22]. These existing theorems are tied to considering liftings of set functors to metric spaces (or in fact more general topological categories [22]); our contribution in the present work is to complement these theorems by a result that instead applies to unrestricted functors on metric spaces (we give a more detailed comparison in the related work section). In particular, our result covers, for the first time, the original expressivity result for probabilistic modal logic on continuous probabilistic transition systems (where "continuous" refers to the structure of the state space) [33, 32] as an instance of a general coalgebraic result. We work not only in coalgebraic generality but also parametrize the development over the choice of a *quantale* $\mathcal{V}$, in which distances and truth values are taken; this covers the case of standard bounded real-valued distances by taking $\mathcal{V}$ to be the unit interval, and the classical two-valued case by taking $\mathcal{V}$ to be the set of Boolean truth values. Previous work on quantalic distances [35] needed to restrict to so-called value quantales [14]; we relax this assumption, covering, for instance, all finite quantales (such as the four-valued quantale used in some paraconsistent logics; see, e.g., [28]), and the square of the unit interval.

Technically, our results are additionally parametrized over the choice of *closure* operators on sets of $\mathcal{V}$-valued predicates, which induce a notion of *density*. The notion of density is the key ingredient that lets our results apply beyond discrete state spaces (e.g. to the mentioned

continuous probabilistic transition systems); by varying the notion of closure, we cover, for instance, both standard metric spaces and ultrametric spaces (which in turn are induced by different quantale structures on the unit interval).

Proofs are mostly omitted and can be found in arXiv version of the paper.

Related work. As indicated above, quantitative coalgebraic Hennessy-Milner theorems exist in previous work [23, 34, 35, 22], from which our present work is distinguished in that it applies to functors that live natively on metric spaces (such as tight Borel distributions) rather than only to liftings of set functors (such as finitely supported distributions). We detach the technical development from both lax extensions [34, 35] and fixpoint induction [23, 34, 35], which work only for monotone modalities; we thereby cover also systems requiring non-monotone modalities, such as weighted transition systems with negative weights. A recent general framework for Hennessy-Milner theorems based on Galois connections between real-valued predicates and (pseudo)metrics is aimed primarily at generality over a linear-time/branching-time spectrum [7].

The framework of codensity liftings developed by Komorida et al. [21, 22] works at a very high level of generality, and in fact applies to topological categories (or $\mathsf{CLat}_\sqcap$-fibrations, in the terminology of *op. cit.*) beyond metric spaces, such as uniform spaces. Our present framework is on the one hand more general in that we do not restrict to functors lifted from the category of sets, but on the other hand less general in that we cover only (quantalic) behavioural distances. In terms of the main technical result, we provide a coalgebraic quantitative Hennessy-Milner theorem that is stated in fairly simple terms, and can be instantiated to concrete logics and systems by just verifying a few fairly straightforward conditions that concern only the functor and the modalities. In particular, we have no conditions requiring that certain sets of formula evaluations on a given coalgebra are *approximating* (cf. [22, Theorems IV.5, IV.7]); instead, we prove similar properties as *lemmas* along the way, using the key notions of closure and density. In fact, one of the conditions of our Hennessy-Milner theorem can be seen as a form of Stone-Weierstraß property, and in particular concerns density of sets of functions closed under suitable propositional combinations; this property depends only on the quantale, not on the functor or the modalities, and we give general Stone-Weierstraß-type theorems for several classes of quantales.

2 Preliminaries

Basic familiarity with category theory will be assumed [1, 5]. More specifically, we make extensive use of topological categories (e.g. [1]). We recall some central notions for convenience.

Universal coalgebra. For an endofunctor $\mathsf{F}\colon \mathcal{C} \to \mathcal{C}$ on a category $\mathcal{C}$, an **F-coalgebra** (X, α) consists of an object X of $\mathcal{C}$, thought of as an object of *states*, and a morphism $\alpha\colon X \to \mathsf{F}X$, thought of as assigning structured collections (sets, distributions, etc.) of *successors* to states. A **coalgebra morphism** from (X, α) to (Y, β) is a morphism $f\colon X \to Y$ such that $\beta \circ f = \mathsf{F}f \circ \alpha$. A **concrete category** over Set comes equipped with a faithful functor $|-|\colon \mathcal{C} \to \mathsf{Set}$, which allows us to speak about individual *states*, as elements of $|X|$. Given a coalgebra (X, α) and states $x, y \in |X|$, we say that x and y are **behaviourally equivalent** if there are a coalgebra (Z, γ) and a coalgebra morphism $f\colon X \to Z$ such that $|f|(x) = |f|(y)$. (For brevity, we restrict the treatment of both behavioural equivalence and behavioural distances to states in the same coalgebra; in all our examples the extension to states in different coalgebras can be accommodated by taking coproducts.) The notion

of behavioural equivalence is strictly two-valued, meaning that different states are either behaviourally equivalent or not. The downside of this notion is thus that in systems dealing with quantitative information, any slight change can render two states behaviourally distinct, even though they may be virtually indistinguishable in any practical context. The rest of this paper is concerned with quantifying the degree to which states differ from each other, as well as with logics to witness these degrees.

▶ **Example 1.**

1. Labelled transition systems w.r.t. a set A of actions are coalgebras for the Set-functor $\mathcal{P}(A \times -)$. Behavioural equivalence coincides with the classical notion of (strong) bisimilarity.
2. We write $\diamond$ for the four-element diamond-shaped lattice, i.e. $\diamond = \{\bot, \mathsf{N}, \mathsf{B}, \top\}$, ordered by $\bot < \mathsf{N} < \top$, $\bot < \mathsf{B} < \top$. Let $\mathcal{B}$ be the $\diamond$-*valued powerset functor*, which sends a set X to the set of all maps $q\colon X \to \diamond$, and a map $f\colon X \to Y$ to the map $\mathcal{B}f$ given by $\mathcal{B}f(q)(y) = \bigvee_{f(x)=y} q(x)$. A map $q\colon X \to \diamond$ can be seen as a $\diamond$-valued fuzzy subset of X, which for every element x tells us either that x is in the set ($q(x) = \top$), or that x is not in the set ($q(x) = \bot$), or that there is evidence both that x is in the set and that x is not in the set ($q(x) = \mathsf{B}$), or that nothing is known ($q(x) = \mathsf{N}$). $\mathcal{B}$-coalgebras have been used in a Kripke-style semantics of paraconsistent modal logics [28] (our $\bot$, $\top$, N, B respectively correspond to f, t, $\bot$, $\top$, and $\leq$ to $\leq_t$ in op. cit.).
3. We denote by $\mathcal{D}$ the functor that maps a set X to the set of finitely supported probability distributions on X. Coalgebras for the functor $FX = (1 + \mathcal{D}X)^A$, for a finite set A of actions, are probabilistic transition systems [24, 11]. In this context, behavioural equivalence instantiates to probabilistic bisimilarity [20].
4. We consider weighted transition systems with possibly negative weights (e.g. [8]): Let $\mathcal{W}$ be the functor on 1-bounded metric spaces that maps every set X to the set of finite $[-1,1]$-weighted sets over X. That is, the elements of $\mathcal{W}X$ are functions $t\colon X \to [-1,1]$ such that $t(x) = 0$ for all but finitely many x, and $\sum_{x\in A} t(x) \in [-1,1]$ for all $A \subseteq X$. On morphisms, $\mathcal{W}$ acts by summing over preimages, that is, $\mathcal{W}g(t)(y) = \sum_{x\in g^{-1}(y)} t(x)$ for $g\colon X \to Y$, $t \in \mathcal{W}X$.
   The distance of $s, t \in \mathcal{W}X$ is given by $d(s,t) = \frac{1}{2}\bigvee_f \sum_{x\in X} s(x)f(x) - t(x)f(x)$ where the join ranges over all nonexpansive functions $f\colon X \to [0,1]$. Then $(\mathcal{W}-)^A$ coalgebras are $([-1,1], +, 0)$-weighted A-labelled transition systems. Behavioural equivalence instantiates to weighted bisimilarity [20].
5. Consider the following variation of the Kantorovich functor $\mathcal{K}$ [32, 2]. We say that a probability measure μ on the Borel σ-algebra of a (pseudo)metric space (X,d) is *tight* if for every $\epsilon > 0$, there is a totally bounded subset $Y \subseteq X$ such that $\mu(X \setminus Y) < \epsilon$. The Kantorovich functor $\mathcal{K}$ maps a (pseudo)metric space (X,d) to the set of tight probability measures on (X,d), equipped with the Kantorovich metric, defined as $d_{\mathcal{K}X}(\mu,\nu) = \sup_f \left\{\int f \,\mathrm{d}\mu - \int f \,\mathrm{d}\nu\right\}$ for $\mu, \nu \in \mathcal{K}X$, where again f ranges over all nonexpansive maps $X \to [0,1]$. On morphisms, $\mathcal{K}$ acts by taking image measures, i.e. for $f\colon X \to Y$ we have $\mathcal{K}f(\mu)(Y') = \mu(f^{-1}(Y'))$ for Borel sets $Y' \subseteq Y$. Given a finite set A of actions, A-labelled continuous probabilistic transition systems are $\mathcal{K}(1 + -)^A$-coalgebras [32, 33] (so the term *continuous* applies to the state space, not the system evolution), and behavioural equivalence instantiates to probabilistic bisimilarity of continuous systems.

Consider the probabilistic transition systems depicted in Figure 1. If $\epsilon > 0$, then the root states are not probabilistically bisimilar, as they have different probabilities of reaching a deadlock state. Still one would like to say that their difference in behaviour is small if ϵ is small. We will review formal definitions of such concepts in Section 4.

Figure 1 Probabilistic transition systems with behaviourally inequivalent root states.

Topological Categories. Let $\mathsf{F}\colon \mathcal{C} \to \mathcal{X}$ be a faithful functor. Given $\mathcal{C}$-objects C, D, we say that an $\mathcal{X}$-morphism $f\colon \mathsf{F}C \to \mathsf{F}D$ **is a morphism** $C \to D$ if $f = \mathsf{F}\bar{f}$ for some (necessarily unique) $\bar{f}\colon C \to D$. A cone $(f_i\colon C \to C_i)_{i\in I}$ in $\mathcal{C}$ (with I a class) is **initial** if the following hods: whenever, given a $\mathcal{C}$-object B and $g\colon \mathsf{F}B \to \mathsf{F}C$, $f_i \circ g$ is a morphism $B \to C_i$ for all $i \in I$, then g is a morphism $B \to C$. A morphism is **initial** if the corresponding singleton cone is initial. A functor $\mathsf{F}\colon \mathcal{C} \to \mathcal{C}$ **preserves initial morphisms** if $\mathsf{F}f$ is initial whenever f is initial. Now, an (F-)**structured morphism** is a pair (f, C) consisting of a $\mathcal{C}$-object C and a morphism $f\colon X \to \mathsf{F}C$, typically written just as $f\colon X \to \mathsf{F}C$, and an ($\mathsf{F}$-)**structured cone** is a family $\mathcal{S} = (f_i\colon X \to \mathsf{F}C_i)_{i\in I}$ of structured morphisms. An **initial lift** of $\mathcal{S}$ is an initial cone $(\bar{f}_i\colon C \to C_i)$ such that $\mathsf{F}\bar{f}_i = f_i$ for all i. The functor F is **topological** if every F-structured cone has an initial lift; we then also say that $\mathcal{C}$ is **topological over** $\mathcal{X}$, leaving F implicit. (We note that we have assumed faithfulness of F only for ease of presentation, and in fact one can show under a prima facie more general definition that all topological functors are faithful [1, Theorem 21.3].) Typical examples of topological categories are topological spaces, pseudometric spaces, and preordered sets (while subcategories of such categories that are determined by separation conditions, e.g. Hausdorff spaces, metric spaces, or ordered sets, are typically only **monotopological**, in the sense that only monic cones are guaranteed to have initial lifts [1]). For instance, the initial lift of a structured cone $(f_i\colon X \to (Y_i, \leq_i))_{i\in I}$ of preordered sets is the preorder $\leq$ on X given by $x \leq z$ iff $f_i(x) \leq_i f_i(z)$ for all i.

Recent work on codensity liftings [21, 22] employs $\mathsf{CLat}_\sqcap$*-fibrations*, which are easily seen to be essentially equivalent to topological functors (more precisely, to *amnestic* topological functors [1]). Topological functors come with a rich and well-developed theory, on which we will draw to some degree in our technical treatment. Basic facts to note are that topological functors lift limits, so topological categories are complete if their underlying category is complete, and that topological functors are also cotopological, so the same holds for colimits.

3 Quantales and Quantale-Enriched Categories

A central notion of our development are quantales, which will serve as objects of both truth values and distance values, subsuming in particular the two-valued and the real-valued case. A **quantale** $(\mathcal{V}, \bigvee, \otimes, k)$, more precisely a commutative and unital quantale, is a complete lattice $\mathcal{V}$ that carries the structure of a commutative monoid $(\mathcal{V}, \otimes, k)$, with $\otimes$ called **tensor** and k called **unit**, such that for every $u \in \mathcal{V}$, the map $u \otimes -\colon \mathcal{V} \to \mathcal{V}$ preserves suprema, which entails that every $u \otimes -\colon \mathcal{V} \to \mathcal{V}$ has a right adjoint $\hom(u, -)\colon \mathcal{V} \to \mathcal{V}$, characterized by the property $u \otimes v \leq w \iff v \leq \hom(u, w)$. We denote by $\top$ and $\bot$ the greatest and the least element of a quantale respectively. A quantale is **non-trivial** if $\bot \neq \top$, and **integral** if $\top = k$.

▶ **Example 2.**

1. Every frame (i.e. a complete lattice in which binary meets distribute over infinite joins) is a quantale with $\otimes = \wedge$ and $k = \top$. In particular, every finite distributive lattice is a quantale, prominently 2, the two-element lattice $\{\bot, \top\}$. In this case, the hom operation is implication in the usual sense.

2. Every left continuous t-norm [4] defines a quantale on the unit interval equipped with its natural order.
3. The previous clause further specializes as follows (up to isomorphism):
   a. The quantale $[0,\infty]_+ = ([0,\infty], \inf, +, 0)$ of non-negative real numbers with infinity, ordered by the greater or equal relation, and with tensor given by addition.
   b. The quantale $[0,\infty]_{\max} = ([0,\infty], \inf, \max, 0)$ of non-negative real numbers with infinity, ordered by the greater or equal relation, and with tensor given by maximum.
   c. The quantale $[0,1]_\oplus = ([0,1], \inf, \oplus, 0)$ of the unit interval, ordered by the greater or equal order, and with tensor given by truncated addition. In this case as well as in $[0,\infty]_+$, the hom operation is truncated addition: $\hom(u,v) = \max(v-u, 0)$.

   (Note that in all these examples, the quantalic order is dual to the standard numeric order.)
4. Every commutative monoid $(M, \cdot, e)$ generates a quantale structure on $(\mathcal{P}M, \bigcup)$, the free quantale on M. The tensor $\otimes$ on $\mathcal{P}M$ is defined by $A \otimes B = \{a \cdot b \mid a \in A \text{ and } b \in B\}$, for all $A, B \subseteq M$. The unit of this multiplication is the set $\{e\}$.
5. For every quantale $\mathcal{V}$ and every partially ordered set X, the set of monotone maps $\mathsf{Pos}(X, \mathcal{V})$ ordered pointwise becomes a quantale with tensor defined pointwise. For instance, $\mathsf{Pos}(2, [0,1]_\oplus)$ with discrete 2 yields the quantale $[0,1]^2_\oplus$, and by replacing 2 with the two-element chain $0 \geq 1$ we obtain the quantale $\mathcal{I}([0,1]_\oplus)$ of non-empty closed subintervals of $[0,1]$ [35].

Category theory highlights preordered sets as 2-enriched categories. By replacing 2 with a quantale $\mathcal{V}$, we enrich the relevant preorders with a quantitative extent: A **$\mathcal{V}$-category** is pair (X, a) consisting of a set X and a map $a\colon X \times X \to \mathcal{V}$ that satisfies the inequalities $k \leq a(x,x)$ and $a(x,y) \otimes a(y,z) \leq a(x,z)$ for all $x, y, z \in X$. We think of a as providing a generalized notion of *similarity* (which under the reverse ordering becomes a notion of *distance*). The quantale $\mathcal{V}$ itself is canonically a $\mathcal{V}$-category $(\mathcal{V}, \hom)$, which we also denote by just $\mathcal{V}$. A **$\mathcal{V}$-functor** $f\colon (X,a) \to (Y,b)$ is a map $f\colon X \to Y$ such that $a(x,y) \leq b(f(x), f(y))$ for all $x, y \in X$. $\mathcal{V}$-categories and $\mathcal{V}$-functors form the category $\mathcal{V}$-Cat.

Every $\mathcal{V}$-category (X,a) carries a **natural order** defined by $x \leq y$ whenever $k \leq a(x,y)$, which induces a faithful functor $\mathcal{V}$-$\mathsf{Cat} \to \mathsf{Ord}$ into the category Ord of partially ordered sets.

A $\mathcal{V}$-category (X,a) is **symmetric** if $a(x,y) = a(y,x)$ for all $x, y \in X$, and **separated** if its natural order is antisymmetric. We denote by $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$ and $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym,sep}}$ the full subcategories of $\mathcal{V}$-Cat determined by the symmetric and the symmetric separated $\mathcal{V}$-categories, respectively. For real-valued $\mathcal{V}$, $\mathcal{V}$-Cat, $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$, and $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym,sep}}$ correspond to categories of hemimetric, pseudometric, and metric spaces, respectively. We will use $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$ as the main device to formalize examples and state our main results, although most of these results can be meaningfully reinterpreted for $\mathcal{V}$-Cat.

▶ **Example 3.**

1. The category 2-Cat is equivalent to the category Ord of **preordered sets** and monotone maps.
2. As noted by Lawvere [25], metric, ultrametric, and bounded metric spaces can be seen as quantale-enriched categories:
   a. The category $[0,\infty]_+$-$\mathsf{Cat}_{\mathsf{sym,sep}}$ is equivalent to the category Met of generalized **metric spaces** and non-expansive maps.
   b. The category $[0,\infty]_{\max}$-$\mathsf{Cat}_{\mathsf{sym,sep}}$ is equivalent to the category UMet of generalized **ultrametric spaces** and non-expansive maps.
   c. The category $[0,1]_\oplus$-$\mathsf{Cat}_{\mathsf{sym,sep}}$ is equivalent to the category BMet of **bounded-by-1 metric spaces** and non-expansive maps.

Table 1 $\mathcal{V}$-categorical notions in the qualitative and the quantitative setting. The prefix "pseudo" refers to absence of separatedness, and the prefix "hemi" additionally indicates absence of symmetry.

General $\mathcal{V}$	Qualitative ($\mathcal{V} = 2$)	Quantitative ($\mathcal{V} = [0,1]_\oplus$)
$\mathcal{V}$-category	preorder	bounded-by-1 hemimetric space
symmetric $\mathcal{V}$-category	equivalence	bounded-by-1 pseudometric space
$\mathcal{V}$-functor	monotone map	non-expansive map
initial $\mathcal{V}$-functor	order-reflecting monotone map	isometry
L-dense $\mathcal{V}$-functor	monotone map that is surjective up to the induced equivalence	non-expansive map with dense image
L-closure	closure under the induced equivalence	topological closure

3. Categories enriched in the powerset of the monoids underlying the quantales of the previous example can be thought of as spaces where a non-deterministic distance is assigned to each pair of points.
4. Categories enriched in the quantale $\mathcal{I}([0,1]_\oplus)$ can be thought of as spaces where a distance range is assigned to each pair of points.

The examples $\mathcal{V} = 2$ and $\mathcal{V} = [0,1]_\oplus$ are particularly instructive, as they represent the most established qualitative and quantitative aspects quantales aim to generalize. Table 1 provides some instances of generic quantale-based concepts (either introduced above or to be introduced presently) in these two cases, for further reference.

The forgetful functor $|-|\colon \mathcal{V}\text{-}\mathsf{Cat} \to \mathsf{Set}$ is topological (Section 2): The **initial lift** (X,a) of a structured cone $\mathcal{S} = (f_i\colon X \to |(X_i,a_i)|)_{i\in I}$, with a referred to as the **initial structure** w.r.t. $\mathcal{S}$, is given by $a(x,y) = \bigwedge_{i\in I} a_i(f_i(x),f_i(y))$ for $x,y \in X$. If all a_i are symmetric, then a is symmetric, and if all a_i are separated *and* the cone $\mathcal{S}$ is monic, then a is separated. Thus, $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ is topological over Set, while $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sep}}$ and $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym,sep}}$ are monotopological over Set (but not topological). It follows by general results [1] that all these categories are reflective in $\mathcal{V}\text{-}\mathsf{Cat}$. In particular, the reflector

$$(-)_q\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym,sep}}$$

quotients (X,a) by its natural preorder, which for symmetric (X,a) is an equivalence. The category $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ is also coreflective in $\mathcal{V}\text{-}\mathsf{Cat}$; the coreflector

$$(-)_s\colon \mathcal{V}\text{-}\mathsf{Cat} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$$

sends a $\mathcal{V}$-category (X,a) to its **symmetrization** $(X,a)_s = (X,a_s)$, where $a_s(x,y) = a(x,y) \wedge a(y,x)$ (keep in mind that in Example 2.3, the order is the dual of the numeric order). In particular, $\mathcal{V}_s$ is the set $\mathcal{V}$ equipped with the $\mathcal{V}$-category structure a given by $a(u,v) = \hom(u,v) \wedge \hom(v,u)$. For instance, for $\mathcal{V} = [0,1]_\oplus$ (Example 2.3), the $\mathcal{V}$-category structure of $\mathcal{V}_s$ is just the usual Euclidean distance $|u-v|$ on $[0,1]$.

For a set X and a $\mathcal{V}$-category (Y,b), we write $(Y,b)^X$ for the set of all maps $X \to Y$, equipped with the $\mathcal{V}$-category structure $[-,-]$ given by $[h,l] = \bigwedge_{x\in X} b(h(x),l(x))$ for $h,l\colon X \to Y$ (in the *numeric* ordering on real-valued quantales, this corresponds to the usual supremum metric on functions). For a $\mathcal{V}$-category (X,a), we moreover write $(Y,b)^{(X,a)}$ for $\mathcal{V}\text{-}\mathsf{Cat}((X,a),(Y,b))$, equipped with the $\mathcal{V}$-category structure inherited from $(Y,b)^X$. We note that we will often designate $\mathcal{V}$-categories just by single letters such as X; disambiguation

between spaces of maps and spaces of $\mathcal{V}$-functors should nevertheless always be clear from the context. For instance, if X is a $\mathcal{V}$-category, then $\mathcal{V}_s^X$ is the space of all $\mathcal{V}$-functors $X \to \mathcal{V}_s$, while $\mathcal{V}_s^{|X|}$ is the space of all maps $X \to \mathcal{V}_s$.

Quantale-enriched categories come equipped with a canonical closure operator. A $\mathcal{V}$-functor $m\colon M \to X$ is **L-dense** [19] if for all $\mathcal{V}$-functors $f, g\colon X \to \mathcal{V}$, $f \cdot m = g \cdot m$ implies $f = g$. The composite of L-dense $\mathcal{V}$-functors is again L-dense. For a subset A of X, the **L-closure** $\overline{A}$ of A in (X, a) is the largest $\mathcal{V}$-subcategory of (X, a) in which A is L-dense; this can be explicitly computed as

$$\overline{A} = \big\{x \in X \mid k \leq \textstyle\bigvee_{y \in A} a(x,y) \otimes a(y,x)\big\}.$$

A subset $A \subseteq X$ of a $\mathcal{V}$-category (X, a) is **L-closed** if $A = \overline{A}$, and **L-dense in** (X, a) if $\overline{A} = X$. A function $f\colon X \to Y$ between $\mathcal{V}$-categories (X, a) and (Y, b) is said to be **L-continuous** if $f[\overline{A}] \subseteq \overline{f[A]}$ for every $A \subseteq X$. It is easy to see that the notion of L-closure is so designed that for metric-like examples, it coincides with the topological closure w.r.t. the open-ball topology, and continuity in the sense defined above coincides with continuity w.r.t. this topology. For a preorder $(X, \leq)$, $A \subseteq X$ is L-closed iff it is closed under the induced equivalence. It is easy to check that every $\mathcal{V}$-functor is L-continuous, in generalization of the standard fact that every non-expansive map of metric spaces is continuous.

▶ **Proposition 4.** *For every $\mathcal{V}$-category X, $\mathcal{V}$-$\mathsf{Cat}(X, \mathcal{V})$ is L-closed in $\mathcal{V}^{|X|}$. For every symmetric $\mathcal{V}$-category X, $\mathcal{V}$-$\mathsf{Cat}(X, \mathcal{V}_s)$ is L-closed in $\mathcal{V}_s^{|X|}$.*

Maps that are continuous w.r.t. L-closure or other closures will be essential in Section 6.

4 Quantitative Coalgebraic Modal Logics

We proceed to introduce a variant of (quantitative) coalgebraic logic [27, 30, 9, 23, 34], which in particular follows the paradigm of interpreting modalities via predicate liftings, in this case of $\mathcal{V}$-valued predicates.

Given a cardinal κ, a classical κ-ary predicate lifting for a functor $\mathsf{F}\colon \mathsf{Set} \to \mathsf{Set}$ is a natural transformation $\lambda\colon \mathsf{Set}(-, 2^\kappa) \to \mathsf{Set}(\mathsf{F}-, 2)$, where we see a map $X \to 2^\kappa$ as κ many predicates on X; we switch freely between predicates and subsets. For example, the Kripke semantics of the modal logic K can be couched in terms of the diamond modality $\Diamond$, which we identify with the unary predicate lifting $\Diamond_X(A) = \{B \subseteq X \mid A \cap B \neq \varnothing\}$ for the powerset functor. This notion of predicate lifting naturally extends to $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$-functors:

▶ **Definition 5.** Given a cardinal κ, a κ-ary **predicate lifting** for a functor $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ is a natural transformation of type $\lambda\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}(-, \mathcal{V}_s^\kappa) \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}(\mathsf{F}-, \mathcal{V}_s)$.

A κ-ary predicate lifting thus lifts κ-many $\mathcal{V}$-functorial $\mathcal{V}$-valued predicates on X to a $\mathcal{V}$-functorial predicate on $\mathsf{F}X$.

▶ Remark 6 (Yoneda Lemma). By the Yoneda lemma, a κ-ary predicate lifting λ for a functor $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ is completely determined by its action on the identity map on $\mathcal{V}_s^\kappa$.

For the sake of brevity, we restrict the technical treatment to unary predicate liftings ($\kappa = 1$) henceforth; we do occasionally use non-unary liftings in examples, in particular nullary ones.

The syntax of quantitative coalgebraic modal logic can now be defined by the grammar

$$\phi ::= \top \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid u \otimes \phi \mid \hom_s(u, \phi) \mid \lambda(\phi) \qquad (u \in \mathcal{V}, \lambda \in \Lambda)$$

where Λ is a set of *modalities*, which we identify, by abuse of notation, with predicate liftings for a functor $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$. We view all other connectives as propositional

operators. Let $\mathcal{L}(\Lambda)$ be the set of modal formulas thus defined. The semantics is given by assigning to each formula $\phi \in \mathcal{L}(\Lambda)$ and each coalgebra $\alpha\colon X \to \mathsf{F}X$ the *interpretation* of ϕ over α, the $\mathcal{V}$-functor $[\![\phi]\!]_\alpha\colon X \to \mathcal{V}_s$ recursively defined as follows:

- for $\phi = \top$, we take $[\![\top]\!]_\alpha$ to be the $\mathcal{V}$-functor given by the constant map into $\top$;
- for an n-ary propositional operator p, we put $[\![p(\phi_1,\dots,\phi_n)]\!]_\alpha = p([\![\phi_1]\!]_\alpha,\dots,[\![\phi_n]\!]_\alpha)$, with p interpreted using the lattice structure of $\mathcal{V}$ and the $\mathcal{V}$-categorical structure $\hom_s$ of $\mathcal{V}_s$, respectively, on the right-hand side;
- for $\lambda \in \Lambda$, we put $[\![\lambda(\phi)]\!]_\alpha = \lambda([\![\phi]\!]_\alpha)\cdot\alpha$.

Given a coalgebra (X,α), we denote the set of all maps of the form $[\![\phi]\!]_\alpha$ by $[\![\mathcal{L}]\!]_\alpha$. Interpreting $\top$ as $\top$ and not as k is essential for non-integral quantales, for which the constant map with value k fails to be a $\mathcal{V}$-functor.

▶ **Example 7.** For a first example instance of the generic logic introduced above (more examples are seen in Section 8), recall the functor $\mathcal{K}$ from Example 1(5), which assigns to a pseudometric space X the space of tight probability measures on X, and put $\mathsf{F} = \mathcal{K}(1+-)^A$ for a set A of actions; then, F-coalgebras are pseudometric probabilistic labelled transition systems [33]. We have the expectation predicate lifting $\mathbb{E}$ for $\mathcal{K}$, given by

$$\mathbb{E}_X(f)(\mu) = \textstyle\int_X f(x)\,d\mu(x)$$

for $\mu \in \mathcal{K}X$ and non-expansive $f\colon X \to [0,1]$. From $\mathbb{E}$, we define a set $\Lambda = \{\mathbb{E}^{a,+1} \mid a \in A\}$ of predicate liftings for F, given by $\mathbb{E}^{a,+1}_X(f)(l) = \mathbb{E}_{1+X}(f^{+1})(l_a)$ for an A-indexed family l of tight probability measures l_a on $1+X$ and non-expansive $f\colon X \to [0,1]$, where $f^{+1}\colon X+1 \to [0,1]$ acts like f on X and sends the other element to 0. The arising instance of quantitative coalgebraic modal logic is van Breugel and Worrell's quantitative probabilistic modal logic [33].

Quantitative coalgebraic modal logic is invariant under coalgebra morphisms:

▶ **Proposition 8.** *Let Λ be a set of predicate liftings for a functor $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$, and let $f\colon (X,\alpha) \to (Y,\beta)$ be a morphism of F-coalgebras. Then, for every formula $\phi \in \mathcal{L}(\Lambda)$, $[\![\phi]\!]_\alpha = [\![\phi]\!]_\beta \cdot f$.*

The established approach to coalgebraic behavioural distances is to start with a set functor $\mathsf{F}\colon \mathsf{Set} \to \mathsf{Set}$ and obtain a $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$-functor as a lifting of F. In the quantalic setting, this approach may take the following shape. Topological properties of $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ entail that every set Λ of predicate liftings for a functor $\mathsf{F}\colon \mathsf{Set} \to \mathsf{Set}$ induces a functor $\mathsf{F}^\Lambda\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$, known as the **Kantorovich lifting** of F w.r.t. Λ [6]. Concretely, F^Λ sends a $\mathcal{V}$-category (X,a) to the $\mathcal{V}$-category determined by the initial structure on $\mathsf{F}X$ w.r.t. the structured cone of all maps $\lambda(f)\colon \mathsf{F}X \to |\mathcal{V}_s|$ with $\lambda \in \Lambda$ and $f\colon (X,a) \to \mathcal{V}_s \in \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$. As the name indicates, F^Λ is indeed a **lifting** of F to $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$, that is, $|-|\cdot\mathsf{F}^\Lambda = \mathsf{F}\cdot|-|$ where $|-|\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathsf{Set}$ is the forgetful functor. Every predicate lifting $\lambda \in \Lambda$ for F becomes a predicate lifting for the Kantorovich lifting F^Λ.

Kantorovich liftings are crucial prerequisites for existing expressivity results of quantitative coalgebraic logics for Set-functors (e.g. [34, 35, 22]). It turns out that the Kantorovich property can be usefully detached from the notion of functor lifting:

▶ **Definition 9** (Kantorovich Functor)**.** Let Λ be a set of predicate liftings for a functor $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$. The functor F is **Λ-Kantorovich** if for every $\mathcal{V}$-category X, the cone of all $\mathcal{V}$-functors $\lambda(f)\colon \mathsf{F}X \to \mathcal{V}_s$ with $\lambda \in \Lambda$ and $f \in \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}(X,\mathcal{V}_s)$ is initial.

Clearly, every Kantorovich lifting F^Λ is Λ-Kantorovich.

▶ **Example 10.** Recall the finite distribution functor $\mathcal{D}\colon \mathsf{Set} \to \mathsf{Set}$ from Example 1(3); in analogy to Example 7, we have the expectation predicate lifting $\mathbb{E}$ for $\mathcal{D}$, given by $\mathbb{E}_X(f)(\mu) = \sum_{x\in X} f(x)\mu(x)$ for $\mu \in \mathbb{D}X$ and $f\colon X \to [0,1]$. The corresponding lifting $\mathcal{D}^{\mathbb{E}}$ is the usual Kantorovich distance on finite distributions. The closely related functor $\mathcal{K}$ from Example 1(5) already lives on bounded pseudometric spaces (that is, on $[0,1]_{\oplus}$-$\mathsf{Cat}_{\mathsf{sym}}$), and is not a lifting of any set functor. However, $\mathcal{K}$ is Λ-Kantorovich for $\Lambda = \{\mathbb{E}\}$ where $\mathbb{E}$ is the expectation predicate lifting for $\mathcal{K}$ as in Example 7. The functor $\mathcal{D}^{\mathbb{E}}$ is a subfunctor of $\mathcal{K}$, and the components of the associated inclusion natural transformation are initial.

Using the above, one easily checks that, similarly, the functor $\mathsf{F} = \mathcal{K}(1+-)^A$ as in Example 7 is Λ-Kantorovich for $\Lambda = \{\mathbb{E}^{a,+1} \mid a \in A\}$, and the functor $\mathcal{D}^{\mathbb{E}}(1+-)^A$ is the Kantorovich lifting of the functor $\mathcal{D}(1+-)^A\colon \mathsf{Set} \to \mathsf{Set}$ w.r.t $\Lambda = \{\mathbb{E}^{a,+1} \mid a \in A\}$, where the predicate liftings $\mathbb{E}^{a,+1}$ for the $\mathcal{D}(1+-)^A$ are given analogously to the predicate liftings $\mathbb{E}^{a,+1}$ for $\mathcal{K}(1+-)^A$.

It has been shown recently that the Kantorovich functors (w.r.t. a class of predicate liftings of possibly ininite arities) are precisely the ones that preserve initial morphisms [16, Theorem 5.3].

5 Behavioural and Logical Distances

Every functor $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ comes with a natural notion of behavioural distance on F-coalgebras, defined in analogy to behavioural equivalence (which identifies two states if they can be identified under some coalgebra morphism) by regarding states as similar if they can be made similar under some coalgebra morphism:

▶ **Definition 11.** Let $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ be a functor. The **behavioural distance** on an F-coalgebra (X,a,α), denoted by bd^{F}_{α}, is defined on all $x,y \in X$ by

$$bd^{\mathsf{F}}_{\alpha}(x,y) = \bigvee\{b(f(x),f(y)) \mid f\colon (X,a,\alpha) \to (Y,b,\beta) \in \mathsf{CoAlg}(\mathsf{F})\}. \tag{1}$$

We observe next that if F preserves initial morphisms, then we can restrict the supremum (1) to morphisms carried by the identity map. Given $\mathcal{V}$-category structures a,b on X such that $a \le b$, we write $\iota_{a,b}$ for the identity $\mathcal{V}$-functor $(X,a) \to (X,b)$. We note that $\iota_{a,b}$ is a coalgebra morphism $\iota_{a,b}\colon (X,a,\alpha) \to (X,b,\beta)$, for some β, under two conditions, the first being commutativity of the diagram

$$\begin{array}{ccc} (X,a) & \xrightarrow{\quad\alpha\quad} & \mathsf{F}(X,a) \\ {\scriptstyle \iota_{a,b}}\downarrow & & \downarrow{\scriptstyle \mathsf{F}\iota_{a,b}} \\ (X,b) & \xrightarrow{\quad\mathsf{F}\iota_{a,b}\cdot\alpha\quad} & \mathsf{F}(X,b) \end{array}$$

which just means that as a map, β must be $\mathsf{F}\iota_{a,b}\cdot\alpha$. The second condition is that $\beta = \mathsf{F}\iota_{a,b}\cdot\alpha$ must be a $\mathcal{V}$-functor $(X,b) \to \mathsf{F}(X,b)$.

▶ **Proposition 12.** *Let (X,a,α) be a coalgebra for a functor $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ that preserves initial morphisms. Then, we can equivalently restrict the definition (1) of behavioural distance to morphisms of the form $f = \iota_{a,b}$:*

$$bd^{\mathsf{F}}_{\alpha}(x,y) = \bigvee\{b(x,y) \mid (X,b) \in \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}, a \le b, \mathsf{F}\iota_{a,b}\cdot\alpha \in \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}((X,b),\mathsf{F}(X,b))\}.$$

▶ Remark 13. Since the forgetful functor $|-|\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathsf{Set}$ is topological, the elements of its fiber over a set X that are greater or equal than an element (X, a) form a complete lattice $\{(X, b) \in \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \mid a \leq b\}$. Moreover, for every F-coalgebra (X, a, α), the endofunction on this complete lattice that sends a $\mathcal{V}$-category (X, b) to the $\mathcal{V}$-category given by the initial structure on X w.r.t. the structured map $|\mathsf{F}\iota_{a,b} \cdot \alpha|\colon X \to |\mathsf{F}(X, b)|$ is monotone. Therefore, by the Knaster-Tarski fixpoint theorem this map has a greatest fixpoint. By Proposition 12, if F preserves initial morphisms, then this greatest fixpoint is precisely the behavioural distance on (X, a, α). In particular, it follows that $\beta\colon (X, bd^{\mathsf{F}}_{\alpha}) \to \mathsf{F}(X, bd^{\mathsf{F}}_{\alpha})$ is a $\mathcal{V}$-functor. Furthermore, if F is a lifting of a functor $\mathsf{G}\colon \mathsf{Set} \to \mathsf{Set}$, then the behavioural distance on an F-coalgebra (X, a, α) is given by the greatest $\mathcal{V}$-categorical structure on X that makes the G-coalgebra $|\alpha|\colon X \to \mathsf{G}X$ an F-coalgebra. This is in line with the notion of behavioural distance based on liftings of Set-functors (e.g. [6, 22]).

Behavioural distance is invariant under coalgebra morphisms:

▶ **Proposition 14.** *Let $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ be a functor and $f\colon (X, a, \alpha) \to (Y, b, \beta)$ a coalgebra morphism of F-coalgebras. Then, for all $x, y \in X$, $bd^{\mathsf{F}}_{\alpha}(x, y) = bd^{\mathsf{F}}_{\beta}(f(x), f(y))$.*

Coalgebraic modal logic complements behavioural distance with a notion of logical distance:

▶ **Definition 15.** Let Λ be a set of predicate liftings for a functor $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$. The **logical distance** ld^{Λ}_{α} on an F-coalgebra (X, a, α) is the initial structure on X w.r.t. the structured cone of all maps $[\![\phi]\!]_{\alpha}\colon X \to |(\mathcal{V}, \hom_s)|$ with $\phi \in \mathcal{L}(\Lambda)$. More explicitly, for $x, y \in X$,

$$ld^{\Lambda}_{\alpha}(x, y) = \bigwedge\{\hom_s([\![\phi]\!]_{\alpha}(x), [\![\phi]\!]_{\alpha}(y)) \mid \phi \in \mathcal{L}(\Lambda)\}.$$

It is immediate from Proposition 8 that logical distance is also invariant under coalgebra morphisms. The remainder of the paper is devoted to establishing criteria under which logical distance and behavioural distance coincide. Recall that a (quantitative) coalgebraic logic is **adequate** if for every F-coalgebra (X, α), $bd^{\mathsf{F}}_{\alpha} \leq ld^{\Lambda}_{\alpha}$, and **expressive** if $ld^{\Lambda}_{\alpha} \leq bd^{\mathsf{F}}_{\alpha}$, for every F-coalgebra (X, α). The former property is straightforward to show:

▶ **Theorem 16** (Adequacy). *Let Λ be a set of predicate liftings for a functor $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$. Then, the coalgebraic logic $\mathcal{L}(\Lambda)$ is adequate.*

6 Expressivity of Quantitative Coalgebraic Modal Logic

We fix a functor $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ and a set Λ of predicate liftings for F throughout this section.

The following lemma is related to the Knaster-Tarski proof principle for expressivity identified in work on codensity liftings [22, Theorem IV.5]:

▶ **Lemma 17.** *Let (X, a, α) be an F-coalgebra. If the cone of all $\mathcal{V}$-functors $\lambda(f)\colon \mathsf{F}(X, ld^{\Lambda}_{\alpha}) \to \mathcal{V}_s$ with $\lambda \in \Lambda$ and $f \in [\![\mathcal{L}(\Lambda)]\!]_{\alpha}$ is initial, then $ld^{\Lambda}_{\alpha} \leq bd^{\mathsf{F}}_{\alpha}$.*

Kantorovich functors come with a natural strategy to show that the assumption of Lemma 17 holds. By definition, the cone of all $\mathcal{V}$-functors $(X, ld^{\Lambda}_{\alpha}) \to \mathcal{V}_s$ that are interpretations of formulas of $\mathcal{L}(\Lambda)$ is initial. Roughly speaking, one wishes to conclude from this fact that one can *approximate* every $\mathcal{V}$-functor $(X, ld^{\Lambda}_{\alpha}) \to \mathcal{V}_s$ by interpretations of formulas; then, to apply Lemma 17 we just need to guarantee that predicate liftings *preserve approximations* (that is, satisfy a notion of *continuity*), as, by the definition of Kantorovich functor, the cone of all $\mathcal{V}$-functors $\lambda(f)\colon \mathsf{F}(X, ld^{\Lambda}_{\alpha}) \to \mathcal{V}_s$ with $\lambda \in \Lambda$ and $f \in \mathcal{V}\text{-}\mathsf{Cat}((X, ld^{\Lambda}_{\alpha}), \mathcal{V}_s)$ is initial. We formalize this approach using closure operators. We begin by introducing some notation.

Given a $\mathcal{V}$-functor $i\colon Y \to X$ and a set $A \subseteq \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}(X, \mathcal{V}_s)$, we denote by $A \cdot i$ the set $\{f \cdot i \mid f \in A\}$, by $\Lambda(A)$ the set $\{\lambda(f) \mid f \in A, \lambda \in \Lambda\}$, and by $|A|$ the set $\{|f| \mid f \in A\}$ (of maps). It will be convenient to encapsulate the propositional part of the logic algebraically:

▶ **Definition 18.** Let X be a $\mathcal{V}$-category. A subset A of $\mathcal{V}_s^X$ is a **propositional algebra** if it contains the $\mathcal{V}$-functor that is constantly $\top$, and is closed under the operations $\wedge$, $\vee$, $\hom_s(u, -)$, and $u \otimes -$, for every $u \in \mathcal{V}$.

In particular, given an F-coalgebra (X, a, α), $[\![\mathcal{L}(\Lambda)]\!]_\alpha \subseteq \mathcal{V}_s^{(X, ld_\alpha^\Lambda)}$ is a propositional algebra.

We will base the mentioned notion of continuity on a notion of *closure*, over which we parametrize the technical framework (with one intended instance being L-closure as recalled in Section 3):

▶ **Definition 19** ($\mathcal{V}_s$-closure)**.** Given a set X, a **$\mathcal{V}_s$-closure operator** is a family $\mathbf{C} = (\mathbf{C}_X)_{X \in \mathsf{Set}}$ of closure operators $\mathbf{C}_X$ on $\mathsf{Set}(X, \mathcal{V})$ (i.e. operators $\mathbf{C}_X\colon \mathcal{P}(\mathsf{Set}(X, \mathcal{V})) \to \mathcal{P}(\mathsf{Set}(X, \mathcal{V}))$ satisfying the standard *extensiveness*, *monotonicity* and *idempotence* laws) such that for every symmetric $\mathcal{V}$-category (X, a), $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}((X, a), \mathcal{V}_s) \subseteq \mathsf{Set}(X, \mathcal{V})$ is closed w.r.t. $\mathbf{C}_X$. When no ambiguities arise, we write $\mathbf{C}(A)$ instead of $\mathbf{C}_X(A)$.

A $\mathcal{V}_s$-closure operator $\mathbf{C}$ is equivalently given by a family $\overline{\mathbf{C}} = (\overline{\mathbf{C}}_X)_{X \in \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}}$ of closure operators on $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}(X, \mathcal{V}_s)$ such that for all $A \subseteq \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}(X, \mathcal{V}_s)$ and $B \subseteq \mathcal{V}\text{-}\mathsf{Cat}(Y, \mathcal{V}_s)$, where $|Y| = |X|$, if $|A| = |B|$ then $|\overline{\mathbf{C}}_X(A)| = |\overline{\mathbf{C}}_Y(B)|$. We make no distinction between $\mathbf{C}$ and $\overline{\mathbf{C}}$, e.g. we apply $\mathbf{C}$ also to $A \subseteq \mathcal{V}_s^X = \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}(X, \mathcal{V}_s)$, for a $\mathcal{V}$-category X. In particular, we say that $A \subseteq \mathcal{V}_s^X$ is **C-dense on** X if $\mathbf{C}(A) = \mathcal{V}_s^X$.

▶ **Example 20.** The following closure operators on $\mathsf{Set}(X, \mathcal{V})$ are $\mathcal{V}_s$-closure operators:

1. the identity operator $\mathbf{Id}_X$;
2. the operator $\mathbf{L}_X^{\mathcal{V}_s}$ that sends every set to its L-closure in the $\mathcal{V}$-category $\mathcal{V}_s^X$;
3. the closure operator $\mathbf{cInf}^\vee{}_X$ that sends every set A to its closure under codirected infima and finite suprema;
4. the closure operator $\mathbf{Inf}_X$ that sends every set A to its closure under infima;
5. the closure operator $\mathbf{Fun}_X$ that sends every set A to $|\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}(X_A, \mathcal{V}_s)|$, where X_A denotes the $\mathcal{V}$-category determined by the initial structure with respected to the structured cone of all maps $f\colon X \to |\mathcal{V}_s|$ with $f \in A$ (**Fun** is in fact the closure operator of a Galois connection, relating to recent work by Beohar et al. [7]).

While **Id** is the *least* $\mathcal{V}_s$-closure operator, somewhat less trivially **Fun** is the *greatest* one (and hence induces the *weakest* notion of density).

The next result connects initiality, closure and density.

▶ **Proposition 21.** *Let* $\mathbf{C}$ *be a* $\mathcal{V}_s$*-closure operator. For every* $\mathcal{V}$*-category* X *and* $A \subseteq \mathcal{V}_s^X$*,*

1. *if* A *is* $\mathbf{C}$*-dense on* X*, then* A *is initial; for* $\mathbf{C} = \mathbf{Fun}$*, the converse holds as well;*
2. *if* $\mathbf{C}(A)$ *is initial, then* A *is initial.*

▶ **Definition 22.** Let $\mathbf{C}$ be a $\mathcal{V}_s$-closure operator. A predicate lifting $\lambda \in \Lambda$ is **C-continuous** if for every $\mathcal{V}$-category X, $\lambda_X\colon \mathcal{V}_s^X \to \mathcal{V}_s^{\mathsf{F}X}$ is continuous w.r.t. $\mathbf{C}_X$ and $\mathbf{C}_{\mathsf{F}X}$ (i.e. $f[\mathbf{C}_X(A)] \subseteq \mathbf{C}_{\mathsf{F}X}(f[A])$ for all $A \subseteq \mathcal{V}_s^X$).

▶ **Example 23.** A predicate lifting λ is

1. always **Id**-continuous;
2. **L**-continuous iff its components are L-continuous;
3. $\mathbf{cInf}^\vee$-continuous iff its components preserve codirected infima and finite suprema;
4. **Inf**-continuous iff its components preserve all infima.

It is easily verified that if a predicate lifting λ for a $\{\lambda\}$-Kantorovich functor is **Fun**-continuous, then it preserves initial cones. Thus, **Fun**-continuity is a very strong assumption, which by Lemma 17 entails that $\mathcal{L}(\{\lambda\})$ is expressive. In order to obtain expressivity results for coalgebraic logics under weaker assumptions, we will consider situations where **Fun**-density can be equivalently described as **C**-density for more suitable $\mathcal{V}_s$-closure operators **C**.

▶ **Definition 24.** Let $\mathcal{I}$ be a class of symmetric $\mathcal{V}$-categories. A $\mathcal{V}_s$-closure operator **C** **characterizes initiality** on $\mathcal{I}$ if for every $\mathcal{V}$-category $X \in \mathcal{I}$, every propositional algebra $A \subseteq \mathcal{V}_s^X$ that is **Fun**-dense on X (that is, by Proposition 21, initial) is already **C**-dense on X (recall that the reverse implication holds universally).

Characterization of initiality for a given class $\mathcal{I}$ depends only on the quantale and the closure operator, and may be seen as a form of Stone-Weierstraß property; we will give general Stone-Weierstraß theorems for some classes of quantales in Section 7. In most of these, $\mathcal{I}$ will be the class of finite symmetric $\mathcal{V}$-categories (and in one case, the class of totally bounded pseudometric spaces). We introduce next a key technical definition.

▶ **Definition 25.** Let **C** be a $\mathcal{V}_s$-closure operator. A cocone $(i\colon X_i \to X)_{i\in\mathcal{I}}$ of morphisms in $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ **coreflects C-density** if the cone $(-\cdot i\colon \mathcal{V}_s^X \to \mathcal{V}_s^{X_i})_{i\in\mathcal{I}}$ **reflects C-density**; that is, for every $A \subseteq \mathcal{V}_s^X$, if $A \cdot i$ is **C**-dense for every $i \in \mathcal{I}$, then A is **C**-dense.

Since by Proposition 21, **Fun**-density is equivalent to initiality, we refer to coreflection of **Fun**-density also as **coreflection of initiality**.

▶ **Example 26.**

1. An initial $\mathcal{V}$-functor coreflects **Id**-density iff it is L-dense.
2. A classical 1-bounded metric space (X, d) is totally bounded if for every $u > 0$ there is a finite set X_u such that for every $x \in X$ there is $y \in X_u$ so that $d(x, y) < u$. It can be shown that for every totally bounded metric space (X, d), the cocone of embeddings of finite subspaces coreflects **L**-density.
3. It follows from [19, Lemma 1.10(4)] that every jointly L-dense directed cocone coreflects initiality. In particular, every directed colimit coreflects initiality.

Since in general, we can only replace **Fun**-density with **C**-density in a restricted class of $\mathcal{V}$-categories, our results will depend on functors that are compatible with such a class.

▶ **Definition 27.** A class $\mathcal{I}$ of symmetric $\mathcal{V}$-categories **coreflects initiality under** F if for every $\mathcal{V}$-category X, the cocone

$$(\mathsf{F}i\colon \mathsf{F}Y \to \mathsf{F}X)_{Y\in\mathcal{I},\, i\colon Y\to X \text{ initial}}$$

coreflects initiality.

▶ **Example 28.**

1. The class $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ coreflects initiality under every $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$-functor.
2. The class of all finite symmetric $\mathcal{V}$-categories coreflects initiality under Kantorovich liftings of finitary Set-functors to $\mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$.

The second example above indicates that coreflection of initiality relates to size bounds on the functor. The following proposition shows that such size bounds are only needed up to approximation.

▶ **Proposition 29.** *Let $j\colon \mathsf{G} \to \mathsf{F}$ be a natural transformation between $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$-functors such that each component of j is initial and L-dense. If a class $\mathcal{I}$ of symmetric $\mathcal{V}$-categories coreflects initiality under G, then $\mathcal{I}$ coreflects initiality under F.*

For instance, combining Example 28.2 and Proposition 29, we obtain:

▶ **Corollary 30.** *If G is a Kantorovich lifting of a finitary set functor and $j\colon \mathsf{G} \to \mathsf{F}$ is a natural transformation between $\mathcal{V}$-$\mathsf{Cat}_{\mathsf{sym}}$-functors such that each component of j is initial and L-dense, then the class of all finite symmetric $\mathcal{V}$-categories coreflects initiality under F.*

▶ Remark 31. The previous notion of a *finitarily separable* lax extension L of a set functor F_0 [34] requires essentially that the finitary part of F_0 is L-dense in the lifting F of F_0 induced by L (in a way that is immaterial here), that is, if for every $\mathcal{V}$-category X, the set $\{\mathsf{F}i(t) \mid Y \text{ finite},\ i\colon Y \to X,\ t \in \mathsf{F}Y\}$ is L-dense in $\mathsf{F}X$. For instance, the standard Kantorovich lifting of the discrete distribution functor is finitarily separable [34]. For finitarily separable F, the class of finite symmetric $\mathcal{V}$-categories coreflects initiality by Corollary 30.

We are now ready to present our main result:

▶ **Theorem 32** (Quantitative coalgebraic Hennessy-Milner theorem)**.** *Let $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ be Λ-Kantorovich, let $\mathbf{C}$ be a $\mathcal{V}_s$-closure operator, and let $\mathcal{I}$ be a class of symmetric $\mathcal{V}$-categories such that $\mathcal{I}$ coreflects initiality under F and $\mathbf{C}$ characterizes initiality on $\mathcal{I}$. If every predicate lifting in Λ is $\mathbf{C}$-continuous, then $\mathcal{L}(\Lambda)$ is expressive.*

We note that there is a balance to be struck in the choice of $\mathcal{I}$: The larger $\mathcal{I}$ is, the more functors one finds under which $\mathcal{I}$ coreflects initiality, but the harder it is to establish a characterization of initiality in $\mathcal{I}$. We tackle the latter issue next.

7 Stone-Weierstraß-Type Theorems

We now develop some usage scenarios for Theorem 32; concrete expressivity proofs using these criteria will be given in Section 8. As a warm-up, we have

▶ **Theorem 33.** *Let $\mathcal{V}$ be a finite quantale. Then the $\mathcal{V}_s$-closure operator $\mathbf{Id}$ characterizes initiality on finite symmetric $\mathcal{V}$-categories.*

(Note that for $\mathcal{V} = 2$, this is essentially the well-known functional completeness of Boolean logic.) Hence, by instantiating Theorem 32, we obtain

▶ **Corollary 34.** *Let $\mathcal{V}$ be a finite quantale, and let $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ be a Λ-Kantorovich functor that admits as an L-dense subfunctor a lifting of a finitary Set-functor. Then the coalgebraic logic $\mathcal{L}(\Lambda)$ is expressive.*

Most remarkably, while Corollary 34 allows us to derive expressivity for many-valued logics, it also relates our present expressivity criterion to the known criterion for the properly qualitative case (i.e. for $\mathcal{V} = 2$) [27, 30]. To that end, recall that a set Λ of predicate liftings for a functor $\mathsf{F}\colon \mathsf{Set} \to \mathsf{Set}$ is **separating** if for every set X the cone of all maps $\lambda(f)\colon \mathsf{F}X \to 2$ with $\lambda \in \Lambda$ and $f\colon X \to 2$ is mono. For a Set-coalgebra (X, α), let us denote by beq_α the standard notion of behavioural equivalence, as explained in preliminaries. Let $\mathsf{Equ} = 2\text{-}\mathsf{Cat}_{\mathsf{sym}}$ (the category of equivalence relations).

The following result generalizes [26, Theorem 11] (which applies only to functor liftings that arise from lax extensions, in particular requires modalities to be monotone):

▶ **Theorem 35.** *Let $\mathsf{F}^\Lambda\colon \mathsf{Equ} \to \mathsf{Equ}$ be a Kantorovich lifting that preserves discrete equivalence relations. Then, for every F-coalgebra (X, α), $bd_\alpha^{\mathsf{F}^\Lambda} = beq_\alpha$.*

We now can recover the general expressivity result [27, 30] for $\mathcal{V} = 2$ as a direct consequence of Corollary 34 and Theorem 35.

▶ **Theorem 36** (Qualitative coalgebraic Hennessy-Milner theorem)**.** *Let Λ be a separating set of predicate liftings for a finitary functor $\mathsf{F}\colon \mathsf{Set} \to \mathsf{Set}$. Then the coalgebraic logic $\mathcal{L}(\Lambda)$ is expressive; that is, if two states in an F-coalgebra are logically indistinguishable, then they are behaviourally equivalent.*

Next, we obtain a characterization of $\mathbf{L}$-density. Recall that an element x of an ordered set is *way above* an element y if whenever $y \geq \bigwedge A$ for a codirected set A, then $x \geq a$ for some $a \in A$.

▶ **Theorem 37.** *Suppose that $\mathcal{V}$ satisfies the condition*

$$k = \bigvee \{u \otimes u \mid u \in \mathcal{V} \text{ and for all } v \in \mathcal{V},\ \hom(u, v) \text{ is way above } v\}. \tag{2}$$

Then $\mathbf{L}$ characterizes initiality on finite symmetric $\mathcal{V}$-categories.

We thus obtain the following quantalic generalization of the previous coalgebraic Hennessy-Milner theorem for finitarily separable $[0,1]_\oplus$-lax extensions [34, Corollary 8.6]:

▶ **Corollary 38.** *Let $\mathcal{V}$ be a quantale satisfying (2), and let $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ be a Λ-Kantorovich functor that admits as an L-dense subfunctor a lifting of a finitary Set-functor. If every predicate lifting in Λ is $\mathbf{L}$-continuous, then $\mathcal{L}(\Lambda)$ is expressive.*

Specifically, besides allowing more general quantales, we drop the assumptions that the modalities in Λ are monotone and that F is a lifting of a Set-functor, and we weaken the assumption of non-expansiveness of predicate liftings to $\mathbf{L}$-continuity.

The following fact sometimes allows us to enlarge the class on which initiality is characterized:

▶ **Proposition 39.** *Let $\mathbf{C}$ be a $\mathcal{V}_s$-closure operator that characterizes initiality on $\mathcal{I}$, and let $\mathcal{J}$ be a class of symmetric $\mathcal{V}$-categories. If for every $X \in \mathcal{J}$, the cocone $(i\colon Y \to X)_{Y \in \mathcal{I},\, i \text{ initial}}$ coreflects $\mathbf{C}$-density, then $\mathbf{C}$ characterizes initiality on $\mathcal{J}$.*

In particular, we obtain by Example 26.2 that for $\mathcal{V} = [0,1]_\oplus$, $\mathbf{L}$ characterizes initiality on totally bounded pseudometric spaces, so we can, in this case, further relax the assumptions of Corollary 38 as follows.

▶ **Definition 40.** A functor $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$, for $\mathcal{V} = [0,1]_\oplus$, is *totally bounded* if for every symmetric $\mathcal{V}$-category X and every $t \in \mathsf{F}X$, there exists a totally bounded $X_0 \subseteq X$ and $t' \in \mathsf{F}X_0$ such that $t = \mathsf{F}i(t')$, where i is the inclusion $X_0 \to X$.

▶ **Corollary 41.** *Let $\mathsf{F}\colon [0,1]_\oplus\text{-}\mathsf{Cat}_{\mathsf{sym}} \to [0,1]_\oplus\text{-}\mathsf{Cat}_{\mathsf{sym}}$ be a Λ-Kantorovich functor that admits an L-dense totally bounded subfunctor. If every predicate lifting in Λ is $\mathbf{L}$-continuous, then $\mathcal{L}(\Lambda)$ is expressive.*

We conclude with some variants employing order-theoretic closure operators:

▶ **Theorem 42.** *Let $\mathcal{V}$ be a quantale such that for every $u \in \mathcal{V}$ the map $u \otimes -$ preserves codirected infima. Then the closure operator $\mathbf{cInf}^\vee$ characterizes initiality on finite symmetric $\mathcal{V}$-categories.*

Notice that the assumption on $\mathcal{V}$ in the above theorem is satisfied in particular when $\mathcal{V}$ is a frame (Example 2.1).

▶ **Corollary 43.** *Let $\mathcal{V}$ be a quantale such that for every $u \in \mathcal{V}$ the map $u \otimes -$ preserves codirected infima, and let $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ be a Λ-Kantorovich functor that admits as an L-dense subfunctor a lifting of a finitary Set-functor. If every predicate lifting in Λ preserves codirected infima and finite suprema, then the coalgebraic logic $\mathcal{L}(\Lambda)$ is expressive.*

▶ **Theorem 44.** *Let $\mathcal{V}$ be a completely distributive quantale such that for every $u \in \mathcal{V}$ the map $u \otimes -$ preserves codirected infima. Then the closure operator* **Inf** *characterizes initiality on finite symmetric $\mathcal{V}$-categories.*

▶ **Corollary 45.** *Let $\mathcal{V}$ be a completely distributive quantale such that for every $u \in \mathcal{V}$ the map $u \otimes -$ preserves codirected infima, and let $\mathsf{F}\colon \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}} \to \mathcal{V}\text{-}\mathsf{Cat}_{\mathsf{sym}}$ be a Λ-Kantorovich functor that admits as an L-dense subfunctor a lifting of a finitary Set-functor. If every predicate lifting in Λ preserves all infima, then the coalgebraic logic $\mathcal{L}(\Lambda)$ is expressive.*

▶ Remark 46. We note that the preservation conditions imposed on the predicate liftings in Corollary 43 and Corollary 45 (which instantiate the continuity condition from Theorem 32) are quite restrictive: Modalities must be diamond-like in Corollary 43, and box-like in Corollary 45. On the other hand, these conditions depend only on the underlying lattice of a quantale and, therefore, have the potential to be applied to multiple quantales based on the same lattice.

8 Examples

We apply the concretizations of the quantitiative coalgebraic Hennessy-Milner theorem (Theorem 32) proved in the previous section (Corollaries 34, 38, 41, 43, and 45) to the logics discussed in Example 1 and variants thereof.

1. Metric transition systems. We consider *finitely branching metric transition systems* in which every *state* is labelled with a non-negative extended real number number (while transitions are unlabelled). Such systems are coalgebras for the functor $\mathsf{F} = [0,\infty] \times \mathcal{P}_\omega$, where $\mathcal{P}_\omega$ denotes the finite powerset functor. Also, we aim for an example based on *ultrametrics*, so we we consider the quantale $[0,\infty]_{\max}$ (see Example 2(3a)). We define a nullary predicate lifting o (formally accommodated as a unary predicate lifting that ignores its argument) by $o_X(r,S) = r$, and a unary predicate lifting $\Diamond$ by

$$\Diamond_X(f)(r,S) = \bigvee_{x\in S} f(x).$$

We write $\bar{\mathsf{F}}$ for the Kantorovich lifting of F under the set Λ of these modalities. We then obtain that the coalgebraic logic $\mathcal{L}(\Lambda)$ is expressive, via Corollary 43: $\bar{\mathsf{F}}$ is itself a lifting of a finitary set functor, o trivially preserves all infima and suprema, and one checks easily that $\Diamond$ preserves codirected infima and finite suprema. Similarly, by replacing the predicate lifting $\Diamond$ with the predicate lifting $\Box$ given by

$$\Box_X(f)(r,S) = \bigwedge_{x\in S} f(x).$$

we obtain that the corresponding coalgebraic logic $\mathcal{L}(\Lambda)$ is expressive via Corollary 45. We note that these results still hold if, for example, we replace the quantale $[0,\infty]_{\max}$ with the quantale $[0,\infty]_+$ (See Example 2(3b)); that is, if we are interested in all generalized metric spaces, not only in the ultrametric ones. The metric version of these results relates to known characteristic logics for metric transition systems [10, 31]; the ultrametric versions appear to be new.

2. $\Diamond$-valued powerset. The lattice $\Diamond$ from Example 1.2 can be equipped with the structure of quantale by defining a commutative operator $*$, with B as a unit, $\bot$ as a zero and $\mathsf{N} * \mathsf{N} = \bot$, $\mathsf{N} * \top = \mathsf{N}$, $\top * \top = \top$. The resulting quantale is finite, and hence our general Corollary 34 applies to it, but not the previously existing expressivity theorem for quantale-valued distances [35], for this quantale is not a value quantale [14]. The induced logic is a paraconsistent four-valued logic with $\mathcal{L}(\Lambda)$ instantiated as follows:

$$\phi ::= \top \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid u * \phi \mid \hom_s(u, \phi) \mid \lambda(\phi) \qquad (u \in \mathcal{V}, \lambda \in \Lambda)$$

where Λ can be defined in different ways, and the expressivity result remains true for any such choice (for the respective Kantorovich lifting, which also depends on Λ). A natural choice is $\Lambda = \{\Diamond\}$ where $\Diamond$ is interpreted as an instance of a generic formula: $\Diamond_X(p \in \mathcal{B}X)(q\colon X \to \mathcal{V}) = \bigvee_{x \in X} p(x) * q(x)$. This predicate lifting was previously considered by Rivieccio et al [28], who interpreted it over generalized Kripke frames, which are precisely coalgebras for the functor $\mathcal{B}$ as per Example 1.2.

3. Discrete probabilistic transition systems. As mentioned already in Example 10, the functor $\mathcal{D}^{\mathbb{E}}(1 + -)^A$ is the Kantorovich lifting of the finitary functor $\mathcal{D}(1 + -)^A$ w.r.t $\Lambda = \{\mathbb{E}^{a,+1} \mid a \in A\}$. Moreover, it is easy to see that every predicate lifting in Λ gives rise to an $\mathbf{L}$-continuous predicate lifting for $\mathcal{D}^{\mathbb{E}}(1 + -)^A$. Hence, we obtain by Corollary 38 that quantitative probabilistic modal logic – the coalgebraic modal logic generated by the expectation modality – is expressive [32, 2].

4. Weighted transition systems with negative weights. The functor $\mathcal{W}$ defining our variant of weighted transition systems is Kantorovich for the set Λ of (*non-monotone*) predicate liftings $\langle a \rangle^{+r}$, for $a \in A$ and $r \in \mathbb{R}$, defined by

$$[\![\langle r \rangle]\!](f)(t) = \min\Big\{1, \max\Big\{0, r + \frac{1}{2}\textstyle\sum_{x \in X} f(x)t(a)(x)\Big\}\Big\}$$

Since $\mathcal{W}$ is, moreover, a lifting of a finitary set functor, we obtain by Corollary 38 the new result that the coalgebraic modal logic $\mathcal{L}(\Lambda)$ is expressive.

5. Continuous probabilistic transition systems. Our variant $\mathcal{K}$ of the Kantorovich functor admits, by definition, a totally bounded L-dense subfunctor that assigns to a space X the set of all Borel distributions on X with totally bounded support. Hence, as A is finite, it follows that the functor $\mathcal{K}(1 + -)^A$ admits a totally bounded L-dense subfunctor. Furthermore, as noted already in Example 10, the functor $\mathcal{K}(1 + -)^A$ is Λ-Kantorovich, where $\Lambda = \{\mathbb{E}^{a,+1} \mid a \in A\}$, and it is easy to see that every predicate lifting in Λ is $\mathbf{L}$-continuous. Therefore, we obtain by Corollary 41 that the coalgebraic modal logic $\mathcal{L}(\Lambda)$ is expressive, thus essentially recovering expressivity of quantitative probabilistic modal logic on continuous probabilistic transition systems [33, 32].

9 Conclusions and Further Work

We have presented a quantitative Hennessy-Milner theorem in coalgebraic and quantalic generality, covering behavioural distances on a wide range of system types. Notably, our results apply to functors on metric spaces that fail to be liftings of any set functor, such as the (tight) Borel distribution functor. A key factor in the technical development was the interplay between notions of density on the one hand, and initiality of cones in the topological category of generalized metric spaces taking values in a quantale $\mathcal{V}$ ($\mathcal{V}$-*categories*)

on the other hand. We have illustrated how to instantiate our results in several salient cases, in particular continuous probabilistic transition systems and weighted transition systems allowing negative weights.

For simplicity, we have worked exclusively with symmetric $\mathcal{V}$-categories throughout; nevertheless, we stress that our results carry over straightforwardly to the non-symmetric case, which covers quantitative analogues of simulation preorders (indeed, some of the existing quantitative coalgebraic Hennessy-Milner theorems already do apply to non-symmetric distances [35, 22, 36]). In fact, we expect our main expressivity theorem to be easily transported to topological categories that admit an initial dense object (which takes the role of $\mathcal{V}_s$). We leave this issue to future work. Another important direction is to develop a general coalgebraic treatment of characteristic logics for non-branching-time (e.g. linear-time) behavioural distances (e.g. [10, 12]), possibly building on recent results in this direction [7].

References

1 Jiří Adámek, Horst Herrlich, and George E. Strecker. *Abstract and concrete categories: The joy of cats.* John Wiley & Sons Inc., 1990. Republished in: Reprints in Theory and Applications of Categories, No. 17 (2006) pp. 1–507. URL: http://tac.mta.ca/tac/reprints/articles/17/tr17abs.html.

2 Jirí Adámek, Stefan Milius, Lawrence S. Moss, and Henning Urbat. On finitary functors and their presentations. *J. Comput. Syst. Sci.*, 81(5):813–833, 2015. doi:10.1016/j.jcss.2014.12.002.

3 Ki Yung Ahn, Ross Horne, and Alwen Tiu. A characterisation of open bisimilarity using an intuitionistic modal logic. In Roland Meyer and Uwe Nestmann, editors, *Concurrency Theory, CONCUR 2017*, volume 85 of *LIPIcs*, pages 7:1–7:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.7.

4 Claudi Alsina, Maurice J. Frank, and Berthold Schweizer. *Associative functions. Triangular norms and copulas.* World Scientific, 2006. doi:10.1142/9789812774200.

5 Steve Awodey. *Category Theory.* Oxford University Press, 2nd edition, 2010.

6 Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König. Coalgebraic behavioral metrics. *Log. Methods Comput. Sci.*, 14(3):1860–5974, 2018. doi:10.23638/lmcs-14(3:20)2018.

7 Harsh Beohar, Sebastian Gurke, Barbara König, and Karla Messing. Hennessy-Milner theorems via Galois connections. In Bartek Klin and Elaine Pimentel, editors, *Computer Science Logic, CSL 2023*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. This volume.

8 Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Jirí Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *Formal Modeling and Analysis of Timed Systems, FORMATS 2008*, volume 5215 of *LNCS*, pages 33–47. Springer, 2008. doi:10.1007/978-3-540-85778-5_4.

9 Corina Cîrstea, Alexander Kurz, Dirk Pattinson, Lutz Schröder, and Yde Venema. Modal logics are coalgebraic. *Comput. J.*, 54(1):31–41, 2011. doi:10.1093/comjnl/bxp004.

10 Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching system metrics. *IEEE Trans. Software Eng.*, 35(2):258–273, 2009. doi:10.1109/TSE.2008.106.

11 Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes. *Inf. Comput.*, 179(2):163–193, 2002. doi:10.1006/inco.2001.2962.

12 Uli Fahrenberg, Axel Legay, and Claus Thrane. The quantitative linear-time–branching-time spectrum. In Supratik Chakraborty and Amit Kumar, editors, *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011*, volume 13 of *LIPIcs*, pages 103–114. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.103.

13 Ihor Filimonov, Ross Horne, Sjouke Mauw, and Zach Smith. Breaking unlinkability of the ICAO 9303 standard for e-passports using bisimilarity. In Kazue Sako, Steve A. Schneider, and Peter Y. A. Ryan, editors, *Computer Security, ESORICS 2019*, volume 11735 of *LNCS*, pages 577–594. Springer, 2019. doi:10.1007/978-3-030-29959-0_28.

14 Robert Flagg. Quantales and continuity spaces. *Algebra Univ.*, 37(3):257–276, 1997.

15 Alessandro Giacalone, Chi-Chang Jou, and Scott Smolka. Algebraic reasoning for probabilistic concurrent systems. In Manfred Broy, editor, *Programming concepts and methods, PCM 1990*, pages 443–458. North-Holland, 1990.

16 Sergey Goncharov, Dirk Hofmann, Pedro Nora, Lutz Schröder, and Paul Wild. Kantorovich functors and characteristic logics for behavioural distances, 2022. doi:10.48550/arXiv.2202.07069.

17 Helle Hvid Hansen, Clemens Kupke, and Eric Pacuit. Neighbourhood structures: Bisimilarity and basic model theory. *Log. Methods Comput. Sci.*, 5(2), 2009. doi:10.2168/LMCS-5(2:2)2009.

18 Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985. doi:10.1145/2455.2460.

19 Dirk Hofmann and Walter Tholen. Lawvere completion and separation via closure. *Applied Categorical Structures*, 18(3):259–287, November 2010. doi:10.1007/s10485-008-9169-9.

20 Bartek Klin. Structural operational semantics for weighted transition systems. In Jens Palsberg, editor, *Semantics and Algebraic Specification, Essays Dedicated to Peter D. Mosses on the Occasion of his 60th Birthday*, volume 5700 of *LNCS*, pages 121–139. Springer, 2009. doi:10.1007/978-3-642-04164-8_7.

21 Yuichi Komorida, Shin-ya Katsumata, Nick Hu, Bartek Klin, and Ichiro Hasuo. Codensity games for bisimilarity. In *Logic in Computer Science, LICS 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785691.

22 Yuichi Komorida, Shin-ya Katsumata, Clemens Kupke, Jurriaan Rot, and Ichiro Hasuo. Expressivity of quantitative modal logics : Categorical foundations via codensity and approximation. In *Logic in Computer Science, LICS 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470656.

23 Barbara König and Christina Mika-Michalski. (Metric) bisimulation games and real-valued modal logics for coalgebras. In Sven Schewe and Lijun Zhang, editors, *Concurrency Theory, CONCUR 2018*, volume 118 of *LIPIcs*, pages 37:1–37:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.37.

24 Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991. doi:10.1016/0890-5401(91)90030-6.

25 F. William Lawvere. Metric spaces, generalized logic, and closed categories. *Rendiconti del Seminario Matemàtico e Fisico di Milano*, 43(1):135–166, 1973. Republished in: Reprints in Theory and Applications of Categories, No. 1 (2002), 1–37. doi:10.1007/bf02924844.

26 Johannes Marti and Yde Venema. Lax extensions of coalgebra functors and their logic. *J. Comput. System Sci.*, 81(5):880–900, 2015. doi:10.1016/j.jcss.2014.12.006.

27 Dirk Pattinson. Expressive logics for coalgebras via terminal sequence induction. *Notre Dame Journal of Formal Logic*, 45(1):19–33, 2004. doi:10.1305/ndjfl/1094155277.

28 Umberto Rivieccio, Achim Jung, and Ramon Jansana. Four-valued modal logic: Kripke semantics and duality. *J. Log. Comput.*, 27(1):155–199, 2017. doi:10.1093/logcom/exv038.

29 J. Rutten. Universal coalgebra: A theory of systems. *Theor. Comput. Sci.*, 249:3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

30 Lutz Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *Theor. Comput. Sci.*, 390(2-3):230–247, 2008. doi:10.1016/j.tcs.2007.09.023.

31 Franck van Breugel. A behavioural pseudometric for metric labelled transition systems. In Martín Abadi and Luca de Alfaro, editors, *Concurrency Theory, CONCUR 2005*, volume 3653 of *LNCS*, pages 141–155. Springer, 2005. doi:10.1007/11539452_14.

32 Franck van Breugel, Claudio Hermida, Michael Makkai, and James Worrell. Recursively defined metric spaces without contraction. *Theor. Comput. Sci.*, 380(1-2):143–163, 2007. doi:10.1016/j.tcs.2007.02.059.

33 Franck van Breugel and James Worrell. A behavioural pseudometric for probabilistic transition systems. *Theor. Comput. Sci.*, 331(1):115–142, 2005. doi:10.1016/j.tcs.2004.09.035.

34 Paul Wild and Lutz Schröder. Characteristic logics for behavioural metrics via fuzzy lax extensions. In Igor Konnov and Laura Kovács, editors, *Concurrency Theory, CONCUR 2020*, volume 171 of *LIPIcs*, pages 27:1–27:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.27.

35 Paul Wild and Lutz Schröder. A quantified coalgebraic van Benthem theorem. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures, FOSSACS 2021*, volume 12650 of *LNCS*, pages 551–571. Springer, 2021. doi:10.1007/978-3-030-71995-1_28.

36 Paul Wild and Lutz Schröder. Characteristic logics for behavioural hemimetrics via fuzzy lax extensions. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/lmcs-18(2:19)2022.

Order-Invariance in the Two-Variable Fragment of First-Order Logic

Julien Grange ✉
LACL, Université Paris-Est Créteil, France

Abstract

We study the expressive power of the two-variable fragment of order-invariant first-order logic. This logic departs from first-order logic in two ways: first, formulas are only allowed to quantify over two variables. Second, formulas can use an additional binary relation, which is interpreted in the structures under scrutiny as a linear order, provided that the truth value of a sentence over a finite structure never depends on which linear order is chosen on its domain.

We prove that on classes of structures of bounded degree, any property expressible in this logic is definable in first-order logic. We then show that the situation remains the same when we add counting quantifiers to this logic.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases Finite model theory, Two-variable logic, Order-invariance

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.23

1 Introduction

The restriction of first-order logic to two variables (FO^2) holds an important place among the fragments of first-order logic (FO), since it is the maximal fragment, with respect to the number of variables, for which the finite and general satisfiability problems are decidable [14] – see [5] for a complete survey on these issues. They become undecidable as soon as we allow formulas to make use of three variables, as three variables are enough to encode grids, and thus runs of Turing machines. Tame as it is in this regard, FO^2 is still fairly expressive (in particular, it embeds modal logic). It is thus natural to investigate its order-invariant extension $<$-inv FO^2.

In the order-invariant extension $<$-inv $\mathcal{L}$ of a logic $\mathcal{L}$, one can make use in the $\mathcal{L}$-formulas of a linear order on the vertices of the structures at hand, provided that the validity of said formulas in a given finite structure does not depend on the choice of a particular order. Such a notion is very natural and useful both in database theory (where it corresponds to the requirement for a query to be independent of the order in which the data is stored on disk) and in descriptive complexity (where structures are needed to be ordered for a logic to capture complexity classes such as PTIME [10] and PSPACE [17]).

If $\mathcal{L} = \mathrm{FO}$, we get $<$-inv FO, whose syntax is not recursively enumerable. It has however been proven by Harwath and Zeume [19] that on the other hand, $<$-inv FO^2 has a recursive syntax. The natural follow-up to this result is to study the expressive power of $<$-inv FO^2. This shall be our endeavor in this article.

Perhaps surprisingly, it has been shown by Gurevich (see Section 5.2 of [13]) that such an order can indeed bring additional expressive power to FO, even when restricted in this way: there exist properties which are not definable in FO, but which can be expressed as soon as the use of a linear order is authorized, even in an invariant fashion. In symbols: $<$-inv $\mathrm{FO} \not\subseteq \mathrm{FO}$. It is not hard to observe that $<$-inv $\mathrm{FO}^2 \not\subseteq \mathrm{FO}^2$ (for instance, one can state in $<$-inv FO^2 that a set has at least three elements, which is not possible in FO^2). It is however not clear whether $<$-inv $\mathrm{FO}^2 \subseteq \mathrm{FO}$, or whether even when restricted to two

variables, the addition of an order allows one to express properties beyond the scope of FO.[a] This question is asked in [19]. Is this paper, we prove that on any class of bounded degree, the inclusion $<$-inv $\mathrm{FO}^2 \subseteq \mathrm{FO}$ indeed holds - this is Theorem 1, whose proof is the object of Sections 4 and 5. We then explain in Section 6 how to extend this result from $<$-inv FO^2 to $<$-inv C^2, where C^2 is the extension of FO^2 with counting quantifiers. Precisely, we show with Theorem 10 that $<$-inv $\mathrm{C}^2 \subseteq \mathrm{FO}$ when the degree is bounded. Let us already mention that both of these inclusions are strict. This matter is further discussed in the conclusion.

Related work

The question of the order-invariance of an FO^2-sentence has been shown to be decidable in [19]. The expressive power of $<$-inv FO^2 is only mentioned there as a follow up question.

We borrow the dichotomy between rare and frequent neighborhood types when the degree is bounded from [7]. Beyond that, the philosophies of the constructions differ widely: in [7], successor relations are constructed in a very regular way, in order to create as few neighborhood types as possible in the structures with successor. On the other hand, we make sure to realize all the possible types in our construction.

One line of research investigates the expressive power of $<$-inv FO. Let us mention [3] and [8], which prove that $<$-inv FO has the same expressive power as FO respectively on trees and on hollow trees. The present paper focuses on a weaker logic, but in a broader setting. Furthermore, while the techniques used in these papers involve the construction of several intermediate orders, making only localized changes at each step (in the fashion of [9], in which it is proved that $<$-inv FO retains the locality of FO), we equip in one go each of our structures with a linear order.

Although not directly related to our inquiries, [18] is also concerned with the expressive power of FO^2 on ordered structures. This paper establishes a strict hierarchy, based on the quantifier rank and quantifier alternation, among properties definable in FO^2 on words.

2 Preliminaries

We use the standard definition of first-order logic $\mathrm{FO}(\Sigma)$ with equality (written FO when Σ is clear from the context) on a finite signature Σ composed of relation and constant symbols. By FO^2 we denote the fragment of FO in which the only two variables are x and y.

Structures are denoted by a name that starts with (or consists of) a calligraphic upper-case letter, while their universes are denoted by the same name starting with a standard upper-case letter instead of the calligraphic one; for instance, Ex is the universe of the structure $\mathcal{E}x$. Throughout this paper, we consider only finite structures.

A sentence $\varphi \in \mathrm{FO}^2(\Sigma \cup \{<\})$, where $<$ is a binary relation symbol not belonging to Σ, is said to be **order-invariant** if for every finite Σ-structure $\mathcal{A}$, and every pair of strict linear orders $<_0$ and $<_1$ on A, $(\mathcal{A}, <_0) \models \varphi$ iff $(\mathcal{A}, <_1) \models \varphi$. It is then convenient to omit the interpretation for the symbol $<$, and to write $\mathcal{A} \models \varphi$ iff $(\mathcal{A}, <_0) \models \varphi$ for any (or, equivalently, every) linear order $<_0$. The set of order-invariant sentences using two variables is denoted $<$-inv FO^2.

[a] We have recently become aware [2] of an upcoming result leading us to believe that, in general, $<$-inv FO^2 can capture properties that are not FO-definable. More precisely, this result states the existence of two classes of structures (of unbounded degree) $\mathcal{C}' \subseteq \mathcal{C}$ such that $\mathcal{C}'$ is not definable in FO, but can be defined in $\mathcal{C}$ by an FO^2-sentence using a linear order, which is order-invariant on all structures of $\mathcal{C}$. Although this does not exactly imply $<$-inv $\mathrm{FO}^2 \not\subseteq \mathrm{FO}$ according to our definition of invariance (since said sentence is not invariant on all finite structures), it leads us to believe that $<$-inv FO^2 can express properties beyond FO's reach.

Let $\mathcal{L}, \mathcal{L}'$ be two logics defined over the same signature, and $\mathcal{C}$ be a class of finite structures on this signature. We say that a property $\mathcal{P} \subseteq \mathcal{C}$ is **definable** (or expressible) in $\mathcal{L}$ if there exists an $\mathcal{L}$-sentence φ such that $\mathcal{P} = \{\mathcal{A} \in \mathcal{C} : \mathcal{A} \models \varphi\}$. We say that $\mathcal{L} \subseteq \mathcal{L}'$ on $\mathcal{C}$ if every property on $\mathcal{C}$ definable in $\mathcal{L}$ is also definable in $\mathcal{L}'$.

It is quite clear that $\mathrm{FO}^2 \subseteq <\text{-inv } \mathrm{FO}^2$: any sentence which does not make use of the order is indeed order-invariant. Furthermore, this inclusion is strict. For instance, over the empty signature, the property of having at least three elements is not definable in FO^2 (this can easily be seen with the tools presented in Section 5.1), but can be expressed in $<$-inv FO^2, for instance via the formula $\exists x\ \exists y\ (x < y\ \wedge\ (\exists x\ y < x))$.

The **quantifier rank** of a formula is the maximal number of quantifiers in a branch of its syntactic tree. Given two Σ-structures $\mathcal{A}_0$ and $\mathcal{A}_1$, and $\mathcal{L}$ being one of $\mathrm{FO}, \mathrm{FO}^2$ and $<$-inv FO^2, we write $\mathcal{A}_0 \equiv_k^{\mathcal{L}} \mathcal{A}_1$ if $\mathcal{A}_0$ and $\mathcal{A}_1$ satisfy the same $\mathcal{L}$-sentences of quantifier rank at most k. In this case, we say that $\mathcal{A}_0$ and $\mathcal{A}_1$ are **$\mathcal{L}$-similar at depth k**.

We write $\mathcal{A}_0 \simeq \mathcal{A}_1$ if $\mathcal{A}_0$ and $\mathcal{A}_1$ are isomorphic.

Atomic types

Let a be an element of a structure $\mathcal{A}$. The **atomic type** $\mathrm{tp}^0_{\mathcal{A}}(a)$ of a in $\mathcal{A}$ is the set of atomic formulas φ with at most one free variable x such that $\mathcal{A}, x \mapsto a \models \varphi$.

We define similarly the atomic type $\mathrm{tp}^0_{\mathcal{A}}(a, b)$ of a pair (a, b) of elements of $\mathcal{A}$ as the set of atomic formulas φ with free variables in $\{x, y\}$ such that $\mathcal{A}, x \mapsto a, y \mapsto b \models \varphi$.

Given a linearly ordered Σ-structure $(\mathcal{A}, <)$, $\mathrm{tp}^0_{(\mathcal{A},<)}(a, b)$ can be divided into $\mathrm{tp}^0_{<}(a, b)$ and $\mathrm{tp}^0_{\mathcal{A}}(a, b)$, where $\mathrm{tp}^0_{<}(a, b)$ is one of $\{x < y\}, \{x > y\}$ and $\{x = y\}$.

Gaifman graphs

The **Gaifman graph** $\mathcal{G}_{\mathcal{A}}$ of a structure $\mathcal{A}$ is defined as (A, E) where $(a, b) \in E$ iff a and b appear in the same tuple of a relation of $\mathcal{A}$. Notice that if a graph is seen as a structure on the signature consisting of a single binary relation symbol, its Gaifman graph is none other than the unoriented version of itself.

By $\mathrm{dist}_{\mathcal{A}}(a, b)$, we denote the distance between a and b in $\mathcal{G}_{\mathcal{A}}$. For $B \subseteq A$, we note $N_{\mathcal{A}}(B)$ the set of elements at distance exactly 1 from B in $\mathcal{G}_{\mathcal{A}}$. In particular, $B \cap N_{\mathcal{A}}(B) = \emptyset$.

The **degree** of $\mathcal{A}$ is the maximal degree of its Gaifman graph, and a class $\mathcal{C}$ of Σ-structures is said to have **bounded degree** if there exists some $d \in \mathbb{N}$ such that the degree of every $\mathcal{A} \in \mathcal{C}$ is at most d.

3 Main result

We are now able to state the main result of this article. Remember that $<$-inv FO^2 allows us to express properties that are beyond the scope of plain FO^2. We give an upper bound to its expressive power, when the degree is bounded:

▶ **Theorem 1.** *Let $\mathcal{C}$ be a class of structures of bounded degree.*
Then $<$-inv $FO^2 \subseteq FO$ on $\mathcal{C}$.

For the remainder of this paper, we fix a signature Σ, an integer d and a class $\mathcal{C}$ of Σ-structures of degree at most d.

Let us now show the skeleton of our proof. The technical part of the proof will be the focus of Sections 4 and 5. Our general strategy is to show the existence of a function $f : \mathbb{N} \to \mathbb{N}$ such that every formula $\varphi \in <$-inv FO^2 of quantifier rank k is equivalent on $\mathcal{C}$ (i.e. satisfied by the same structures of $\mathcal{C}$) to an FO-formula ψ of quantifier rank at most $f(k)$.

To prove this, it is enough to show that for any two structures $\mathcal{A}_0, \mathcal{A}_1 \in \mathcal{C}$ such that $\mathcal{A}_0 \equiv^{\mathrm{FO}}_{f(k)} \mathcal{A}_1$, we have $\mathcal{A}_0 \equiv^{<\text{-inv FO}^2}_k \mathcal{A}_1$. Indeed, the class of structures satisfying a formula $\varphi \in <\text{-inv FO}^2$ of quantifier rank k is a union of equivalence classes for the equivalence relation $\equiv^{<\text{-inv FO}^2}_k$, whose intersection with $\mathcal{C}$ is in turn the intersection of $\mathcal{C}$ with a union of equivalence classes for $\equiv^{\mathrm{FO}}_{f(k)}$. It is folklore (see, e.g., [13]) that the equivalence relation $\equiv^{\mathrm{FO}}_{f(k)}$ has finite index, and that each of its equivalence classes is definable by an FO-sentence of quantifier rank $f(k)$. Then ψ is just the finite disjunction of these FO-sentences.

In order to show that $\mathcal{A}_0 \equiv^{<\text{-inv FO}^2}_k \mathcal{A}_1$, we will construct in Section 4 two particular orders $<_0, <_1$ on these respective structures, and we will prove in Section 5 that

$$(\mathcal{A}_0, <_0) \equiv^{\mathrm{FO}^2}_k (\mathcal{A}_1, <_1)\,. \tag{1}$$

This concludes the proof, since any sentence $\theta \in <\text{-inv FO}^2$ with quantifier rank at most k holds in $\mathcal{A}_0$ iff it holds in $(\mathcal{A}_0, <_0)$ (by definition of order-invariance), iff it holds in $(\mathcal{A}_1, <_1)$ (by (1)), iff it holds in $\mathcal{A}_1$.

4 Constructing linear orders on $\mathcal{A}_0$ and $\mathcal{A}_1$

Recall from Section 3 that our goal is to find a function f such that, given two structures $\mathcal{A}_0, \mathcal{A}_1$ in $\mathcal{C}$ such that

$$\mathcal{A}_0 \equiv^{\mathrm{FO}}_{f(k)} \mathcal{A}_1\,, \tag{2}$$

we are able to construct two linear orders $<_0, <_1$ such that $(\mathcal{A}_0, <_0) \equiv^{\mathrm{FO}^2}_k (\mathcal{A}_1, <_1)$.

In this section, we define f and we detail the construction of such orders. The proof of $<$-inv FO-similarity between $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$ will be the focus of Section 5.

Let us now explain how we define f. For that, we need to introduce the notion of neighborhood and neighborhood type. These notions are defined in Section 4.1. We then explain in Section 4.2 how to divide neighborhood types into rare ones and frequent ones. Finally, the details of the construction are given in Section 4.3.

4.1 Neighborhoods

Let us now define the notion of neighborhood of an element in a structure.

Let c be a new constant symbol, and let $\mathcal{A} \in \mathcal{C}$. For $k \in \mathbb{N}$ and $a \in A$, the (pointed) ***k*-neighborhood** $\mathcal{N}^k_{\mathcal{A}}(a)$ of a in $\mathcal{A}$ is the $(\Sigma \cup \{c\})$-structure whose restriction to the vocabulary Σ is the substructure of $\mathcal{A}$ induced by the set $N^k_{\mathcal{A}}(a) = \{b \in A : \mathrm{dist}_{\mathcal{A}}(a,b) \leq k\}$, and where c is interpreted as a. In other words, it consists of all the elements at distance at most k from a in $\mathcal{A}$, together with the relations they share in $\mathcal{A}$; the center a being marked by the constant c. We sometimes refer to $N^k_{\mathcal{A}}(a)$ as the k-neighborhood of a in $\mathcal{A}$ as well, but the context will always make clear whether we refer to the whole substructure or only its domain. The ***k*-neighborhood type** $\tau = \text{neigh-tp}^k_{\mathcal{A}}(a)$ of a in $\mathcal{A}$ is the isomorphism class of its k-neighborhood. We say that τ is a k-neighborhood type over Σ, and that a is an **occurrence** of τ. We denote by $|\mathcal{A}|_\tau$ the number of occurrences of τ in $\mathcal{A}$, and we write $[\![\mathcal{A}_0]\!]_k =^t [\![\mathcal{A}_1]\!]_k$ to mean that for every k-neighborhood type τ, $|\mathcal{A}_0|_\tau$ and $|\mathcal{A}_1|_\tau$ are either equal, or both larger than t.

Let $\textsc{NeighType}^d_k$ denote the set of k-neighborhood types over Σ occurring in structures of degree at most d. Note that $\textsc{NeighType}^d_k$ is a finite set.

The interest of this notion resides in the fact that when the degree is bounded, FO is exactly able to count the number of occurrences of neighborhood types up to some threshold [4]. We will only use one direction of this characterization, namely:

▶ **Proposition 2.** *For all integers k and t, there exists some $\hat{f}(k,t) \in \mathbb{N}$ (which also depends on the bound d on the degree of structures in $\mathcal{C}$) such that for all structures $\mathcal{A}_0, \mathcal{A}_1 \in \mathcal{C}$,*

$$\mathcal{A}_0 \equiv^{FO}_{\hat{f}(k,t)} \mathcal{A}_1 \quad \rightarrow \quad [\![\mathcal{A}_0]\!]_k =^t [\![\mathcal{A}_1]\!]_k .$$

We now exhibit a function $\Theta : \mathbb{N} \to \mathbb{N}$ such that, if $[\![\mathcal{A}_0]\!]_k =^{\Theta(k)} [\![\mathcal{A}_1]\!]_k$, then one can construct $<_0, <_1$ satisfying (1). Proposition 2 then ensures that $f : k \mapsto \hat{f}(k, \Theta(k))$ fits the bill. Let us now explain how the function Θ is chosen.

4.2 Frequency of a neighborhood type

Let us denote $|\textsc{NeighType}^d_k|$ as N.

Recall that every $\mathcal{A} \in \mathcal{C}$ has degree at most d. What this means is that if we consider the set $\textsc{Freq}[\mathcal{A}]_k$ of k-neighborhood types that have enough occurrences in $\mathcal{A}$ (where "enough" will be given a precise meaning later on), each type in $\textsc{Freq}[\mathcal{A}]_k$ must have many occurrences that are scattered across $\mathcal{A}$. Not only that, but we can also make sure that such occurrences are far from all the occurrences of every k-neighborhood type not in $\textsc{Freq}[\mathcal{A}]_k$, which by definition have few occurrences in $\mathcal{A}$. Since the degree is bounded, N is bounded too, which prevents our distinction (which will be formalized later on) between rare neighborhood types and frequent neighborhood types from being circular.

Such a dichotomy is introduced and detailed in [7]; we simply adapt this construction to our needs. In the remainder of this section, we describe this construction at a high level, and leave the technical details (such as the exact bounds) to the reader.

The proof of the following lemma (in the vein of [1]) is straightforward, and relies on the degree boundedness hypothesis. Intuitively, Lemma 3 states that when the degree is bounded, it is not possible for all the elements of large sets to be concentrated in one corner of the structure, thus making it possible to pick elements in each set that are scattered across the structure.

▶ **Lemma 3.** *Given three integers m, δ, s, there exists a threshold $g(m,\delta,s) \in \mathbb{N}$ such that for all $\mathcal{A} \in \mathcal{C}$, all $B \subseteq A$ of size at most s, and all subsets $C_1, \cdots, C_n \subseteq A$ (with $n \leq N$) of size at least $g(m,\delta,s)$, it is possible to find elements $c^1_j, \cdots, c^m_j \in C_j$ for all $j \in \{1, \cdots, n\}$, such that for all $j, j' \in \{1, \cdots, n\}$ and $i, i' \in \{1, \cdots, m\}$, $dist_{\mathcal{A}}(c^i_j, B) > \delta$ and $dist_{\mathcal{A}}(c^i_j, c^{i'}_{j'}) > \delta$ if $(j,i) \neq (j',i')$.*

Note that the N is this lemma could be replaced by any constant.

Our goal is, given a structure $\mathcal{A} \in \mathcal{C}$, to partition the k-neighborhood types into two classes: the frequent types, and the rare types. The property we wish to ensure is that there exist in $\mathcal{A}$ some number m (which will be made precise later on, but only depends on k) of occurrences of each one of the frequent k-neighborhood types which are both

- at distance greater than δ (which, as for m, is a function of k and will be fixed in the following) from one another, and
- at distance greater than δ from every occurrence of a rare k-neighborhood type.

To establish this property, we would like to use Lemma 3, with s being the total number of occurrences of all the rare k-neighborhood types, and $C_1, \cdots, C_n$ being the sets of occurrences of the n distinct frequent k-neighborhood types.

The number N of different k-neighborhood types of degree at most d is bounded by a function of k (as well as Σ and d, which are fixed). Hence, we can proceed according to the following (terminating) algorithm to make the distinction between frequent and rare types:

1. First, let us mark every k-neighborhood type as frequent.
2. Among the types which are currently marked as frequent, let τ be one with the smallest number of occurrences in $\mathcal{A}$.
3. If $|\mathcal{A}|_\tau$ is at least $g(m, \delta, s)$ (g being the function from Lemma 3) where s is the total number of occurrences of all the k-neighborhood types which are currently marked as rare, then we are done and the marking frequent/rare is final. Otherwise, mark τ as rare, and go back to step 2 if there remains at least one frequent k-neighborhood type.

Notice that we can go at most N times through step 2, where N depends only on k. Furthermore, each time we add a type to the set of rare k-neighborhood types, we have the guarantee that this type has few occurrences (namely, less than $g(m, \delta, s)$, where s can be bounded by a function of k).

It is thus apparent that the threshold t such that a k-neighborhood type τ is frequent in $\mathcal{A}$ iff $|\mathcal{A}|_\tau \geq t$ can be bounded by some T depending only on k - importantly, T is the same for all structures of $\mathcal{C}$.

Let us now make the above more formal. For $t \in \mathbb{N}$ and $\mathcal{A} \in \mathcal{C}$, let $\textsc{Freq}[\mathcal{A}]_k^{\geq t} \subseteq \textsc{NeighType}_k^d$ denote the set of k-neighborhood types which have at least t occurrences in $\mathcal{A}$. By applying the procedure presented above, we derive the following lemma:

▶ **Lemma 4.** *Let $k, m, \delta \in \mathbb{N}$. There exists $T \in \mathbb{N}$ such that for every $\mathcal{A} \in \mathcal{C}$, there exists some $t \leq T$ such that*

$$t \geq g(m, \delta, \sum_{\tau \notin \textsc{Freq}[\mathcal{A}]_k^{\geq t}} |\mathcal{A}|_\tau)\,.$$

Let $\textsc{Freq}[\mathcal{A}]_k := \textsc{Freq}[\mathcal{A}]_k^{\geq t}$ for the smallest threshold t given in Lemma 4. Some k-neighborhood type $\tau \in \textsc{NeighType}_k^d$ is said to be **frequent** in $\mathcal{A} \in \mathcal{C}$ if it belongs to $\textsc{Freq}[\mathcal{A}]_k$; that is, if $|\mathcal{A}|_\tau \geq t$. Otherwise, τ is said to be **rare**. With the definition of g in mind, Lemma 4 can then be reformulated as follows: in every structure $\mathcal{A} \in \mathcal{C}$, one can find m occurrences of each frequent k-neighborhood type which are at distance greater than δ from one another and from the set of occurrences of every rare k-neighborhood type.

All that remains is for us to give a value (depending only on k) to the integers m and δ: let $M := \max\{|\tau| : \tau \in \textsc{NeighType}_k^d\}$ (M indeed exists, and is a function of k - recall that the signature Σ and the degree d are assumed to be fixed). Let us consider

$$m := 2 \cdot (k+1) \cdot M! \qquad \text{and} \qquad \delta := 4k\,. \tag{3}$$

We then define $\Theta(k)$ as the integer T provided by Lemma 4 for these values of m and δ. The threshold $\Theta(k)$ indeed only depends on k. Finally, notice that if $[\![\mathcal{A}_0]\!]_k =^{\Theta(k)} [\![\mathcal{A}_1]\!]_k$, then $\textsc{Freq}[\mathcal{A}_0]_k = \textsc{Freq}[\mathcal{A}_1]_k$.

As discussed in Section 4.1, there exists a function f such that $\mathcal{A}_0 \equiv^{\text{FO}}_{f(k)} \mathcal{A}_1$ entails $[\![\mathcal{A}_0]\!]_k =^{\Theta(k)} [\![\mathcal{A}_1]\!]_k$. We also make sure that $f(k) \geq \Theta(k) \cdot N + 1$ for every k.

Let us now consider $\mathcal{A}_0, \mathcal{A}_1 \in \mathcal{C}$ such that $\mathcal{A}_0 \equiv^{\text{FO}}_{f(k)} \mathcal{A}_1$ for such an f. If $\textsc{Freq}[\mathcal{A}_0]_k = \emptyset$, then $|\mathcal{A}_0| \leq \Theta(k) \cdot N$. This guarantees that $\mathcal{A}_0 \simeq \mathcal{A}_1$, and in particular that $\mathcal{A}_0 \equiv_k^{<\text{-inv FO}^2} \mathcal{A}_1$. From now on, we suppose that there is at least one frequent k-neighborhood type.

The construction of two linear orders $<_0$ and $<_1$ satisfying $(\mathcal{A}_0, <_0) \equiv_k^{\text{FO}^2} (\mathcal{A}_1, <_1)$ is the object of Section 4.3.

4.3 Construction of $<_0$ and $<_1$

This section is dedicated to the definition of two linear orders $<_0, <_1$ on $\mathcal{A}_0, \mathcal{A}_1 \in \mathcal{C}$. We then prove in Section 5 that $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$ are FO^2-similar at depth k.

Recall that by hypothesis, $\mathcal{A}_0$ and $\mathcal{A}_1$ are FO-similar at depth $f(k)$, which entails that they have the same number of occurrences of each $\tau \in \textsc{NeighType}_k^d$ up to a threshold $\Theta(k)$.

To construct our two linear orders, we need to define the notion of k-environment: given $\mathcal{A} \in \mathcal{C}$, a linear order $<$ on A, $k \in \mathbb{N}$ and an element $a \in A$, we define the ***k*-environment $\mathcal{E}\text{nv}^k_{(\mathcal{A},<)}(a)$ of a in $(\mathcal{A}, <)$** as the restriction of $(\mathcal{A}, <)$ to the k-neighborhood of a in $\mathcal{A}$, where a is the interpretation of the constant symbol c. Note that the order is not taken into account when determining the domain of the substructure (it would otherwise be A, given that any two distinct elements are adjacent for $<$). The ***k*-environment type** $\text{env-tp}^k_{(\mathcal{A},<)}(a)$ is the isomorphism class of $\mathcal{E}\text{nv}^k_{(\mathcal{A},<)}(a)$. In other words, $\text{env-tp}^k_{(\mathcal{A},<)}(a)$ contains the information of $\mathcal{N}^k_{\mathcal{A}}(a)$ together with the order of its elements in $(\mathcal{A}, <)$.

Given $\tau \in \textsc{NeighType}_k^d$, we define $\textsc{Env}(\tau)$ as the set of k-environment types whose underlying k-neighborhood type is τ.

For $i \in \{0, 1\}$, we aim to partition A_i into $2(2k + 1) + 2$ segments:

$$A_i = X_i \cup \bigcup_{j=0}^{2k} (L_i^j \cup R_i^j) \cup M_i \,.$$

Once we have set a linear order on each segment, the linear order $<_i$ on A_i will result from the concatenation of the orders on the segments as follows:

$$(A_i, <_i) := X_i \cdot L_i^0 \cdot L_i^1 \cdots L_i^{2k} \cdot M_i \cdot R_i^{2k} \cdots R_i^1 \cdot R_i^0 \,.$$

Each segment L_i^j, for $j \in \{0, \cdots, 2k\}$ is itself decomposed into two segments $NL_i^j \cdot UL_i^j$. The UL_i^j for $j \in \{k + 1, \cdots, 2k\}$ will be empty; they are defined solely in order to keep the notations uniform. The 'N' stands for "neighbor" and the 'U' for "universal", for reasons that will soon become apparent. Symmetrically, each R_i^j is decomposed into $UR_i^j \cdot NR_i^j$, with empty UR_i^j as soon as $j \geq k + 1$.

For $i \in \{0, 1\}$ and $r \in \{0, \cdots, 2k\}$, we define S_i^r as

$$S_i^r := X_i \cup \bigcup_{j=0}^{r} (L_i^j \cup R_i^j) \,.$$

Let us now explain how the segments are constructed in $\mathcal{A}_0$; see Figure 1 for an illustration.

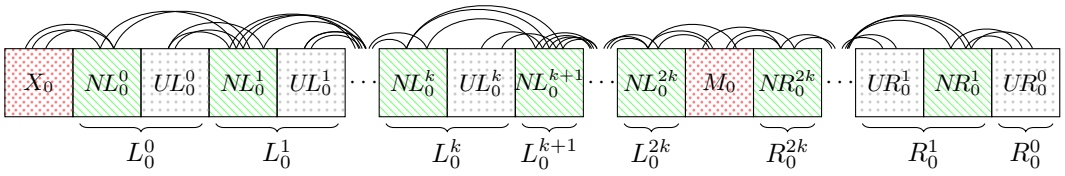


Figure 1 The black curvy edges represent the edges between elements belonging to different segments. Edges between elements of the same segment are not represented here. The order $<_0$ grows from the left to the right.

For every $\tau \in \textsc{Freq}[\mathcal{A}_0]_k$, let $\tau_1, \cdots, \tau_{|\textsc{Env}(\tau)|}$ be an enumeration of $\textsc{Env}(\tau)$. Recall that we defined M in Section 4.2 as $\max\{|\tau| : \tau \in \textsc{NeighType}_k^d\}$. Thus, we have $|\textsc{Env}(\tau)| \leq M!$ for every $\tau \in \textsc{NeighType}_k^d$.

In particular, by definition of frequency, and by choice of m and δ in (3), Lemma 4 ensures that we are able to pick, for every $\tau \in \textsc{Freq}[\mathcal{A}_0]_k$, every $l \in \{1, \cdots, |\textsc{Env}(\tau)|\}$ and every $j \in \{0, \cdots, k\}$, two elements $a[\tau_l]_L^j$ and $a[\tau_l]_R^j$ which have τ as k-neighborhood type in $\mathcal{A}_0$, such that all the $a[\tau_l]_*^j$ are at distance at least $4k + 1$ from each other and from any occurrence of a rare k-neighborhood type in $\mathcal{A}_0$.

Construction of X_0 and NL_0^0

We start with the set X_0, which contains all the occurrences of rare k-neighborhood types, together with their k-neighborhoods.
Formally, the domain of X_0 is $\bigcup_{a\in A_0:\ \text{neigh-tp}^k_{\mathcal{A}_0}(a)\notin \text{FREQ}[\mathcal{A}_0]_k} N^k_{\mathcal{A}_0}(a)$.
We set $NL_0^0 := N_{\mathcal{A}_0}(X_0)$ (the set of neighbors of elements of X_0), and define the order $<_0$ on X_0 and on NL_0^0 in an arbitrary way.

Construction of UL_0^j

If $k < j \leq 2k$, then we set $UL_0^j := \emptyset$. Otherwise, for $j \in \{0, \cdots, k\}$, once we have constructed $L_0^0, \cdots, L_0^{j-1}$ and NL_0^j, we construct UL_0^j as follows.
The elements of UL_0^j are $\bigcup_{\tau\in \text{FREQ}[\mathcal{A}_0]_k} \bigcup_{l=1}^{|\text{ENV}(\tau)|} N^k_{\mathcal{A}_0}(a[\tau_l]^j_L)$.
Note that UL_0^j does not intersect the previously constructed segments, by choice of the $a[\tau_l]^j_L$ and of $\delta = 4k$ in (3). Furthermore, the $N^k_{\mathcal{A}_0}(a[\tau_l]^j_L)$ are pairwise disjoint, hence we can fix $<_0$ freely and independently on each of them. Unsurprisingly, we order each $N^k_{\mathcal{A}_0}(a[\tau_l]^j_L)$ so that $\text{env-tp}^k_{(\mathcal{A}_0,<_0)}(a[\tau_l]^j_L) = \tau_l$. This is possible because for every $\tau \in \text{FREQ}[\mathcal{A}_0]_k$ and each l, $\text{neigh-tp}^k_{\mathcal{A}_0}(a[\tau_l]^j_L) = \tau$ by choice of $a[\tau_l]^j_L$.
Once each $N^k_{\mathcal{A}_0}(a[\tau_l]^j_L)$ is ordered according to τ_l, the linear order $<_0$ on UL_0^j can be completed in an arbitrary way. Note that every possible k-environment type extending a frequent k-neighborhood type in $\mathcal{A}_0$ occurs in each UL_0^j. The UL_0^j are *universal* in that sense.

Construction of NL_0^j

Now, let us see how the NL_0^j are constructed. For $j \in \{1, \cdots, 2k\}$, suppose that we have constructed $L_0^0, \cdots, L_0^{j-1}$. The domain of NL_0^j consists of all the neighbors (in $\mathcal{A}_0$) of the elements of L_0^{j-1} not already belonging to the construction so far. Formally, $N_{\mathcal{A}_0}(L_0^{j-1}) \setminus \left(X_0 \cup \bigcup_{m=0}^{j-2} L_0^m\right)$.
The order $<_0$ on NL_0^j is chosen arbitrarily.

Construction of R_0^j

We construct similarly the R_0^j, for $j \in \{0, \cdots, 2k\}$, starting with $NR_0^0 := \emptyset$, then UR_0^0 which contains each $a[\tau_l]^0_R$ together with its k-neighborhood in $\mathcal{A}_0$ ordered according to τ_l, then $NR_0^1 := N_{\mathcal{A}_0}(R_0^0)$, then UR_0^1, etc.
Note that the $a[\tau_l]^j_R$ have been chosen so that they are far enough in $\mathcal{A}_0$ from all the segments that have been constructed so far, allowing us once more to order their k-neighborhood in $\mathcal{A}_0$ as we see fit.

Construction of M_0

M_0 contains all the elements of A_0 besides those already belonging to S_0^{2k}. The order $<_0$ chosen on M_0 is arbitrary.

Transfer on $\mathcal{A}_1$

Suppose that we have constructed S_0^{2k}. We can make sure, retrospectively, that the index $f(k)$ in (2) is large enough so that there exists a set $S \subseteq A_1$ so that $\mathcal{A}_0|_{S_0^{2k}\cup N_{\mathcal{A}_0}(S_0^{2k})} \simeq \mathcal{A}_1|_S$ (this is ensured as long as $f(k) \geq |S_0^{2k} \cup N_{\mathcal{A}_0}(S_0^{2k})| + 1$, which can be bounded by a function of k, independent of $\mathcal{A}_0$ and $\mathcal{A}_1$).

Let $\varphi_0 : \mathcal{A}_0|_{S_0^{2k}} \to \mathcal{A}_1|_{S'}$ be the restriction to S_0^{2k} of said isomorphism, and let φ_1 be its converse. By construction, the k-neighborhood of every $a \in S_0^k$ is included in S_0^{2k}; hence every such a has the same k-neighborhood type in $\mathcal{A}_0$ as has $\varphi_0(a)$ in $\mathcal{A}_1$.
We transfer alongside φ_0 all the segments, with their order, from $(\mathcal{A}_0, <_0)$ to $\mathcal{A}_1$, thus defining $X_1, NL_1^j, UL_1^j, \cdots$ as the respective images by φ_0 of $X_0, NL_0^j, UL_0^j, \cdots$, and define M_1 as the counterpart to M_0. Note that the properties concerning neighborhood are transferred; e.g. all the neighbors of an element in L_1^j, $1 \leq j < 2k$, belong to $L_1^{j-1} \cup L_1^j \cup L_1^{j+1}$.

By construction, we get the following lemma:

▶ **Lemma 5.** *For each $a \in S_0^k$, we have $\text{env-tp}^k_{(\mathcal{A}_0,<_0)}(a) = \text{env-tp}^k_{(\mathcal{A}_1,<_1)}(\varphi_0(a))$.*

Lemma 5 has two immediate consequences:

- The set X_1 contains the occurrences in $\mathcal{A}_1$ of all the rare k-neighborhood types (just forget about the order on the k-environments, and remember that $\mathcal{A}_0$ and $\mathcal{A}_1$ have the same number of occurrences of each rare k-neighborhood type).
- All the universal segments UL_1^j and UR_1^j, for $0 \leq j \leq k$, contain at least one occurrence of each environment in $\textsc{Env}(\tau)$, for each $\tau \in \textsc{Freq}[\mathcal{A}_0]_k$.

Our construction also guarantees the following result:

▶ **Lemma 6.** *For each $a, b \in S_0^k$, we have $tp^0_{(\mathcal{A}_0,<_0)}(a,b) = tp^0_{(\mathcal{A}_1,<_1)}(\varphi_0(a), \varphi_0(b))$.*

In particular, for $a = b \in S_0^k$, we have $\text{tp}^0_{(\mathcal{A}_0,<_0)}(a) = \text{tp}^0_{(\mathcal{A}_1,<_1)}(\varphi_0(a))$.

5 Proof of the FO²-similarity of $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$

In this section, we aim to show the following result:

▶ **Proposition 7.** *We have that $(\mathcal{A}_0, <_0) \equiv_k^{FO^2} (\mathcal{A}_1, <_1)$.*

5.1 The two-pebble Ehrenfeucht-Fraïssé game

To establish Proposition 7, we use Ehrenfeucht-Fraïssé games with two pebbles. These games have been introduced by Immerman and Kozen [11]. Let us adapt their definition to our context.

The **k-round two-pebble Ehrenfeucht-Fraïssé game on $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$** is played by two players: the spoiler and the duplicator. The spoiler tries to expose differences between the two structures, while the duplicator tries to establish their indistinguishability.

There are two pebbles associated with each structure: p_0^x and p_0^y on $(\mathcal{A}_0, <_0)$, and p_1^x and p_1^y on $(\mathcal{A}_1, <_1)$. Formally, these pebbles can be seen as the interpretations in each structure of two new constant symbols, but it will be convenient to see them as moving pieces.

At the start of the game, the duplicator places p_0^x and p_0^y on elements of $(\mathcal{A}_0, <_0)$, and p_1^x and p_1^y on elements of $(\mathcal{A}_1, <_1)$. The spoiler wins if the duplicator is unable to ensure that $\text{tp}^0_{(\mathcal{A}_0,<_0)}(p_0^x, p_0^y) = \text{tp}^0_{(\mathcal{A}_1,<_1)}(p_1^x, p_1^y)$. Otherwise, the proper game starts. Note that in the usual definition of the starting position, the pebbles are not on the board; however, it will be convenient to have them placed in order to uniformize our invariant. This change is not profound and does not affect the properties of the game.

For each of the k rounds, the spoiler starts by choosing a structure and a pebble in this structure, and places this pebble on a element of the chosen structure. In turn, the duplicator must place the corresponding pebble in the other structure on an element of that structure. The spoiler wins at once if $\text{tp}^0_{(\mathcal{A}_0,<_0)}(p_0^x, p_0^y) \neq \text{tp}^0_{(\mathcal{A}_1,<_1)}(p_1^x, p_1^y)$. Otherwise, another round is played. If the spoiler has not won after k rounds, then the duplicator wins.

The main interest of these games is that they capture the expressive power of FO^2 [11]. We will only need the fact that these games are correct:

▶ **Theorem 8.** *If the duplicator has a winning strategy in the k-round two-pebble Ehrenfeucht-Fraïssé game on $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$, then $(\mathcal{A}_0, <_0) \equiv_k^{FO^2} (\mathcal{A}_1, <_1)$.*

Thus, in order to prove Proposition 7, we show that the duplicator wins the k-round two-pebble Ehrenfeucht-Fraïssé game on $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$. For that, let us show by a decreasing induction on $r = k, \cdots, 0$ that the duplicator can ensure, after $k - r$ rounds, that the three following properties (described below) hold:

$$\forall i \in \{0,1\}, \forall \alpha \in \{x,y\},\ p_i^\alpha \in S_i^r \ \rightarrow\ p_{1-i}^\alpha = \varphi_i(p_i^\alpha) \tag{S_r}$$

$$\forall \alpha \in \{x,y\},\ \text{env-tp}^r_{(\mathcal{A}_0,<_0)}(p_0^\alpha) = \text{env-tp}^r_{(\mathcal{A}_1,<_1)}(p_1^\alpha) \tag{E_r}$$

$$\text{tp}^0_{(\mathcal{A}_0,<_0)}(p_0^x, p_0^y) = \text{tp}^0_{(\mathcal{A}_1,<_1)}(p_1^x, p_1^y) \tag{T_r}$$

The first property, (S_r), guarantees that if a pebble is close (in a sense that depends on the number of rounds left in the game) to one of the $<_i$-minimal or $<_i$-maximal elements, the corresponding pebble in the other structure is located at the same position with respect to this $<_i$-extremal element.
As for (E_r), it states that two corresponding pebbles are always placed on elements sharing the same r-environment type. Once again, the satefy distance decreases with each round that goes.
Finally, (T_r) controls that both pebbles have the same relative position (both with respect to the order and the original vocabulary) in the two ordered structures. In particular, the duplicator wins the game if (T_r) is satisfied at the begining of the game, and after each of the k rounds of the game.

5.2 Base case: proofs of (S_k), (E_k) and (T_k)

We start by proving (S_k), (E_k) and (T_k).
At the start of the game, the duplicator places both p_0^x and p_0^y on the $<_0$-minimal element of $(\mathcal{A}_0, <_0)$, and both p_1^x and p_1^y on the $<_1$-minimal element of $(\mathcal{A}_1, <_1)$. In particular,

$$p_1^x = p_1^y = \varphi_0(p_0^x) = \varphi_0(p_0^y)\,.$$

This ensures that (S_k) holds, while (E_k) and (T_k) respectively follow from Lemma 5 and Lemma 6.

5.3 Strategy for the duplicator

We now describe the duplicator's strategy to ensure that (S_r), (E_r) and (T_r) hold no matter how the spoiler plays.

Suppose that we have (S_{r+1}), (E_{r+1}) and (T_{r+1}) for some $0 \leq r < k$, after $k - r - 1$ rounds of the game. Without loss of generality, we may assume that, in the $(k-r)$-th round of the Ehrenfeucht-Fraïssé game between $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$, the spoiler moves p_0^x in $(\mathcal{A}_0, <_0)$. Let us first explain informally the general idea behind the duplicator's strategy.

1. If the spoiler plays around the endpoints (by which we mean the elements that are $<_i$-minimal and maximal), the duplicator has no choice but to play a tit-for-tat strategy, i.e. to respond to the placement of p_i^α near the endpoints by moving p_{1-i}^α on $\varphi_i(p_i^\alpha)$.

If the duplicator does not respond this way, then the spoiler will be able to expose the difference between $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$ in the subsequent moves, by forcing the duplicator to play closer and closer to the endpoint, which will prove to be impossible at some point.

On top of that, the occurrences of rare neighborhood types are located in $(\mathcal{A}_i, <_i)$ near the $<_i$-minimal element. If the duplicator does not play according to φ_0 in this area, it will be easy enough for the spoiler to win the game.

The reason we introduced the segments NL_i^j, UL_i^j, NR_i^j and UR_i^j is precisely to bound the area in which the duplicator must implement the tit-for-tat strategy. Indeed, as soon as a pebble is placed in M_i, there is no way for the spoiler to join the endpoints in less than k moves while forcing the duplicator's hand.

The case where the spoiler plays near the endpoints corresponds to Case (I) below, and is detailed in Section 5.4.

2. Next, suppose that the spoiler places a pebble, say p_0^x, next (in $\mathcal{A}_0$) to p_0^y, i.e. such that $p_0^x \in N^1_{\mathcal{A}_0}(p_0^y)$. The duplicator must place p_1^x on an element whose relative position to p_1^y is the same as the relative position of p_0^x with respect to p_0^y. Note that once this is done, the spoiler can change variable, and place p_0^y (or p_1^y, if they decide to play in $(\mathcal{A}_1, <_1)$) in $N^1_{\mathcal{A}_0}(p_0^x)$, thus forcing the duplicator to play near p_1^x. In order to prevent the spoiler from being able, in k such moves, to expose the difference between $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$, the duplicator must make sure, with r rounds left, that p_0^x and p_1^x (as well as p_0^y and p_1^y) share the same r-environment in $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$. This will guarantee that the duplicator can play along if the spoiler decides to take r moves adjacent (in $\mathcal{A}_i$) to one another.

   The case where the spoiler places a pebble next (in the structure without ordering) to the other pebble is our Case (II), and is treated in Section 5.5.

3. Suppose now that the spoiler's move does not fall under the previous templates. Let us assume that the spoiler plays in $(\mathcal{A}_0, <_0)$, and moves p_0^x to the left of p_0^y (i.e. such that $(\mathcal{A}_0, <_0) \models p_0^x < p_0^y$).

   In order to play according to the remarks from Cases 1 and 2, the duplicator must place p_1^x on an element which shares the same r-environment with p_0^x (where r is the number of rounds left in the game), which is not near the endpoints.

   It must be the case that the k-neighborhood type of p_0^x in $\mathcal{A}_0$ is frequent, since it is not near the endpoints of $(\mathcal{A}_0, <_0)$, hence not in X_0. By construction, every universal segment UL_1^j, for $0 \leq j \leq k$, contains elements of each k-environment type extending any frequent k-neighborhood type. In particular, it contains an element having the same r-environment as p_0^x. The duplicator will place p_1^x on such an element in the leftmost segment UL_1^j which is not considered to be near the endpoints (this notion depends on the number r of rounds left in the game). This is detailed in Cases (III) and (V) (for the symmetrical case where p_0^x is placed to the right of p_0^y) below.

   However, we have to consider a subcase, where p_1^y is itself in the leftmost segment L_1^j which is not near the endpoints. Indeed, in this case, placing p_1^x as discussed may result in p_1^x being to the right of p_1^y, or being in $N^1_{\mathcal{A}_1}(p_1^y)$; either of which being game-losing to the duplicator. However, since p_1^y was considered to be near the endpoints in the previous round of the game, we know that the duplicator played a tit-for-tat strategy at that point, which allows us to replicate the placement of p_0^x according to φ_0. This subcase, as well as the equivalent subcase where the spoiler places p_0^x to the right of p_0^y, are formalized in Cases (IV) and (VI) below.

We are now ready to describe formally the strategy implemented by the duplicator:

(I) If $p_0^x \in S_0^r$, then the duplicator responds by placing p_1^x on $\varphi_0(p_0^x)$.
This corresponds to the tit-for-tat strategy implemented when the spoiler plays near the endpoints, as discussed in Case 1.

(II) Else, if $p_0^x \notin S_0^r$, and $p_0^x \in N^1_{\mathcal{A}_0}(p_0^y)$, then (E_{r+1}) ensures that there exists an isomorphism $\psi : \mathcal{E}\mathrm{nv}^{r+1}_{(\mathcal{A}_0,<_0)}(p_0^y) \to \mathcal{E}\mathrm{nv}^{r+1}_{(\mathcal{A}_1,<_1)}(p_1^y)$. The duplicator responds by placing p_1^x on $\psi(p_0^x)$.
This makes formal the duplicator's response to a move next to the other pebble, as discussed in Case 2 above.

(III) Else suppose that $(\mathcal{A}_0, <_0) \models p_0^x < p_0^y$ and $p_0^y \notin L_0^{r+1}$. Note that $\tau := \text{neigh-tp}^k_{\mathcal{A}_0}(p_0^x) \in \textsc{Freq}[\mathcal{A}_0]_k$, since $p_0^x \notin X_0$. Let $\tau_l := \text{env-tp}^k_{(\mathcal{A}_0,<_0)}(p_0^x)$.
The duplicator responds by placing p_1^x on $\varphi_0(a[\tau_l]_L^{r+1})$.

(IV) Else, if $(\mathcal{A}_0, <_0) \models p_0^x < p_0^y$ and $p_0^y \in L_0^{r+1}$, then the duplicator moves p_1^x on $\varphi_0(p_0^x)$ (by (S_{r+1}), p_0^x indeed belongs to the domain of φ_0).

(V) Else, suppose that $(\mathcal{A}_0, <_0) \models p_0^y < p_0^x$ and $p_0^y \notin R_0^{r+1}$. This case is symmetric to Case (III).
Similarly, the duplicator opts to play p_1^x on $\varphi_0(a[\tau_l]_R^{r+1})$, where $\tau_l := \text{env-tp}^k_{(\mathcal{A}_0,<_0)}(p_0^x)$.

(VI) If we are in none of the cases above, it means that the spoiler has placed p_0^x to the right of p_0^y, and that $p_0^y \in R_0^{r+1}$. This case is symmetric to Case (IV).
Once again, the duplicator places p_1^x on $\varphi_0(p_0^x)$.

It remains to show that this strategy satisfies our invariants: under the inductive assumption that (S_{r+1}), (E_{r+1}) and (T_{r+1}) hold, for some $0 \leq r < k$, we need to show that this strategy ensures that (S_r), (E_r) and (T_r) hold.

We treat each case in its own section: Section 5.4 is devoted to Case (I) while Section 5.5 covers Case (II). Both Cases (III) and (IV) are treated in Section 5.6. Cases (V) and (VI), being their exact symmetric counterparts, are left to the reader.

▶ Note 9. Note that some properties need no verification. Since p_0^y and p_1^y are left untouched by the players, (S_{r+1}) ensures that half of (S_r) automatically holds, namely that

$$\forall i \in \{0,1\}, \quad p_i^y \in S_i^r \quad \to \quad p_{1-i}^y = \varphi_i(p_i^y)\,.$$

Similarly, the part of (E_r) concerning p_0^y and p_1^y follows from (E_{r+1}):

$$\text{env-tp}^r_{(\mathcal{A}_0,<_0)}(p_0^y) = \text{env-tp}^r_{(\mathcal{A}_1,<_1)}(p_1^y)\,.$$

Lastly, notice that once we have shown that (E_r) holds, it follows that

$$\begin{cases} \text{tp}^0_{\mathcal{A}_0}(p_0^x) = \text{tp}^0_{\mathcal{A}_1}(p_1^x) \\ \text{tp}^0_{\mathcal{A}_0}(p_0^y) = \text{tp}^0_{\mathcal{A}_1}(p_1^y) \end{cases}$$

5.4 When the spoiler plays near the endpoints: Case (I)

In this section, we treat the case where the spoiler places p_0^x near the $<_0$-minimal or $<_0$-maximal element of $(\mathcal{A}_0, <_0)$. Obviously, what "near" means depends on the number of rounds left in the game; the more rounds remain, the more the duplicator must be cautious regarding the possibility for the spoiler to reach an endpoint and potentially expose a difference between $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$.

As we have stated in Case (I), with r rounds left, we consider a move on p_0^x by the spoiler to be near the endpoints if it is made in S_0^r. In that case, the duplicator responds along the tit-for-tat strategy, namely by placing p_1^x on $\varphi_0(p_0^x)$.

Let us now prove that this strategy guarantees that (S_r), (E_r) and (T_r) hold. Recall from Note 9 that part of the task is already taken care of.

Proof of (S_r) in Case (I)

We have to show that $\forall i \in \{0,1\},\ p_i^x \in S_i^r \ \rightarrow\ p_{1-i}^x = \varphi_i(p_i^x)$. This follows directly from the duplicator's strategy, since $p_1^x = \varphi_0(p_0^x)$ (thus $p_0^x = \varphi_1(p_1^x)$).

Proof of (E_r) in Case (I)

We need to prove that $\text{env-tp}^r_{(\mathcal{A}_0,<_0)}(p_0^x) = \text{env-tp}^r_{(\mathcal{A}_1,<_1)}(p_1^x)$, which is a consequence of Lemma 5 given that $p_1^x = \varphi_0(p_0^x)$ and $r < k$.

Proof of (T_r) in Case (I)

First, suppose that $p_0^y \in S_0^{r+1}$. By (S_{r+1}), we know that $p_1^y = \varphi_0(p_0^y)$. Thus, Lemma 6 allows us to conclude that $\text{tp}^0_{(\mathcal{A}_0,<_0)}(p_0^x, p_0^y) = \text{tp}^0_{(\mathcal{A}_1,<_1)}(p_1^x, p_1^y)$.

Otherwise, $p_0^y \notin S_0^{r+1}$ and (S_{r+1}) entails that $p_1^y \notin S_1^{r+1}$.
We have two points to establish:

$$\text{tp}^0_{\mathcal{A}_0}(p_0^x, p_0^y) = \text{tp}^0_{\mathcal{A}_1}(p_1^x, p_1^y) \tag{4}$$
$$\text{tp}^0_{<_0}(p_0^x, p_0^y) = \text{tp}^0_{<_1}(p_1^x, p_1^y) \tag{5}$$

Notice that

$$\begin{cases} \text{tp}^0_{\mathcal{A}_0}(p_0^x, p_0^y) = \text{tp}^0_{\mathcal{A}_0}(p_0^x) \cup \text{tp}^0_{\mathcal{A}_0}(p_0^y) \\ \text{tp}^0_{\mathcal{A}_1}(p_1^x, p_1^y) = \text{tp}^0_{\mathcal{A}_1}(p_1^x) \cup \text{tp}^0_{\mathcal{A}_1}(p_1^y) \end{cases}$$

This is because, by construction, the neighbors in $\mathcal{A}_i$ of an element of S_i^r all belong to S_i^{r+1}. Equation (4) follows from this remark and Note 9.
As for Equation (5), either

$$p_0^x \in X_0 \cup \bigcup_{0\leq j\leq r} L_0^j \quad \text{and} \quad p_1^x \in X_1 \cup \bigcup_{0\leq j\leq r} L_1^j\,,$$

in which case $\text{tp}^0_{<_0}(p_0^x, p_0^y) = \{x < y\} = \text{tp}^0_{<_1}(p_1^x, p_1^y)$, or

$$p_0^x \in \bigcup_{0\leq j\leq r} R_0^j \quad \text{and} \quad p_1^x \in \bigcup_{0\leq j\leq r} R_1^j\,,$$

in which case $\text{tp}^0_{<_0}(p_0^x, p_0^y) = \{x > y\} = \text{tp}^0_{<_1}(p_1^x, p_1^y)$.

5.5 When the spoiler plays next to the other pebble: Case (II)

Suppose now that the spoiler places p_0^x next to the other pebble in $\mathcal{A}_0$ (i.e. $p_0^x \in N^1_{\mathcal{A}_0}(p_0^y)$), but not in S_0^r (for that move would fall under the jurisdiction of Case (I)). In that case, the duplicator must place p_1^x so that the relative position of p_1^x and p_1^y is the same as that of p_0^x and p_0^y.

For that, we can use (E_{r+1}), which guarantees that $\text{env-tp}^{r+1}_{(\mathcal{A}_0,<_0)}(p_0^y) = \text{env-tp}^{r+1}_{(\mathcal{A}_1,<_1)}(p_1^y)$. Thus there exists an isomorphism ψ between $\mathcal{E}\text{nv}^{r+1}_{(\mathcal{A}_0,<_0)}(p_0^y)$ and $\mathcal{E}\text{nv}^{r+1}_{(\mathcal{A}_1,<_1)}(p_1^y)$. Note that this isomorphism is unique, by virtue of $<_0$ and $<_1$ being linear orders.
The duplicator's response is to place p_1^x on $\psi(p_0^x)$. Let us now prove that this strategy is correct with respect to our invariants (S_r), (E_r) and (T_r).

Proof of (S_r) in Case (II)

Because the spoiler's move does not fall under Case (I), we know that $p_0^x \notin S_0^r$.
Let us now show that p_1^x is not near the endpoints either: suppose that $p_1^x \in S_1^r$. By construction, since p_1^x and p_1^y are neighbors in $\mathcal{A}_1$, this entails that $p_1^y \in S_1^{r+1}$. But then, we know by (S_{r+1}) that $p_0^y = \varphi_1(p_1^y)$; and because ψ is the unique isomorphism between $\mathcal{E}\mathrm{nv}^{r+1}_{(\mathcal{A}_0,<_0)}(p_0^y)$ and $\mathcal{E}\mathrm{nv}^{r+1}_{(\mathcal{A}_1,<_1)}(p_1^y)$, ψ is equal to the restriction $\widetilde{\varphi_0}$ of φ_0:

$$\widetilde{\varphi_0} \,:\, \mathcal{E}\mathrm{nv}^{r+1}_{(\mathcal{A}_0,<_0)}(p_0^y) \;\to\; \mathcal{E}\mathrm{nv}^{r+1}_{(\mathcal{A}_1,<_1)}(p_1^y)\,.$$

Thus $p_0^x = \psi^{-1}(p_1^x) = \widetilde{\varphi_0}^{-1}(p_1^x) = \varphi_1(p_1^x)$, and by definition of the segments on $(\mathcal{A}_1, <_1)$, which are just a transposition of the segments of $(\mathcal{A}_0, <_0)$ via φ_0, $p_1^x \in S_1^r$ then entails that $p_0^x \in S_0^r$, which is absurd.
Since we neither have $p_0^x \in S_0^r$ nor $p_1^x \in S_1^r$, (S_r) holds - recall from Note 9 that the part concerning p_0^y and p_1^y is always satisfied.

Proof of (E_r) in Case (II)

Recall that the duplicator placed p_1^x on the image of p_0^x by the isomorphism

$$\psi : \mathcal{E}\mathrm{nv}^{r+1}_{(\mathcal{A}_0,<_0)}(p_0^y) \to \mathcal{E}\mathrm{nv}^{r+1}_{(\mathcal{A}_1,<_1)}(p_1^y)\,.$$

It is easy to check that the restriction $\widetilde{\psi}$ of ψ: $\widetilde{\psi} \,:\, \mathcal{E}\mathrm{nv}^{r}_{(\mathcal{A}_0,<_0)}(p_0^x) \;\to\; \mathcal{E}\mathrm{nv}^{r}_{(\mathcal{A}_1,<_1)}(p_1^x)$ is well defined, and is indeed an isomorphism.
This ensures that $\text{env-tp}^r_{(\mathcal{A}_0,<_0)}(p_0^x) = \text{env-tp}^r_{(\mathcal{A}_1,<_1)}(p_1^x)$, thus completing the proof of (E_r).

Proof of (T_r) in Case (II)

This follows immediately from the fact that the isomorphism ψ maps p_0^x to p_1^x and p_0^y to p_1^y: all the atomic facts about these elements are preserved.

5.6 When the spoiler plays to the left: Cases (III) and (IV)

We now treat our last case, which covers both Cases (III) and (IV), i.e. the instances where the spoiler places p_0^x to the left of p_0^y (formally: such that $(\mathcal{A}_0, <_0) \models p_0^x < p_0^y$), which do not already fall in Cases (I) and (II).
Note that the scenario in which the spoiler plays to the right of the other pebble is the exact symmetric of this one (since the X_i play no role in this case, left and right can be interchanged harmlessly).

The idea here is very simple: since the spoiler has placed p_0^x to the left of p_0^y, but neither in S_0^r nor in $N^1_{\mathcal{A}_0}(p_0^y)$, the duplicator responds by placing p_1^x on an element of UL_1^{r+1} (the leftmost universal segment not in S_1^r) sharing the same k-environment. This is possible by construction of the universal segments: if $\tau_l := \text{env-tp}^k_{(\mathcal{A}_0,<_0)}(p_0^x)$ (which must extend a frequent k-neighborhood type, since $p_0^x \notin X_0$), then $\varphi_0(a[\tau_l]_L^{r+1})$ satisfies the requirements.

There is one caveat to this strategy. If p_1^y is itself in L_1^{r+1}, two problems may arise: first, it is possible for p_1^x and p_1^y to be in the wrong order (i.e. such that $(\mathcal{A}_1, <_1) \models p_1^x > p_1^y$). Second, it may be the case that p_1^x and p_1^y are neighbors in $\mathcal{A}_1$, which, together with the fact that p_0^x and p_0^y are orthogonal in $\mathcal{A}_0$ (i.e. $\mathrm{tp}^0_{\mathcal{A}_0}(p_0^x, p_0^y) = \mathrm{tp}^0_{\mathcal{A}_0}(p_0^x) \cup \mathrm{tp}^0_{\mathcal{A}_0}(p_0^y)$), would break (T_r).
This is why the duplicator's strategy depends on whether $p_1^y \in L_1^{r+1}$:

- if this is not the case, then the duplicator places p_1^x on $\varphi_0(a[\tau_l]_L^{r+1})$. This corresponds to Case (III).
- if $p_1^y \in L_1^{r+1}$, then (S_{r+1}) guarantees that $p_0^y \in L_0^{r+1}$. Hence p_0^x, which is located to the left of p_0^y, is in the domain of φ_0: the duplicator moves p_1^x to $\varphi_0(p_0^x)$. This situation corresponds to Case (IV).

Let us prove that (S_r), (E_r) and (T_r) hold in both of these instances.

Proof of (S_r) in Case (III)

Since the spoiler's move does not fall under Case (I), we have that $p_0^x \notin S_0^r$.
By construction, $a[\tau_l]_L^{r+1} \in L_0^{r+1}$, thus $\varphi_0(a[\tau_l]_L^{r+1}) \in L_1^{r+1}$, and $p_1^x \notin S_1^r$.

Proof of (E_r) in Case (III)

It follows from $\text{env-tp}^k_{(\mathcal{A}_0,<_0)}(a[\tau_l]_L^{r+1}) = \tau_l$ together with Lemma 5 that

$$\text{env-tp}^k_{(\mathcal{A}_0,<_0)}(p_0^x) = \text{env-tp}^k_{(\mathcal{A}_1,<_1)}(p_1^x)\,.$$

A fortiori, $\text{env-tp}^r_{(\mathcal{A}_0,<_0)}(p_0^x) = \text{env-tp}^r_{(\mathcal{A}_1,<_1)}(p_1^x)$.

Proof of (T_r) in Case (III)

Because the spoiler's move does not fall under Case (II), $p_0^x \notin N^1_{\mathcal{A}_0}(p_0^y)$. In other words,

$$\text{tp}^0_{\mathcal{A}_0}(p_0^x, p_0^y) = \text{tp}^0_{\mathcal{A}_0}(p_0^x) \cup \text{tp}^0_{\mathcal{A}_0}(p_0^y)\,.$$

Recall the construction of UL_0^{r+1}: the whole k-neighborhood of $a[\tau_l]_L^{r+1}$ was included in this segment. In particular, $N^1_{\mathcal{A}_1}(p_1^x) = N^1_{\mathcal{A}_1}(\varphi_0(a[\tau_l]_L^{r+1})) \subseteq UL_1^{r+1}$. By assumption, $p_1^y \notin L_1^{r+1}$, which entails that $\text{tp}^0_{\mathcal{A}_1}(p_1^x, p_1^y) = \text{tp}^0_{\mathcal{A}_1}(p_1^x) \cup \text{tp}^0_{\mathcal{A}_1}(p_1^y)$.
It then follows from the last observation of Note 9 that $\text{tp}^0_{\mathcal{A}_0}(p_0^x, p_0^y) = \text{tp}^0_{\mathcal{A}_1}(p_1^x, p_1^y)$.

Let us now prove that $\text{tp}^0_{<_1}(p_1^x, p_1^y) = \{x < y\}$.
We claim that $p_1^y \notin X_1 \cup \bigcup_{0\leq j\leq r+1} L_1^j$. Suppose otherwise: (S_{r+1}) would entail that $p_0^y \in X_0 \cup \bigcup_{0\leq j\leq r+1} L_0^j$ which, together with the hypothesis $p_0^y \notin L_0^{r+1}$ and $p_0^x < p_0^y$, would result in p_0^x being in S_0^r, which is absurd.
Thus, $\text{tp}^0_{<_1}(p_1^x, p_1^y) = \{x < y\} = \text{tp}^0_{<_0}(p_0^x, p_0^y)$, which concludes the proof of (T_r).

Proof of (S_r), (E_r) and (T_r) in Case (IV)

Let us now move to the case where $p_1^y \in L_1^{r+1}$. Recall that under this assumption, $p_0^y = \varphi_1(p_1^y) \in L_0^{r+1}$ and since $p_0^x < p_0^y$ and $p_0^y \notin S_0^r$, we have that $p_0^x \in L_0^{r+1}$.
The duplicator places the pebble p_1^x on $\varphi_0(p_0^x)$; in particular, $p_1^x \in L_1^{r+1}$.

The proof of (S_r) follows from the simple observation that $p_0^x \notin S_0^r$ and $p_1^x \notin S_1^r$.
As for (E_r) and (T_r), they follow readily from Lemma 5 and 6 and the fact that $p_1^x = \varphi_0(p_0^x)$ and $p_1^y = \varphi_0(p_0^y)$.

6 Counting quantifiers

We now consider the natural extension C^2 of FO^2, where one is allowed to use counting quantifiers of the form $\exists^{\geq i}x$ and $\exists^{\geq i}y$, for $i \in \mathbb{N}$. Such a quantifier, as expected, expresses the existence of at least i elements satisfying the formula which follows it. This logic C^2 has been extensively studied. On an expressiveness standpoint, C^2 stricly extends FO^2

(which cannot count up to three), and contrary to the latter, C^2 does not enjoy the small model property (meaning that contrary to FO^2, there exist satisfiable C^2-sentences which do not have small - or even finite - models). However, the satisfiability problem for C^2 is still decidable [6, 15, 16]. To the best of our knowledge, it is not known whether $<$-inv C^2 has a decidable syntax. Let us now explain how the proof of Theorem 1 can be adapted to show the following stronger version:

▶ **Theorem 10.** *Let $\mathcal{C}$ be a class of structures of bounded degree. Then $<$-inv $C^2 \subseteq FO$ on $\mathcal{C}$.*

Proof. The proof is very similar as to that of Theorem 1. The difference is that we now need to show, at the end of the construction, that the structures $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$ are not only FO^2-similar, but C^2-similar. More precisely, we show that for every $k \in \mathbb{N}$, there exists some $f(k) \in \mathbb{N}$ such that if $\mathcal{A}_0 \equiv^{\mathrm{FO}}_{f(k)} \mathcal{A}_1$, then it is possible to construct two linear orders $<_0$ and $<_1$ such that $(\mathcal{A}_0, <_0)$ and $(\mathcal{A}_1, <_1)$ agree on all C^2-sentences of quantifier rank at most k, and with counting indexes at most k, which we denote $(\mathcal{A}_0, <_0) \equiv^{\mathrm{C}^2}_{k,k} (\mathcal{A}_1, <_1)$. This is enough to complete the proof, as these classes of C^2-sentences cover all the C^2-definable properties.

In order to prove that $(\mathcal{A}_0, <_0) \equiv^{\mathrm{C}^2}_{k,k} (\mathcal{A}_1, <_1)$, we need an Ehrenfeucht-Fraïssé-game capturing $\equiv^{\mathrm{C}^2}_{k,k}$. It is not hard to derive such a game from the Ehrenfeucht-Fraïssé-game for C^2 [12]. This game only differs from the two-pebble Ehrenfeucht-Fraïssé-game in that in each round, once the spoiler has chosen a structure (say $(\mathcal{A}_0, <_0)$) and a pebble to move (say p_0^x), the spoiler picks not only one element of that structure, but a set P_0 of up to k elements. Then the duplicator must respond with a set P_1 of same cardinality in $(\mathcal{A}_1, <_1)$. The spoiler then places p_1^x on any element of P_1, to which the duplicator responds by placing p_0^x on some element of P_0. As usual, the spoiler wins after this round if $\mathrm{tp}^0_{(\mathcal{A}_0,<_0)}(p_0^x, p_0^y) \neq \mathrm{tp}^0_{(\mathcal{A}_1,<_1)}(p_1^x, p_1^y)$. Otherwise, the game goes on until k rounds are played.

It is not hard to establish that this game indeed captures $\equiv^{\mathrm{C}^2}_{k,k}$, in the sense that $(\mathcal{A}_0, <_0) \equiv^{\mathrm{C}^2}_{k,k} (\mathcal{A}_1, <_1)$ if and only if the duplicator has a winning strategy for k rounds of this game. The restriction on the cardinal of the set chosen by the spoiler (which is at most k) indeed corresponds to the fact that the counting indexes of the formulas are at most k. As for the number of rounds (namely, k), it corresponds as usual to the quantifier rank. This can be easily derived from a proof of Theorem 5.3 in [12], and is left to the reader.

Let us now explain how to modify the construction of $<_0$ and $<_1$ presented in Section 4 in order for the duplicator to maintain similarity for k-round in such a game. The only difference lies in the choice of the universal elements. Recall that in the previous construction, we chose, for each k-environment type τ_l extending a frequent k-neighborhood type and each segment UL_0^j, an element $a[\tau_l]^j_L$ whose k-environment type in $(\mathcal{A}_0, <_0)$ is destined to be τ_l (and similarly for UR_0^j and $a[\tau_l]^j_R$).

In the new construction, we pick k such elements, instead of just one. Just as previously, all these elements must be far enough from one another in the Gaifman graph of $\mathcal{A}_0$. Once again, this condition can be met by virtue of the k-neighborhood type τ underlying τ_l being frequent, and thus having many occurrences scattered across $\mathcal{A}_0$ (remember that we have a bound on the degree of $\mathcal{A}_0$, thus all the occurrences of τ cannot be concentrated). We only need to multiply the value of m by k in (3).

When the spoiler picks a set of elements of size at most k in one of the structures (say P_0 in $(\mathcal{A}_0, <_0)$), the duplicator responds by selecting, for each one of the elements of P_0, an element in $(\mathcal{A}_1, <_1)$ along the strategy for the FO^2-game explained in Section 5.3. All that remains to be shown is that it is possible for the duplicator to answer each element of P_0 with a different element in $(\mathcal{A}_1, <_1)$.

Note that if the duplicator follows the strategy from Section 5.3, they will never answer two moves by the spoiler falling under different cases among Cases (I)-(VI) with the same element. Thus we can treat separately each one of these cases; and for each case, we show that if the spoiler chooses up to k elements in $(\mathcal{A}_0, <_0)$ falling under this case in P_0, then the duplicator can find the same number of elements in $(\mathcal{A}_1, <_1)$, following the aforementionned strategy.

- For Case (I), this is straightforward, since the strategy is based on the isomorphism between the borders of the linear orders. The same goes for Cases (II), (IV) and (VI), as the strategy in these cases also relies on an isomorphism argument.
- Suppose now that $p_0^y \notin L_0^{r+1}$, and assume that the spoiler chooses several elements to the left of p_0^y, but outside of S_0^r and not adjacent to p_0^y. This corresponds to Case (III). Recall that our new construction guarantees, for each k-environment type extending a frequent k-neighborhood type, the existence in L_1^{r+1} of k elements having this environment. This lets us choose, in L_1^{r+1}, a distinct answer for each element in the set selected by the spoiler, sharing the same k-environment type. Case (V) is obviously symmetric.

This concludes the proof of Theorem 10. ◀

7 Conclusion

We have established that, when the degree is bounded, properties definable in the order-invariant extension of the two-variable fragment of first-order logic with counting are definable in first-order logic. From there, there seem to be three axes in which one can try to complete the picture.

The first natural question is whether this inclusion of expressiveness still holds when we release the hypothesis on the degree. As stated in the introduction, an upcoming result [2] exhibits two classes of structures (of unbounded degree) $\mathcal{C}' \subseteq \mathcal{C}$ such that (i) $\mathcal{C}'$ is definable in $\mathcal{C}$ by an FO^2-sentence φ using an order, such that φ is order-invariant on all structures of $\mathcal{C}$ (ii) $\mathcal{C}'$ is not FO-definable, and (iii) $\mathcal{C}$ is FO^3-definable but not FO^2-definable. Note that according to the standard definition, φ is not a sentence of $<$-inv FO^2, because its order-invariance does not hold on all finite structures. However, in view of this construction, it stands to reason to believe that in general, $<$-inv $\mathrm{FO}^2 \not\subseteq$ FO. Formally proving this non-inclusion would be an interesting goal.

Second, it is quite clear that the inclusion $<$-inv $\mathrm{C}^2 \subseteq$ FO when the degree is bounded is a severe over-approximation of the expressive power of $<$-inv C^2. For instance, it is not hard to prove that $<$-inv C^2 cannot define the class of triangle-free graphs: no sentence from $<$-inv C^2 can make a distinction between a large enough cycle and the disjoint union of a large enough cycle together with a triangle. This can be seen, following the general strategy detailed in Section 3, by constructing two carefully chosen linear orders on these graphs. Figure 2 illustrates the construction of such orders. Notice that there are only three kinds of elements: those whose two neighbors are on their left, those for which they are on their right, and those which have one neighbor on each side. By making sure to always respond to a move by the spoiler with an element of the same kind (and, of course, by implementing a tit-for-tat strategy near the endpoints), the duplicator can easily win the Ehrenfeucht-Fraïssé-game capturing $\equiv_{k,k}^{\mathrm{C}^2}$, provided that the cycles are long enough with respect to k.

It would be interesting to find upper bounds for $<$-inv FO^2 and $<$-inv C^2 tighter than FO; that is, tighter than the fragment $\exists^*\forall^*\exists^*$FO, to which FO collaspes when the degree is bounded [4] (since already in this fragment, one can count the number of occurrences of neighborhood types up to some threshold). Let us briefly explain why we fall short of giving

Figure 2 Illustration of two linear orders (growing from left to right) on a cycle (left figure) and the disjoint union of a cycle and a triangle (right figure), which are indistinguishable by any C^2-sentence of small enough quantifier rank and maximal counting index (where "small" is understood with respect to the length of the cycles).

such a bound: in such an attempt, the initial assumption about the similarity between $\mathcal{A}_0$ and $\mathcal{A}_1$ would be weaker than FO-similarity, and it would not be possible to base our work on neighborhoods. In this context, the starting hypothesis on the two structures would lack the rigidity which seems necessary to construct linear orders preserving FO^2-similarity or C^2-similarity. Establishing such a tighter bound thus seems to call for new techniques.

Last, we conjecture that the inclusion still holds when we lift the restriction on the number of variables; namely that $<$-inv FO = FO when the degree is bounded. This would generalize the equality Succ-inv FO = FO when the degree is bounded, obtained in [7]. Our hope is that a construction inspired by this one, albeit significantly refined, and in particular by the alternation of universal and neighbors segments, could possibly lead to establish such a result. We leave these three questions, as well as the issue of the syntactic decidability of $<$-inv C^2, for further research.

References

1 Albert Atserias, Anuj Dawar, and Martin Grohe. Preservation under extensions on well-behaved finite structures. *SIAM Journal on Computing*, 2008.

2 Bartosz Bednarczyk. Personal communication, July 2022.

3 Michael Benedikt and Luc Segoufin. Towards a characterization of order-invariant queries over tame graphs. *J. Symb. Log.*, 2009.

4 Ronald Fagin, Larry J. Stockmeyer, and Moshe Y. Vardi. On monadic NP vs. monadic co-NP. *Inf. Comput.*, 1995.

5 Erich Grädel and Martin Otto. On logics with two variables. *Theor. Comput. Sci.*, 1999.

6 Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *LICS*, 1997.

7 Julien Grange. Successor-invariant first-order logic on classes of bounded degree. *Log. Methods Comput. Sci.*, 2021.

8 Julien Grange and Luc Segoufin. Order-Invariant First-Order Logic over Hollow Trees. In *Computer Science Logic, CSL*, 2020.

9 Martin Grohe and Thomas Schwentick. Locality of order-invariant first-order formulas. *ACM Trans. Comput. Log.*, 2000. doi:10.1145/343369.343386.

10 Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 1986.

11 Neil Immerman and Dexter Kozen. Definability with bounded number of bound variables. *Inf. Comput.*, 1989.

12 Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In *Complexity theory retrospective*. Springer, 1990.

13 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

14 Michael Mortimer. On languages with two variables. *Math. Log. Q.*, 1975.

15 Ian Pratt-Hartmann. Complexity of the guarded two-variable fragment with counting quantifiers. *J. Log. Comput.*, 2007.

16 Ian Pratt-Hartmann. The two-variable fragment with counting revisited. In *WoLLIC*, 2010.

17 Moshe Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, 1982.

18 Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FOˆ2 on words. *Log. Methods Comput. Sci.*, 2009.

19 Thomas Zeume and Frederik Harwath. Order-invariance of two-variable logic is decidable. In *LICS*, 2016.

Explorable Automata

Emile Hazard
CNRS, LIP, ENS Lyon, France

Denis Kuperberg
CNRS, LIP, ENS Lyon, France

Abstract

We define the class of explorable automata on finite or infinite words. This is a generalization of History-Deterministic (HD) automata, where this time non-deterministic choices can be resolved by building finitely many simultaneous runs instead of just one. We show that recognizing HD parity automata of fixed index among explorable ones is in PTime, thereby giving a strong link between the two notions. We then show that recognizing explorable automata is ExpTime-complete, in the case of finite words or Büchi automata. Additionally, we define the notion of ω-explorable automata on infinite words, where countably many runs can be used to resolve the non-deterministic choices. We show that all reachability automata are ω-explorable, but this is not the case for safety ones. We finally show ExpTime-completeness for ω-explorability of automata on infinite words for the safety and co-Büchi acceptance conditions.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Nondeterminism, automata, complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.24

Related Version *Full Version*: https://hal.archives-ouvertes.fr/hal-03669659/

1 Introduction

In several fields of theoretical science, the tension between deterministic and non-deterministic models is a source of fundamental open questions, and has led to important lines of research. The most famous of this kind is the P vs NP question in complexity theory. This paper aims at further investigating the frontier between determinism and non-determinism in automata theory. Although Non-deterministic and Deterministic Finite Automata (NFA and DFA) are known to be equivalent in terms of expressive power, many subtle questions remain about the cost of determinism, and a deep understanding of non-determinism will be needed to solve them.

One of the approaches investigating non-determinism in automata is the study of History-Deterministic (HD) automata, introduced in [16] under the name Good-For-Game (GFG) automata. An automaton is HD if, when reading input letters one by one, its non-determinism can be resolved on-the-fly without any need to guess the future. This constitutes a model that is intermediary between non-determinism and determinism, and can sometimes bring the best of both worlds. Like deterministic automata, HD automata on infinite words retain good properties such as their soundness with respect to composition with games, making them appropriate for use in Church synthesis algorithms [16]. On the other hand, like non-deterministic automata, they can be exponentially more succinct than deterministic ones [19]. There is a very active line of research trying to understand the various properties of HD automata, see e.g. [1, 2, 7, 9, 20, 13, 24] for latest developments. The terminology *history-deterministic*, was introduced originally in the theory of regular cost functions [11]. The name "history-deterministic" corresponds to the above intuition of solving non-determinism on-the-fly, while the earlier name of "good-for-games" refers to sound composition with games. These two notions may actually differ in some quantitative frameworks, but coincide

on boolean automata [8], and have been used interchangeably in most of the literature on the topic. In this paper, since we are mainly interested in resolving the non-determinism on-the-fly, we choose the HD denomination to emphasize this aspect[1].

The goal of this paper is to pursue this line of research by introducing and studying the class of explorable automata on finite and infinite words. The intuition behind explorability is to limit the amount of non-determinism required by the automaton to accept its language, in a more permissive way than HD automata. If $k \in \mathbb{N}$, an automaton is k-explorable if when reading input letters, it suffices to keep track of k runs to build an accepting one, if it exists. An automaton is explorable if it is k-explorable for some $k \in \mathbb{N}$. This can be seen as a variation on the notion of HD automaton, which corresponds to the case $k = 1$. The present work can be compared to [18], where a notion related to k-explorability (called *width*) is introduced and studied, see Section 2.3. In particular, some results of [18] also apply to k-explorability, notably ExpTime-completeness of deciding k-explorability of an NFA if k is part of the input. Surprisingly however, the techniques used in [18] are quite different from the ones we need here. This shows that fixing a bound k for the number of runs leads to very different problems compared to asking for the existence of such a bound.

One of the motivations to introduce the notion of explorability is to tackle one of the important open questions about HD automata: what is the complexity of deciding whether an automaton is HD? We explain in the following why explorability is relevant for this question, and show obstructions to some of our initial hopes in this direction.

Recognizing HD automata is known to be in PTime for Büchi [3] and co-Büchi [19] automata, but even for 3 parity ranks, the only known upper bound is ExpTime via the naive algorithm from [16]. We show how explorable automata can simplify this question: if the input automaton is explorable, then the problem becomes PTime for any fixed acceptance condition. Therefore, the question of recognizing HD automata can be shifted to: how hard is it to recognize explorable automata?

We then proceed to study the decidability and complexity of the explorability problem: deciding whether an input automaton on finite or infinite words is explorable. For this, we establish a connection with the population control problem studied in [4]. This problem asks, given an NFA with an arbitrary number of tokens in the initial state, whether a controller can choose input letters, thereby forcing every token to reach a designated state, even if tokens are controlled by an opponent. It is shown in [4] that the population control problem is ExpTime-complete, and we adapt their proof to our setting to show that the explorability problem is ExpTime-complete as well, already for NFAs. We also show that a direct reduction is possible, but at an exponential cost, yielding only a 2-ExpTime algorithm for the NFA explorability problem. In the case of infinite words, we adapt the proof to the Büchi case, thereby showing that the Büchi explorability problem is in ExpTime as well. We also remark that, as in [4], the number of tokens needed to witness explorability can go as high as doubly exponential in the size of the automaton.

This ExpTime-completeness result means that we unfortunately cannot directly use the intermediate notion of explorable automata to improve on the complexity of recognizing HD automata in full generality, as could have been the hope. However, there can also be some frameworks where we can guarantee to obtain an explorable automaton, and therefore easily decide whether it is HD. A recent example of this from [9] is detailed in Section 2.4. More generally, we believe that this explorability notion is of interest towards a better understanding of non-determinism in automata theory.

[1] This departs from earlier practices consisting in using HD and GFG in a way coherent with their contexts of introduction: HD for cost functions and GFG for boolean automata. Hence most of the papers cited here use GFG.

Notice that interestingly, from a model-checking perspective, our approach is dual to [4]: in the population control problem, an NFA is well-behaved when we can "control" it by forcing arbitrarily many runs to simultaneously reach a designated state, via an appropriate choice of input letters. On the contrary, in our approach, the input letters form an adversarial environment, and our NFA is well-behaved when its non-determinism is limited, in the sense that it is enough to spread finitely many runs to explore all possible behaviors.

On infinite words, we push further the notion of explorability, by remarking that for some automata, even following a countable number of runs is not enough. This leads to defining the class of *ω-explorable automata*, as those automata on infinite words where non-determinism can be resolved using countably many runs. We show that ω-explorable automata form a non-trivial class even for the safety acceptance condition (but not for reachability), and give an ExpTime algorithm recognizing ω-explorable automata, encompassing the safety and co-Büchi conditions. We also show ExpTime-hardness of this problem, by adapting the ExpTime-hardness proof of [4] to the setting of ω-explorability.

Summary of the contributions. We show that given an explorable parity automaton of fixed parity index, it is in PTime to solve its *HDness problem*, i.e. decide whether it is HD. The idea was already used in [3], and in [9] for quantitative automata. The algorithm used for Büchi HDness in [3] is conjectured to work for any acceptance condition (this is the "G_2 conjecture"), and it is in fact this algorithm that is shown here to work on any explorable parity automaton.

We show that given an NFA or Büchi automaton, it is decidable and ExpTime-complete to check whether it is explorable. Our proof of ExpTime-completeness for NFAs uses techniques developed in [4], where ExpTime-completeness is shown for the NFA population control problem. We generalize this result to ExpTime explorability checking for Büchi automata, requiring further adaptations. We also give a black box reduction using the result from [4]. This is enough to show decidability of the NFA explorability problem, but it yields a 2-ExpTime algorithm. As in [4], the ExpTime algorithm yields a doubly exponential tight upper bound on the number of tokens needed to witness explorability.

On infinite words, we show that any reachability automaton is ω-explorable, but that this is not the case for safety automata. We show that both the safety and co-Büchi ω-explorability problems are ExpTime-complete.

Related Works. Many works aim at quantifying the amount of non-determinism in automata. A survey by Colcombet [12] gives useful references on this question. Let us mention for instance the notion of ambiguity, which quantifies the number of simultaneous accepting runs. Similarly to [18], we can note that ambiguity is orthogonal to k-explorability. Remark however that our finite/countable/uncountable explorability hierarchy is reminiscent of the finite/polynomial/exponential ambiguity hierarchy (see e.g. [25]).

In [17], several ways of quantifying the non-determinism in automata are studied from the point of view of complexity, including notions such as the number of advice bits needed.

Another approach is studied in [23], where a measure of the maximum non-deterministic branching along a run is defined and compared to other existing measures.

Following the HD approach, a hierarchy of non-determinism and an analysis of this hierarchy via probabilistic models is given in [2].

The idea of k-explorability stems from the approach in [3], using games with tokens to tackle the HDness problem for Büchi automata. In this previous work, the idea of following a finite number of runs in parallel plays a central role in the proof. Remark however that

the notion of explorability as studied here is stronger than what is needed in [3]. The k-explorability (and explorability) property was explicitly defined under the name k-History-Determinism in [9], as a proof tool to decide the HDness of LimInf and LimSup automata. The work [9] is part of a research effort to understand how partial determinism notions such as HDness play out in quantitative automata, see survey [5]. Our goal here is to investigate explorability as defining a natural class of automata on finite and infinite words, somehow giving it an "official status" not restricted to an intermediate proof tool.

History of this work. It is traditional in our community to present results as a finished product, abstracting away the path that led to it. This paragraph is an experiment: we believe that in addition to this practice, it can be interesting for the reader to have access to a history of how ideas developed.

The interest we took in the explorability notion originated in the fact that it makes deciding HDness much easier, and the hope was that by using this notion as an intermediate, we could obtain an algorithm improving on the ExpTime upper bound for deciding GFGness of parity automata of fixed index, e.g. to PSpace. As we described above, we ended up showing that this approach cannot yield an algorithm below ExpTime (at least not in full generality). However, although this was initially only a tool for this decision problem, explorability turns out to be a natural generalisation of HD automata, and an interesting class to study in itself. The first investigation of this notion, and in particular of its decidability, was the object of a short research internship by Milla Valnet under the supervision of the second author. It was expected that decidability of explorability would be a reachable goal for such a short internship, but it turned out that this was overly optimistic. The internship yielded preliminary results, and in particular was useful to introduce and study the notion of "coverability". This version of the problem does not take accepting conditions into account, but only asks that at any point of the run, every state that could be reached is actually occupied by a token. After the internship, we continued to use this coverability notion as a stepping stone towards an understanding of explorability. However, after more preliminary results and unsuccessful attempts at obtaining decidability, we discovered the connection between explorability and population control from [4], that rendered the intermediate notion of coverability useless for our purposes, and we then focused on exploiting that link. We chose to leave coverability out of the present exposition, as it feels like a "watered-down" version of explorability, but it could be useful in some contexts, hence we briefly mention it in this chronological account. Let us just informally state here that it is straightforward to modify our proofs in order to show that deciding whether an NFA is coverable is ExpTime-complete as well.

2 Explorable automata

2.1 Automata

If $i \leq j$ are integers, we will denote by $[i, j]$ the integer interval $\{i, i+1, \ldots, j\}$. If S is a set, its cardinal will be denoted $|S|$, and its powerset $\mathcal{P}(S)$.

We work with a fixed finite alphabet Σ. We will use the following default notation for the components of an automaton $\mathcal{A}$: $Q_{\mathcal{A}}$ for its set of states, $q_0^{\mathcal{A}}$ for its initial state, $F_{\mathcal{A}}$ for its accepting states, $\Delta_{\mathcal{A}}$ for its set of transitions. Letter $\mathcal{A}$ in the components might be omitted when clear from context. We might also specify its alphabet by $\Sigma_{\mathcal{A}}$ instead of Σ for cases where different alphabets come into play. If $\Delta \subseteq Q \times \Sigma \times Q$ is the transition

relation, and $(p, a) \in Q \times \Sigma$, we will note $\Delta(p, a) = \{q \in Q, (p, a, q) \in \Delta\}$. If $X \subseteq Q$, we note $\Delta(X, a) = \bigcup_{p \in X} \Delta(p, a)$.

To simplify definitions, all automata in this paper will be assumed to be complete (by adding a rejecting sink state if needed). This means that for all $(p, a) \in Q \times \Sigma$, we assume $\Delta(p, a) \neq \emptyset$. The rejecting sink state will often be implicit in our constructions and examples.

We will consider non-deterministic automata on finite words (NFAs). A run of such an automaton on a word $a_1 a_2 \dots a_n \in \Sigma^*$ is a sequence of states $q_0 q_1 \dots q_n \in Q^*$ (q_0 being the initial state), such that, for all $i \in [0, n-1]$, we have $q_{i+1} \in \Delta(q_i, a_{i+1})$. Such a run is accepting if $q_n \in F$, i.e. if the run belongs to Q^*F. As usual, the language of an automaton $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that admit an accepting run.

We will also deal with automata on infinite words, and we recall here some of the standard acceptance conditions for such automata. A run on an infinite word $w \in \Sigma^\omega$ is now an infinite sequence of states, i.e. an element of Q^ω, starting in q_0 and following as before transitions of the automaton according to the letters of w. Such a run of Q^ω is accepting in a safety (resp. reachability, Büchi, co-Büchi) automaton if it belongs to F^ω (resp. Q^*FQ^ω, $(Q^*F)^\omega$, Q^*F^ω). States from F will be called Büchi states in Büchi automata, and states from $Q \setminus F$ will be called co-Büchi states in co-Büchi automata.

Finally, we also mention the parity acceptance condition: it uses a ranking function rk from Q to an interval of integers $[i, j]$. A run is accepting if the minimal rank appearing infinitely often is even (following the convention of [4]).

We will also use standard game notions such as arena, play and strategy. See e.g. [15] for a complete definition of these objects.

2.2 Games

A *game* $\mathcal{G} = (V_0, V_1, v_I, E, W)$ of infinite duration between two players 0 and 1 consists of: a finite set of *positions* V being a disjoint union of V_0 and V_1; an *initial position* $v_I \in V$; a set of *edges* $E \subseteq V \times V$; and a *winning condition* $W \subseteq V^\omega$. We will later use names more explicit than 0 and 1 for the players, describing their roles in the various games we will define.

A *play* is an infinite sequence of positions $v_0 v_1 v_2 \dots \in V^\omega$ such that $v_0 = v_I$ and for all $n \in \mathbb{N}$, $(v_n, v_{n+1}) \in E$. A play $\pi \in V^\omega$ is *winning* for Player 0 if it belongs to W. Otherwise π is *winning* for Player 1.

A *strategy* for Player 0 (resp. 1) is a function $\sigma_0 : V^* \times V_0 \to V$ (resp. $\sigma_1 : V^* \times V_1 \to V$), describing which edge should be played given the history of the play $u \in V^*$ and the current position $v \in V$. A strategy σ_P has to obey the edge relation, i.e. there has to be an edge in E from v to $\sigma_P(u, v)$. A play $\pi = v_0 v_1 v_2 \dots$ is *consistent* with a strategy σ_P of a player P if for every n such that $v_n \in V_P$ we have $v_{n+1} = \sigma_P(v_0 \dots v_{n-1}, v_n)$.

A strategy for Player 0 (resp. Player 1) is *positional* (or *memoryless*) if it does not use the history of the play, i.e. it can be seen as a function $V_0 \to V$ (resp. $V_1 \to V$).

We say that a strategy σ_P of a player P is *winning* if every play consistent with σ_P is winning for P. In this case, we say that P *wins* the game $\mathcal{G}$.

A game is *positionally determined* if exactly one of the players has a positional winning strategy in the game.

In the interest of readability, when describing games in the paper, we will not give explicit definitions of the sets V_0, V_1 and E, but give slightly more informal descriptions in terms of possible actions of players at each round. It is straightforward to build a formal description of the game from such a description.

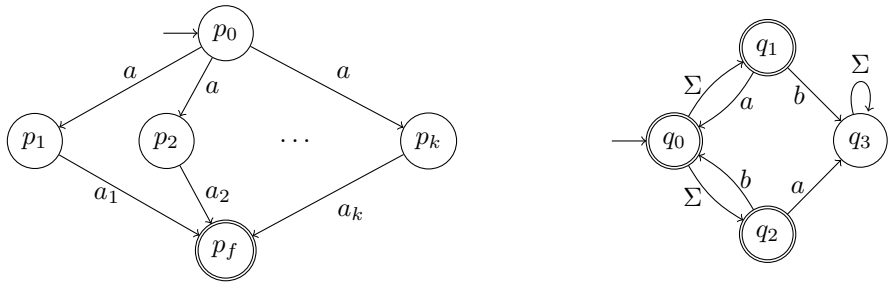


Figure 1 An explorable and a non-explorable automata.

2.3 Explorability

We start by introducing the *k-explorability game*, which is the central tool allowing us to define the class of explorable automata.

▶ **Definition 1** (k-explorability game)**.** *Consider a non-deterministic automaton $\mathcal{A}$ on finite or infinite words, and an integer k. The k-explorability game on $\mathcal{A}$ is played on the arena Q^k. The two players are called Determiniser and Spoiler, and they play as follows.*

- *The initial position is the k-tuple $S_0 = (q_0, \dots, q_0)$.*
- *At step i from a position $S_{i-1} \in Q^k$, Spoiler chooses a letter $a_i \in \Sigma$, and Determiniser chooses $S_i \in Q^k$ such that for every token $l \in [0, k-1]$, $S_{i-1}(l) \xrightarrow{a_i} S_i(l)$ is a transition of $\mathcal{A}$ (where $S_i(l)$ stands for the l-th component in S_i).*

The play is won by Determiniser if for all $\beta \leq \omega$ such that the word $(a_i)_{1 \leq i < \beta}$ is in $\mathcal{L}(\mathcal{A})$, there is a token $l < k$ being accepted by $\mathcal{A}$, meaning that the sequence $(S_i(l))_{i<\beta}$ is an accepting run[2]*. Otherwise, the winner is Spoiler.*
We will say that $\mathcal{A}$ is k-explorable *if Determiniser wins the k-explorability game (i.e. has a winning strategy, ensuring the win independently of the choices of Spoiler).*
We will say that $\mathcal{A}$ is explorable *if it is k-explorable for some $k \in \mathbb{N}$.*

▶ **Example 2.** Consider the automata from Figure 1. The NFA $\mathcal{A}_k$ on alphabet $\{a, a_1, \dots, a_k\}$ is k-explorable, but not $(k-1)$-explorable. It can easily be adapted to a binary alphabet, by replacing in the automaton $a_1, \dots, a_k$ by distinct words of the same length.

On the other hand, the NFA $\mathcal{C}$ is a non-explorable NFA accepting all words on alphabet $\Sigma = \{a, b\}$. Indeed, Spoiler can win the k-explorability game for all k, by eliminating tokens one by one, choosing at each step the letter b if q_1 is occupied by at least one token, and the letter a otherwise.

▶ **Example 3.** The NFA $\mathcal{B}_k$ from Figure 2 with $3k+1$ states on alphabet $\Sigma = \{a, b\}$ is explorable, but requires 2^k tokens. Indeed, since when choosing the $2i^{\text{th}}$ letter Spoiler can always pick the state p_i or r_i containing the least amount of tokens to decide whether to play a or b, the best strategy for Determiniser is to split his tokens evenly at each q_i. This means he needs to start with 2^k tokens to end up with at least one token in q_k after a word of Σ^{2k}.

[2] This condition $\beta \leq \omega$ is actually accounting separately for the two cases of finite and infinite words, corresponding respectively to $\beta < \omega$ and $\beta = \omega$.

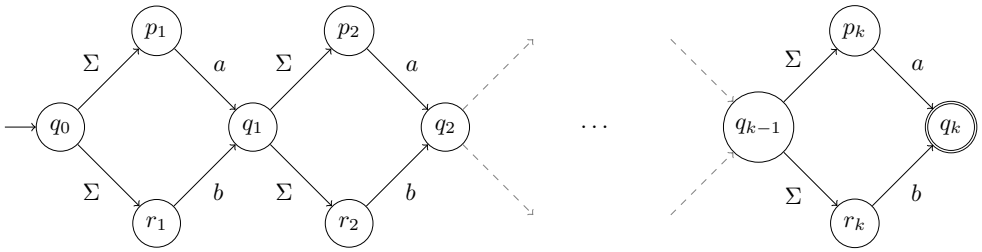


Figure 2 An explorable automaton $\mathcal{B}_k$ requiring exponentially many tokens.

Let us mention a few facts that follow from the definition of explorability:

▶ **Lemma 4.**

- *Any finite language is explorable.*
- *If $\mathcal{A}$ is k-explorable, then it is n-explorable for all $n \geq k$.*
- *If $\mathcal{A}$ is k-explorable and $\mathcal{B}$ is n-explorable, then*
  - *$\mathcal{A} \cup \mathcal{B}$ (with states $Q = \{q_0\} \cup Q_\mathcal{A} \cup Q_\mathcal{B}$) is $(k+n)$-explorable,*
  - *the union product $\mathcal{A} \times \mathcal{B}$ (with $F = (F_\mathcal{A} \times Q_\mathcal{B}) \cup (Q_\mathcal{A} \times F_\mathcal{B})$) is $\max(k,n)$-explorable,*
  - *the intersection product $\mathcal{A} \times \mathcal{B}$ (with $F = F_\mathcal{A} \times F_\mathcal{B}$) is (kn)-explorable.*

Proof. If $L(\mathcal{A})$ is finite, it is enough to take $k = |L(\mathcal{A})|$ tokens to witness explorability: for each $u \in L(\mathcal{A})$, the token t_u assumes that the input word is u and follows an accepting run of $\mathcal{A}$ over u as long as input letters are compatible with u. As soon as an input letter is not compatible with u, the token t_u is discarded and behaves arbitrarily for the rest of the play.

If $\mathcal{A}$ is k-explorable and $n \geq k$, then Determiniser can win the n-explorability game by using the same strategy with the first k tokens and making arbitrary choices with the $n-k$ remaining tokens.

If $\mathcal{A}$ and $\mathcal{B}$ are k- and n-explorable respectively, then Determiniser can use both strategies simultaneously with $k+n$ tokens in $\mathcal{A} \cup \mathcal{B}$, using k tokens in $\mathcal{A}$ and n tokens in $\mathcal{B}$. If the input word is in $\mathcal{A}$ (resp. $\mathcal{B}$), then the tokens playing in $\mathcal{A}$ (resp. $\mathcal{B}$) will win the play.

In the union product $\mathcal{A} \times \mathcal{B}$, it is enough to take $\max(k,n)$ tokens: if $0 \leq i < \min(k,n)$, the token number i follows the strategy of the token i in $\mathcal{A}$ on the first coordinate, and the strategy of the token i in $\mathcal{B}$ in the second one. If $\min(k,n) \leq i < \max(k,n)$, say wlog $k \leq i < n$, the token i follows an arbitrary strategy on the $\mathcal{A}$-component and the strategy of token i on the $\mathcal{B}$-component.

However, Determiniser may need up to kn tokens to play in $\mathcal{A} \times \mathcal{B}$ when the accepting set is $F_\mathcal{A} \times F_\mathcal{B}$: the token (i,j) will use the strategy of the token i in the k-explorability game of $\mathcal{A}$ together with the strategy of the token j in the n-explorability game of $\mathcal{B}$. This lower bound of kn cannot be improved: consider for instance the intersection product $\mathcal{A}_k \times \mathcal{A}_n$, where $\mathcal{A}_k, \mathcal{A}_n$ are from Example 2, using as alphabet the cartesian product of their respective alphabets: $\{a, a_1, a_2, \ldots, a_k\} \times \{a, b\}$. ◀

Notice that a similar notion was introduced in [18] under the name *width*. In [18], the emphasis is put on another version of the explorability game, where tokens can be duplicated, and $|Q|$ is an upper bound for the number of necessary tokens. In this work, we will on the contrary focus on non-duplicable tokens. However, some results of [18] still apply here. In particular the following holds:

▶ **Theorem 5** ([18, Rem. 6.9]). *Given an NFA $\mathcal{A}$ and an integer k, it is ExpTime-complete to decide whether $\mathcal{A}$ is k-explorable (even if we fix $k = |Q_\mathcal{A}|/2$).*

We aim here at answering a different question:

▶ **Definition 6** (Explorability problem). *The explorability problem is the question, given a non-deterministic automaton $\mathcal{A}$, of deciding whether it is explorable (i.e., whether there exists $k \in \mathbb{N}$ such that it is k-explorable).*

Questions: Is the explorability problem decidable? If yes, what is its complexity?
We will first give some motivation for this problem in Section 2.4.

2.4 Link with HD automata

An automaton $\mathcal{A}$ is History-Deterministic (HD) if it is 1-explorable, i.e. if there is a strategy $\sigma : \Sigma^* \to Q$ resolving the non-determinism based on the word read so far, with the guarantee that the run piloted by this strategy is accepting whenever the input word is in $L(\mathcal{A})$. See e.g. [6] for an introduction to HD automata.

We will give here an additional and stronger link between explorable and HD automata. In this part, we will mainly be interested in automata on infinite words.

The arguments in this section are already hinted at in [3], and made explicit in the context of quantitative automata in [9]. We sketch them here for completeness, in order to give some context for the relevance of the class of explorable automata.

One of the main open problems related to HD automata on infinite words is to decide, given a non-deterministic parity automaton, whether it is HD. For now, the problem is only known to be in PTime for co-Büchi [19] and Büchi [3] automata. Extending this result even to 3 parity ranks is still open, and only a naive ExpTime upper bound [16] is known in this case. The following result shows that explorability is relevant in this context:

▶ **Theorem 7.** *Given an explorable parity automaton $\mathcal{A}$ of fixed parity index, it is in PTime to decide whether it is HD.*

This is one of the motivations to get a better understanding of explorable automata. Indeed, if we can obtain an efficient algorithm for recognizing them, or if we are in a context guaranteeing that we are only dealing with explorable automata, this result shows that we can obtain an efficient algorithm for recognizing HD automata. Alternatively, even if membership to the class of explorable automata is provably hard to decide in general (as it will turn out), there can be some contexts where explorable automata are sufficient for the intended purposes. An example is given in [9], where it is shown that for LimSup and LimInf automata, Eve winning the game G_2 (defined below) implies that the automaton is explorable. Since Theorem 7 actually shows that Eve winning G_2 characterizes HDness for explorable automata, in this case it implies that the automaton is HD, as was shown in [9].

The rest of this section will be devoted to give a proof sketch of Theorem 7. See the long version (Appendix A.1) for formal details.

Let $\mathcal{A}$ be an explorable parity automaton, of fixed parity index $[i, j]$.

We briefly recall the definition of the k-token game $G_k(\mathcal{A})$ defined in [3], for an arbitrary $k \in \mathbb{N}$. At each round, Adam plays a letter $a \in \Sigma$, then Eve moves her token according to an a-transition, and finally Adam moves his k tokens according to a-transitions. Eve wins the play if her token builds an accepting run, or if all of Adam's tokens build a rejecting run.

We will prove that the game $G_2(\mathcal{A})$ is won by Eve if and only $\mathcal{A}$ is HD. Since $G_2(\mathcal{A})$ can be solved in PTime for fixed parity index [3], this is enough to conclude.

First, it is clear that if $\mathcal{A}$ is HD, then Eve wins $G_2(\mathcal{A})$ [3]: Eve can simply play her HD strategy with her token, ignoring Adam's tokens.

The interesting direction is the converse: we assume that Eve wins $G_2(\mathcal{A})$, and we show that under this assumption, $\mathcal{A}$ is necessarily HD. We use the following lemma:

▶ **Lemma 8** ([3, Thm. 14]). *Eve wins $G_2(\mathcal{A})$ if and only if Eve wins $G_k(\mathcal{A})$ for all $k \geq 2$.*

Since $\mathcal{A}$ is explorable, there is $k \in \mathbb{N}$ such that $\mathcal{A}$ is k-explorable. Let τ_k be a winning strategy for Determiniser in the k-explorability game of $\mathcal{A}$, and σ_k be a winning strategy for Eve in $G_k(\mathcal{A})$. We show that we can combine these two strategies to yield a HD strategy σ for $\mathcal{A}$. This proof follows the same idea as in [3] where the explorability hypothesis is not available, but $\mathcal{A}$ is assumed to be Büchi. The strategy σ will store k virtual tokens in its memory. When the automaton reads a new letter $a \in \Sigma$, these k tokens will be updated according to τ_k. Then the choice of σ will follow the strategy σ_k against these k tokens. Notice that the strategies τ_k and σ_k might use additional memory, but this is completely transparent in this proof scheme. If the input word is in $L(\mathcal{A})$, then by correctness of τ_k, one of the k virtual tokens will accept. Thus, by correctness of σ_k, the run chosen by σ will be accepting. Therefore, σ is a correct HD strategy, witnessing that $\mathcal{A}$ is HD. This concludes the proof sketch of Theorem 7.

3 Decidability and complexity of the explorability problem

In this section, we prove that the explorability problem is decidable and ExpTime-complete.

We start by showing in Section 3.1 decidability of the explorability problem for NFAs using the results of [4] as a black box. This yields an algorithm in 2-ExpTime. We give in Section 3.2 a polynomial reduction in the other direction, thereby obtaining ExpTime-hardness of the NFA explorability problem. To obtain a matching upper bound and show ExpTime-completeness, we use again [4], but this time we must "open the black box" and dig into the technicalities of their ExpTime algorithm while adapting them to our setting. We do so in Section 3.3, directly treating the more general case of Büchi automata.

3.1 2-ExpTime algorithm via a black box reduction

Let us start by recalling the population control problem (PCP) of [4].

▶ **Definition 9** (k-population game). *Given an NFA $\mathcal{B}$ with a distinguished target state $f \in Q_{\mathcal{B}}$, and an integer $k \in \mathbb{N}$, the k-population game is played similarly to the k-explorability game, only the winning condition differs: Spoiler wins if the game reaches a position where all tokens are in the state f.*

The PCP asks, given $\mathcal{B}$ and $f \in Q_{\mathcal{B}}$, whether Spoiler wins the k-population game for all $k \in \mathbb{N}$. Notice that this convention is opposite to explorability, where positive instances are defined via a win of Determiniser. The PCP is shown in [4] to be ExpTime-complete. We will present here a direct exponential reduction from the explorability problem to the PCP.

Let $\mathcal{A} = (\Sigma, Q_{\mathcal{A}}, q_0^{\mathcal{A}}, F_{\mathcal{A}}, \Delta_{\mathcal{A}})$ be an NFA. Our goal is to build an exponential NFA $\mathcal{B}$ with a distinguished state f such that $(\mathcal{B}, f)$ is a negative instance of the PCP if and only if $\mathcal{A}$ is explorable.

We choose $Q_{\mathcal{B}} = (Q_{\mathcal{A}} \times \mathcal{P}(Q_{\mathcal{A}})) \uplus \{f, \bot\}$, where $f, \bot$ are fresh sink states. The alphabet of $\mathcal{B}$ will be $\Sigma_{\mathcal{B}} = \Sigma \uplus \{a_{\text{test}}\}$, where a_{test} is a fresh letter.

The initial state of $\mathcal{B}$ is $q_0^{\mathcal{B}} = (q_0^{\mathcal{A}}, \{q_0^{\mathcal{A}}\})$. Notice that we do not need to specify accepting states in $\mathcal{B}$, as acceptance plays no role in the PCP.

We finally define the transitions of $\mathcal{B}$ in the following way:

- $(p, X) \xrightarrow{a} (q, \Delta_{\mathcal{A}}(X, a))$ if $a \in \Sigma$ and $q \in \Delta_{\mathcal{A}}(p, a)$,
- $(p, X) \xrightarrow{a_{\text{test}}} f$ if $p \notin F_{\mathcal{A}}$ and $X \cap F_{\mathcal{A}} \neq \emptyset$.
- $(p, X) \xrightarrow{a_{\text{test}}} \bot$ otherwise.

We aim at proving the following Lemma:

▶ **Lemma 10.** *For any $k \in \mathbb{N}$, $\mathcal{A}$ is k-explorable if and only if Determiniser wins the k-population game on $(\mathcal{B}, f)$.*

Notice that as long as letters of Σ are played, the second component of states of $\mathcal{B}$ evolves deterministically and keeps track of the set of reachable states in $\mathcal{A}$. Moreover, the letter a_{test} also acts deterministically on $Q_{\mathcal{B}}$. Therefore, the only non-determinism to be resolved in $\mathcal{B}$ is how the first component evolves, which amounts to building a run in $\mathcal{A}$. Thus, strategies driving tokens in $\mathcal{A}$ and $\mathcal{B}$ are isomorphic. It now suffices to observe that Spoiler wins the k-population game on $(\mathcal{B}, f)$ if and only if he has a strategy allowing to eventually play a_{test} while all tokens are in a state of the form (q, X) with $q \notin F_{\mathcal{A}}$ and $X \cap F_{\mathcal{A}} \neq \emptyset$. This is equivalent to Spoiler winning the k-explorability game of $\mathcal{A}$, since $X \cap F_{\mathcal{A}} \neq \emptyset$ witnesses that the word played so far is in $L(\mathcal{A})$.

This concludes the proof that $\mathcal{A}$ is explorable if and only if $(\mathcal{B}, f)$ is a negative instance of the PCP. So given an NFA $\mathcal{A}$ that we want to test for explorability, it suffices to build $(\mathcal{B}, f)$ as above, and use the ExpTime algorithm from [4] as a black box on $(\mathcal{B}, f)$. Since $\mathcal{B}$ is of exponential size compared to $\mathcal{A}$, we obtain the following result:

▶ **Theorem 11.** *The NFA explorability problem is decidable and in 2-ExpTime.*

3.2 ExpTime-hardness of NFA explorability

We will perform here an encoding in the converse direction: starting from an instance $(\mathcal{B}, f)$ of the PCP, we build polynomially an NFA $\mathcal{A}$ such that $\mathcal{A}$ is explorable if and only if $(\mathcal{B}, f)$ is a negative instance of the PCP.

It is stated in [4] that, without loss of generality, we can assume that f is a sink state in $\mathcal{B}$, and we will use this assumption here.

Let $\mathcal{C}$ be the 4-state automaton of Example 2, that is non-explorable and accepts all words on alphabet $\Sigma_{\mathcal{C}} = \{a, b\}$. Recall that, as an instance of the PCP, $\mathcal{B}$ does not come with an acceptance condition. We can assume that its accepting set is $F_{\mathcal{B}} = Q_{\mathcal{B}} \setminus \{f\}$.

We will take for $\mathcal{A}$ the product automaton $\mathcal{B} \times \mathcal{C}$ on alphabet $\Sigma_A = \Sigma_{\mathcal{B}} \times \Sigma_{\mathcal{C}}$, with the union acceptance condition: $\mathcal{A}$ accepts whenever one of its components accepts. The transitions of $\mathcal{A}$ are defined as expected: $(p, p') \xrightarrow{a_1, a_2} (q, q')$ in $\mathcal{A}$ whenever $p \xrightarrow{a_1} q$ in $\mathcal{B}$ and $p' \xrightarrow{a_2} q'$ in $\mathcal{C}$.

Since $L(\mathcal{C}) = (\Sigma_{\mathcal{C}})^*$, we have $L(\mathcal{A}) = (\Sigma_{\mathcal{A}})^*$. The intuition for the role of $\mathcal{C}$ in this construction is the following: it allows us to modify $\mathcal{B}$ in order to accept all words, without interfering with its explorability status.

We claim that for any $k \in \mathbb{N}$, $\mathcal{A}$ is k-explorable if and only if Determiniser wins the k-population game on $(\mathcal{B}, f)$.

Assume that $\mathcal{A}$ is k-explorable, via a strategy σ. Then Determiniser can play in the k-population game on $(\mathcal{B}, f)$ using σ as a guide. In order to simulate σ, one must feed to it letters from $\Sigma_{\mathcal{C}}$ in addition to letters from $\Sigma_{\mathcal{B}}$ chosen by Spoiler. This is done by applying a winning strategy for Spoiler in the k-explorability game of $\mathcal{C}$. Assume for contradiction that, at some point, this strategy σ reaches a position where all tokens are in a state of the form (f, q) with $q \in Q_{\mathcal{C}}$. Since f is a sink state, when the play continues it will eventually reach a point where all tokens are in (f, q_3), where q_3 is the rejecting sink of $\mathcal{C}$. This is because we are playing letters from $\Sigma_{\mathcal{C}}$ according to a winning strategy for Spoiler in the k-explorability game of $\mathcal{C}$, and this strategy guarantees that all tokens eventually reach q_3 in $\mathcal{C}$. But this state (f, q_3) is rejecting in $\mathcal{A}$, and $L(\mathcal{A}) = (\Sigma_{\mathcal{A}})^*$, so this is a losing position for

Determiniser in the k-explorability game of $\mathcal{A}$. Since we assumed σ is a winning strategy in this game, we reach a contradiction. This means that following this strategy σ together with an appropriate choice for letters from $\Sigma_{\mathcal{C}}$, we guarantee that at least one token never reaches the sink state f on its $\mathcal{B}$-component. This corresponds to Determiniser winning in the k-population game on $(\mathcal{B}, f)$.

Conversely, assume that Determiniser wins in the k-population game on $(\mathcal{B}, f)$, via a strategy σ. The same strategy can be used in the k-explorability game of $\mathcal{A}$, by making arbitrary choices on the $\mathcal{C}$ component. As before, this corresponds to a winning strategy in the k-explorability game of $\mathcal{A}$, since there is always at least one token with $\mathcal{B}$-component in $F_{\mathcal{B}} = Q_{\mathcal{B}} \setminus \{f\}$. This completes the hardness reduction, and allows us to conclude:

▶ **Theorem 12.** *The NFA explorability problem is ExpTime-hard.*

▶ Remark 13. Using standard arguments, it is straightforward to extend Theorem 12 to ExpTime-hardness of explorability for automata on infinite words, using any of the acceptance conditions defined in Section 2.1.

Let us give some intuition on why we can obtain a polynomial reduction in one direction, but did not manage to do so in the other direction. Intuitively, the explorability problem is "more difficult" than the PCP for the following reason. In the PCP, Spoiler is allowed to play any letters, and the winning condition just depends on the current position. On the contrary, the winning condition of the k-explorability game mentions that the word chosen by Spoiler must belong to the language of the NFA. In order to verify this, we a priori need to append to the arena an exponential deterministic automaton for this language, and this is what is done in Section 3.1. This complicated winning condition is also the source of difficulty of recognizing HD parity automata.

3.3 ExpTime algorithm for Büchi explorability

▶ **Theorem 14.** *The explorability problem can be solved in ExpTime for Büchi automata (and all simpler conditions: NFA, safety, reachability).*

Due to space constraints, we will only sketch the proof of Theorem 14 here. A more detailed account is given in the long version (Appendix A.2).

The algorithm is adapted from the ExpTime algorithm for the PCP from [4]. We will recall here the main ideas of this algorithm, and describe how we adapt it to our setting.

Let $\mathcal{A}$ be an NFA, together with a target state f. The idea in [4] is to abstract the population game with arbitrary many tokens by a game called the *capacity game*. This game allows Determiniser to describe only the support of his set of tokens, i.e. the set of states occupied by tokens. The sequence of states obtained in a play can be analyzed via a notion of *bounded capacity*, in order to detect whether it actually corresponds to a play with finitely many tokens. This notion can be approximated by the more relaxed *finite capacity*, which is a regular property that is equivalent to bounded capacity in a context where games are finite-memory determined. This property of finite capacity can be verified by a deterministic parity automaton, yielding a parity game that can be won by Spoiler if and only if $(\mathcal{A}, f)$ is a positive instance of the PCP. Since this parity game has size exponential in $\mathcal{A}$, this yields an ExpTime algorithm for the PCP.

Here, we will perform the following tweaks to this construction. We now start with a Büchi automaton $\mathcal{A}$, and want to decide whether it is explorable.

First, we need to control that the infinite word played by Spoiler is in $L(\mathcal{A})$. This requires to build a deterministic parity automaton $\mathcal{D}$ recognising $L(\mathcal{A})$, and incorporate it into the arena. The size of $\mathcal{D}$ is exponential with respect to $\mathcal{A}$. We then follow [4] and build the

capacity game augmented with $\mathcal{D}$. This time, a sequence of supports is winning if infinitely many of them contain an accepting state. We emphasize that we use here a particularity of the Büchi condition: observing the sequences of support sets of tokens is enough to decide whether one of the tokens follows an accepting run. The same particularity was used in [3], and was a crucial tool allowing to give a PTime algorithm for Büchi HDness. Since this modification still allows us to manipulate supports as simple sets, we can make use of the capacity game as before. See the long version, Appendix A.2, Remark 40 for an example showing that a naive adaptation of this construction to co-Büchi automata would not be correct.

Finally, we show that we can as in [4] obtain a parity game of exponential size characterizing explorability of $\mathcal{A}$, yielding the wanted ExpTime algorithm.

We also remark that, as in [4], this construction gives a doubly exponential upper bound on the number of tokens needed to witness explorability. Moreover, the proof from [4] that this is tight also stands here.

4 Explorability with countably many tokens

In this section, we look at the same problem of explorability of an automaton, but we now allow for infinitely many tokens. More precisely, we will redefine the explorability game to allow an arbitrary cardinal for the number of tokens, then consider decidability problems regarding that game. This notion will mainly be interesting for automata on infinite words.

4.1 Definition and basic results

The following definition extends the notion of k-explorability to non-integer cardinals:

▶ **Definition 15** (κ-explorability game). *Consider an automaton $\mathcal{A}$ and a cardinal κ. The κ-explorability game on $\mathcal{A}$ is played on the arena $(Q_\mathcal{A})^\kappa$, between Determiniser and Spoiler. They play as follows.*

- *The initial position is S_0 associating q_0 to all κ tokens.*
- *At step i, from position S_{i-1}, Spoiler chooses a letter $a_i \in \Sigma$, and Determiniser chooses S_i such that for every token α, $S_{i-1}(\alpha) \xrightarrow{a_i} S_i(\alpha)$ is a transition in $\mathcal{A}$.*

The play is won by Determiniser if for all $\beta \leq \omega$ such that the word $(a_i)_{1\leq i<\beta}$ is in $\mathcal{L}(\mathcal{A})$, there is a token $\alpha \in \kappa$ building an accepting run, meaning that the sequence $(S_i(\alpha))_{i<\beta}$ is an accepting run. Otherwise, the winner is Spoiler.

We will say in particular that $\mathcal{A}$ is ω-explorable if Determiniser wins the game with ω tokens. We use here the notation ω for convenience, it should be understood as the countably infinite cardinal $\aleph_0$. We will however explicitly use the fact that such an amount of tokens can be labelled by $\mathbb{N}$, in order to describe strategies for Spoiler or Determiniser in the ω-explorability game. The following lemma gives a first few results on generalized explorability.

▶ **Lemma 16.**

- *Determiniser wins the explorability game on $\mathcal{A}$ with $|\mathcal{L}(\mathcal{A})|$ tokens.*
- *ω-explorability is not equivalent to explorability.*
- *There are non ω-explorable safety automata.*

Proof. For the first item, a strategy for Determiniser is to associate a token to each word of $\mathcal{L}(\mathcal{A})$ and to have it follow an accepting run for that word. Let us add a few details on the cardinality of $L(\mathcal{A})$. First, a dichotomy result has been shown in [22] (even in the

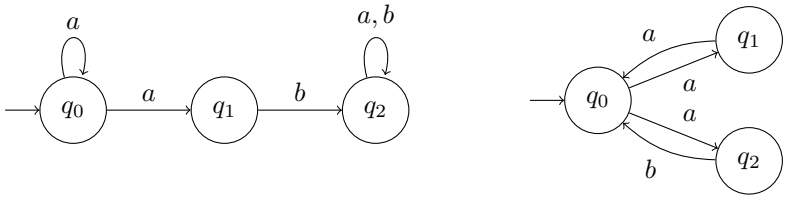


Figure 3 Two safety automata. Left: ω-explorable but not explorable. Right: not ω-explorable.

more general case of infinite trees): if $L(\mathcal{A})$ is not countable, then it has the cardinality of continuum, and this happens if and only if $L(\mathcal{A})$ contains a non ultimately periodic word. In this case, we can simply associate a token with every possible run. In the other case where $L(\mathcal{A})$ is countable, we have to associate an accepting run to each word, and this can be done without needing the Axiom of Countable Choice: a canonical run can be selected (e.g. lexicographically minimal).

We now want to prove that there are automata that are ω-explorable but not explorable. One such automaton is given in Figure 3 (left), where the rejecting sink state is omitted. Against any finite number of tokens, Spoiler has a strategy to eliminate them one by one, by playing a while Determiniser sends tokens to q_1, and b the first time q_1 is empty after the play of Determiniser. On the other hand, with tokens indexed by ω, Determiniser can keep the token 0 in q_0, and send token i to q_1 at step i. Those strategies are winning, which proves both non explorability and ω-explorability of the automaton.

The last item is proven by the second example from Figure 3. A winning strategy for Spoiler against countable tokens consists in labelling the tokens with integers, then targeting each token one by one (first token 0, then 1, 2, etc.). Each token is removed using the correct two-letters sequence (a, then b if the token is in q_1 or a if it is in q_2). With this strategy, every token is removed at some point, even if there might always be tokens in the game. ◀

The first item of Lemma 16 implies that the ω-explorability game only gets interesting when we look at automata over infinite words: since any language of finite words over a finite alphabet is countable, Determiniser wins the corresponding ω-explorability game. We will therefore focus on infinite words in the following.

Let us emphasize the following slightly counter-intuitive fact: in the ω-explorability game, it is always possible for Determiniser to guarantee that infinitely many tokens occupy each currently reachable state. However, even in a safety automaton, this is not enough to win the game, as it does not prevent that each individual token might be eventually "killed" at some point. As the following Lemma shows, this phenomenon does not occur in reachability automata on infinite words.

▶ **Lemma 17.** *Any reachability automaton is ω-explorable.*

Proof. Notice first that although the argument is very similar to the one for finite words, we cannot use exactly the same property: a reachability language can still be uncountable, so using one token per word of the language is not possible.

For every $w \in \Sigma^*$ such that there is a finite run ρ leading to an accepting state, Determiniser can use a single token following ρ. This token will accept all words of $w \cdot \Sigma^\omega$. Since Σ^* is countable, we only need countably many such tokens to cover the whole language, hence the result.

Let us give another equally simple view: a winning strategy for Determiniser in the ω-explorability game is to keep infinitely many tokens in each currently reachable state, as described above. Since acceptance in a reachability automaton is witnessed at a finite time, this strategy is winning. ◀

4.2 ExpTime algorithm for co-Büchi automata

We already know, from the example of Figure 3, that the result from Lemma 17 does not hold in the case of safety automata. However, we have the following decidability result, which talks about co-Büchi automata, and therefore still holds for safety automata as a subclass of co-Büchi.

▶ **Theorem 18.** *The ω-explorability of co-Büchi automata is decidable in ExpTime.*

To prove this result, we will use the following *elimination game*. $\mathcal{A}$ will from here on correspond to a co-Büchi (complete) automaton. We start by building a deterministic co-Büchi automaton $\mathcal{D}$ for $L(\mathcal{A})$ (e.g. using the breakpoint construction [21]).

▶ **Definition 19** (Elimination game)**.** *The elimination game is played on the arena $\mathcal{P}(Q_\mathcal{A}) \times Q_\mathcal{A} \times Q_\mathcal{D}$. The two players are named Protector and Eliminator, and the game proceeds as follows, starting in the position $(\{q_0^\mathcal{A}\}, q_0^\mathcal{A}, q_0^\mathcal{D})$.*

- *From position (B, q, p) Eliminator chooses a letter $a \in \Sigma$.*
- *If q is not a co-Büchi state, Protector picks a state $q' \in \Delta_\mathcal{A}(q, a)$.*
- *If q is a co-Büchi state, Protector picks any state $q' \in \Delta_\mathcal{A}(B, a)$. Such an event is called* elimination*.*
- *The play moves to position $(\Delta_\mathcal{A}(B, a), q', \delta_\mathcal{D}(p, a))$.*

Such a play can be written $(B_0, q_0, p_0) \xrightarrow{a_1} (B_1, q_1, p_1) \xrightarrow{a_2} (B_2, q_2, p_2) \ldots$, and Eliminator wins if infinitely many q_i and finitely many p_i are co-Büchi states.

Intuitively, what is happening in this game is that Protector is placing a token that he wants to protect in a reachable state, and Eliminator aims at bringing that token to a co-Büchi state while playing a word of $L(\mathcal{A})$. If Protector eventually manages to preserve his token from elimination on an infinite suffix of the play, he wins.

▶ **Lemma 20.** *The elimination game can be solved in polynomial time (in the size of the game).*

Proof. To prove this result, we simply need to note that the winning condition is a parity condition of fixed index. If we label the co-Büchi states p_i with rank 1, the co-Büchi states q_i with rank 2, and the others with 3, then take the lowest rank in (B_i, q_i, p_i) (ignoring B_i), Eliminator wins if and only if the inferior limit of ranks is even. As any parity game with 3 ranks can be solved in polynomial time [10], this is enough to get the result. ◀

We want to prove the equivalence between this game and the ω-explorability game to obtain Theorem 18.

▶ **Lemma 21.** *$\mathcal{A}$ is ω-explorable if and only if Protector wins the elimination game on $\mathcal{A}$.*

Proof. First, let us suppose that Eliminator wins the elimination game on $\mathcal{A}$. To build a strategy for Spoiler in the ω-explorability game of $\mathcal{A}$, we first take a function $f : \mathbb{N} \to \mathbb{N}$ such that for any $n \in \mathbb{N}$, $|f^{-1}(n)|$ is infinite (for instance f is described by the sequence $0,\ 0,1,\ 0,1,2,\ 0,1,2,3,\ldots$). The strategy for Spoiler will focus on sending token $f(0)$, then $f(1)$, then $f(2)$, etc. to a co-Büchi state.

Let σ be a memoryless winning strategy for Eliminator in the elimination game (recall that parity games do not require memory [14]). Spoiler will follow this strategy σ in the ω-explorability game, by keeping an imaginary play of the elimination game in his memory: $M = \mathcal{P}(Q_\mathcal{A}) \times Q_\mathcal{A} \times Q_\mathcal{D} \times \mathbb{N}$.

- At first, the memory holds the initial state $(\{q_0^{\mathcal{A}}\}, q_0^{\mathcal{A}}, q_0^{\mathcal{D}}, 0)$, and the current target is given by the last component: it is the token $f(0)$.
- From (B, q, p, n) Spoiler plays in both games the letter a given by $\sigma(B, q, p)$.
- Once Determiniser has played, Spoiler moves the memory to $(\Delta_{\mathcal{A}}(B, a), q', \delta_{\mathcal{D}}(p, a), n)$ where q' is the new position of the token $f(n)$, except if q was a co-Büchi state, in which case we move to $(\Delta_{\mathcal{A}}(B, a), q', \delta_{\mathcal{D}}(p, a), n + 1)$ where q' is the new position of the token $f(n + 1)$. We then go back to the previous step.

This strategy builds a play of the elimination game in the memory, that is consistent with σ. We know that σ is winning, which implies that the word played is in $\mathcal{L}(\mathcal{A})$, and that every $n \in \mathbb{N}$ is visited (each elimination increments n, and there are infinitely many of those). An elimination happening while the target is the token $f(n)$ corresponds, on the exploration game, to that token visiting a co-Büchi state. Ultimately this means that Determiniser did not provide any accepting run, while Spoiler did play a word from $\mathcal{L}(\mathcal{A})$, and therefore won.

Let us now consider the situation where Protector wins the elimination game, using some strategy τ. We want to build a winning strategy for Determiniser in the ω-explorability game. Similarly, this strategy will keep track of a play in the elimination game in its memory. Determiniser will maintain ω tokens in any reachable state, while focusing on a particular token which follows the path of the current target in the elimination game. When that token visits a co-Büchi state, we switch to the new token specified by τ.

Since τ is winning in the elimination game, either the word played by Spoiler is not in $\mathcal{L}(\mathcal{A})$, which ensures a win for Determiniser, or there are no eliminations after some point, meaning that the target token at that point never visits another co-Büchi state, which also implies that Determiniser wins. ◀

With Lemmas 20 and 21 we get a proof of Theorem 18, since the elimination game associated to $\mathcal{A}$ is of exponential size and can be built using exponential time.

4.3 ExpTime-hardness of the ω-explorability problem

▶ **Theorem 22.** *The ω-explorability problem for (any automaton model embedding) safety automata is ExpTime-hard.*

We give a quick summary of the proof in this section. The full proof can be found in the long version (Appendix A.3). The main idea will be to reduce the acceptance problem of a PSpace alternating Turing machine (ATM) to the ω-explorability problem of some automaton that we build from the machine. This reduction is an adaptation of the one from [4] showing ExpTime-hardness of the NFA population control problem (defined in Section 3.1).

The computation of an ATM can be seen as a game between two players, who respectively aim for acceptance and rejection of the input. These players influence the output by choosing the transitions when facing a non-deterministic choice, that can belong to either one of them.

Let us first describe the automaton built in [4]. In that reduction, the choices made by the ATM players are translated into choices for Determiniser and Spoiler. The automaton has two main blocks: one dedicated to keeping track of the machine's configuration, which we call Config, and another focusing on the simulation of the ATM choices, which we call Choices. In Config, there is no non-determinism: the tokens move following the transitions of the machine given as input to the automaton. In Choices, Determiniser can pick a transition by sending his token to the corresponding state, while Spoiler uses letters to pick his.

The automaton constructed this way will basically read a sequence of runs of the ATM. At each run, some tokens must be sent into both blocks. Reaching an accepting state of a run lets Spoiler send some tokens from Choices to his target state, specifically those whose

choices for the transitions of the ATM were followed. He can then restart with the remaining tokens until all are in the target. This process will ensure a win for Spoiler if he has a winning strategy in the ATM game. If he does not, then Determiniser can use a strategy ensuring rejection in the ATM game to avoid the configurations where he loses tokens, provided he starts with enough tokens.

This equivalence between acceptance of the ATM and the automaton being a positive instance of the PCP provides the ExpTime-hardness of their problem.

In our setup, getting rid of tokens one by one is not enough: Spoiler needs to be able to target a specific token and send it to the target state (which is now the rejecting state $\bot$) in one run. If he can do that, repeating the process for every token, without omitting any, ensures his win. If he cannot, then Determiniser has a strategy to pick a specific token and preserving it from $\bot$, and therefore wins.

This is why we adapt our reduction to allow Spoiler to target a specific token, no matter where it chooses to go. To do so, we change the transitions so that winning a run lets Spoiler additionally send every token from Config into $\bot$. With that and the fact that he can already target a token in Choices, we get a winning strategy for Spoiler when the ATM is accepting.

If the ATM is rejecting, Spoiler is still able to send some tokens to $\bot$, but he no longer has that targeting ability, which is how Determiniser is able to build a strategy preserving a specific token to win. To ensure the sustainability of this method, Determiniser needs to keep ω additional tokens following his designated token, so that he always has ω tokens to spread into the gadgets every time a new run starts.

Overall, we are able to compute in polynomial time from the ATM a safety automaton that is ω-explorable if and only if the ATM rejects its input. Since acceptance of a polynomial space ATM is known to be ExpTime-hard, we obtain Theorem 22.

Conclusion

We introduced and studied the notions of explorability and ω-explorability, for automata on finite and infinite words. We showed that these problems are ExpTime-complete for Büchi condition in the first case and co-Büchi condition in the second case.

It is plausible that these results could be generalized to higher parity conditions, for instance by replacing the notion of support set by Safra trees, but this is outside the scope of this paper, and we leave this investigation for further research.

We showed that the original motivation of using explorability to improve the current knowledge on the complexity of the HDness problem for all parity automata cannot be directly achieved, since deciding explorability is at least as hard as HDness. Although this is a negative result, we believe it to be of importance. Moreover, some contexts naturally yield explorable automata, such as [9] where it leads to a PTime algorithm deciding the HDness of quantitative LimInf and LimSup automata. More generally, explorability is a natural property in the study of degrees of nondeterminism, and this notion could be used in other contexts as a middle ground between deterministic and non-deterministic automata.

References

1 Bader Abu Radi and Orna Kupferman. Minimization and canonization of GFG transition-based automata. *CoRR*, abs/2106.06745, 2021.

2 Bader Abu Radi, Orna Kupferman, and Ofer Leshkowitz. A hierarchy of nondeterminism. In *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

3 Marc Bagnol and Denis Kuperberg. Büchi good-for-games automata are efficiently recognizable. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

4 Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, Hugo Gimbert, and Adwait Amit Godbole. Controlling a population. *Log. Methods Comput. Sci.*, 15(3), 2019.

5 Udi Boker. Between deterministic and nondeterministic quantitative automata (invited talk). In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

6 Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Automata, Languages, and Programming – 40th International Colloquium, ICALP 2013*, Lecture Notes in Computer Science. Springer, 2013.

7 Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michał Skrzypczak. On the succinctness of alternating parity good-for-games automata. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

8 Udi Boker and Karoliina Lehtinen. History determinism vs. good for gameness in quantitative automata. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

9 Udi Boker and Karoliina Lehtinen. Token games and history-deterministic quantitative automata. In *Foundations of Software Science and Computation Structures – 25th International Conference, FOSSACS 2022*, Lecture Notes in Computer Science. Springer, 2022.

10 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 252–263. ACM, 2017.

11 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150, Berlin, 2009. Springer.

12 Thomas Colcombet. Forms of determinism for automata (invited talk). In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

13 Thomas Colcombet and Nathanaël Fijalkow. Universal graphs and good for games automata: New tools for infinite duration games. In *Foundations of Software Science and Computation Structures – 22nd International Conference, FOSSACS 2019*, Lecture Notes in Computer Science. Springer, 2019.

14 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991.

15 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

16 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *Computer Science Logic, 20th International Workshop, CSL 2006*, 2006.

17 Juraj Hromkovic, Juhani Karhumäki, Hartmut Klauck, Georg Schnitger, and Sebastian Seibert. Measures of nondeterminism in finite automata. *Electronic Colloquium on Computational Complexity (ECCC)*, 7, January 2000.

18 Denis Kuperberg and Anirban Majumdar. Computing the width of non-deterministic automata. *Log. Methods Comput. Sci. (LMCS)*, 15(4), 2019.

19 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015*, Lecture Notes in Computer Science. Springer, 2015.

20 Karoliina Lehtinen and Martin Zimmermann. Good-for-games ω-pushdown automata. In *LICS 2020: 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 689–702. ACM, 2020.

21 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoret. Comput. Sci.*, 32(3):321–330, 1984.

22 Damian Niwiński. On the cardinality of sets of infinite trees recognizable by finite automata. In *Mathematical Foundations of Computer Science 1991, 16th International Symposium, MFCS'91*, Lecture Notes in Computer Science. Springer, 1991.

23 Alexandros Palioudakis, Kai Salomaa, and Selim G. Akl. Worst case branching and other measures of nondeterminism. *Int. J. Found. Comput. Sci.*, 28(3):195–210, 2017.

24 Sven Schewe. Minimising good-for-games automata is NP-complete. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

25 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theor. Comput. Sci.*, 88(2):325–349, 1991.

The Expressive Power of CSP-Quantifiers

Lauri Hella
Faculty of Information Technology and Communication Sciences, Tampere University, Finland

Abstract

A generalized quantifier $Q_{\mathcal{K}}$ is called a CSP-quantifier if its defining class $\mathcal{K}$ consists of all structures that can be homomorphically mapped to a fixed finite template structure. For all positive integers $n \geq 2$ and k, we define a pebble game that characterizes equivalence of structures with respect to the logic $L^k_{\infty\omega}(\mathbf{CSP}^+_n)$, where $\mathbf{CSP}^+_n$ is the union of the class $\mathbf{Q}_1$ of all unary quantifiers and the class $\mathbf{CSP}_n$ of all CSP-quantifiers with template structures that have at most n elements. Using these games we prove that for every $n \geq 2$ there exists a CSP-quantifier with template of size $n+1$ which is not definable in $L^\omega_{\infty\omega}(\mathbf{CSP}^+_n)$. The proof of this result is based on a new variation of the well-known Cai-Fürer-Immerman construction.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases generalized quantifiers, constraint satisfaction problems, pebble games, finite variable logics, descriptive complexity theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.25

1 Introduction

The present paper continues the research line in descriptive complexity theory that was originated in the seminal paper [3] of Cai, Fürer and Immerman. Using a pebble game characterization, introduced in [11], for the infinitary k-variable logic with counting, $C^k_{\infty\omega}$, they proved that there are PTIME-computable properties that are not definable in $C^\omega_{\infty\omega} = \bigcup_{k\in\omega} C^k_{\infty\omega}$. Since fixed point logic with counting, IFPC, is contained in $C^\omega_{\infty\omega}$, they obtained as a corollary that IFPC does not capture PTIME on finite unordered structures. However, perhaps the most important contribution in [3] is the clever technique for constructing pairs $(\mathfrak{A}_k, \mathfrak{B}_k)$ of non-isomorphic structures such that Duplicator has a winning strategy in the pebble game with k pebbles. This *Cai-Fürer-Immerman* (CFI) construction has been later adapted for several different pebble games that characterize various extensions of $C^\omega_{\infty\omega}$.

The counting logic $C^k_{\infty\omega}$ is obtained by adding the counting quantifiers $\exists^{\geq m}x$ ("there are at least m elements x such that") to the corresponding infinitary k-variable logic $L^k_{\infty\omega}$. The counting quantifiers are examples of unary generalized quantifiers as they bind a single variable in the formula that follows. Generalized quantifiers can also bind variables in several formulas simultaneously, as well as several variables in each of the formulas. An example of the former is the Härtig quantifier $I\,x,y\,(\varphi(x),\psi(y))$ stating that "the number of x satisfying φ is the same as the number of y satisfying ψ". As an example of the latter, if $\mathcal{C}$ is the class of connected graphs, then $Q_{\mathcal{C}}$ is a generalized quantifier binding two variables in a formula, and $Q_{\mathcal{C}}xy\,\varphi(x,y)$ has the meaning "the binary relation defined by the formula $\varphi(x,y)$ is the edge relation of a connected graph". More generally, any isomorphism closed class $\mathcal{K}$ of structures in a finite relational vocabulary can be used as an interpretation of a generalized quantifier $Q_{\mathcal{K}}$. The quantifier $Q_{\mathcal{K}}$ is r-ary, if it binds at most r variables in each formula.

Since $C^\omega_{\infty\omega}$ turned out to be too weak to define all PTIME computable properties of finite structures, it was natural to study the expressive power of extensions of $L^\omega_{\infty\omega}$ with quantifiers of arity more than 1. For this purpose we introduced in [10] the *r-bijective k-pebble game* that characterizes equivalence with respect to the logic $L^k_{\infty\omega}(\mathbf{Q}_r)$, the extension of $L^k_{\infty\omega}$ by the class $\mathbf{Q}_r$ of all generalized quantifiers of arity at most r. Furthermore, we proved that, for any positive integer r, there is a PTIME-computable generalized quantifier of arity

$r+1$ which is not definable in $L^{\omega}_{\infty\omega}(\mathbf{Q}_r)$. As a corollary, we got the result that there is no set $\mathbf{Q}$ of generalized quantifiers of bounded arity such that IFP($\mathbf{Q}$) captures PTIME. The proof of these results is based on the n-bijective k-pebble game and a variation of the CFI construction, where the binary edge relation in the so-called "gadget graphs" is replaced by an $r+1$-ary relation. Since $C^{\omega}_{\infty\omega}$ has the same expressive power as $L^{\omega}_{\infty\omega}(\mathbf{Q}_1)$, the results in [10] are a natural extension of those in [3].

More than a decade after the publication of [10], research on extensions of $C^{\omega}_{\infty\omega}$, pebble games and CFI constructions became active again, when Dawar, Grohe, Holm and Laubner [5] introduced the extension IFPR of IFP by rank operators. A rank operator rk_p binds two tuples $\vec{x}$ and $\vec{y}$ of variables in a formula φ, and outputs the rank of the matrix defined by φ over the field F_p, where values of $\vec{x}$ are thought as columns and values of $\vec{y}$ as rows. Dawar and Holm [6] defined a pebble game that characterizes equivalence with respect to the extension of $L^{k}_{\infty\omega}$ with the rank operators, and also the so-called invertible map game, which corresponds to a stronger equivalence. The invertible map game was later shown in [4] to characterize equivalence with respect to the extension of $L^{k}_{\infty\omega}$ by all linear algebraic operators, i.e., operators that are invariant under similarity mappings (with respect to the field F_p considered).

The rank logic IFPR was shown to fall short of capturing PTIME by Grädel and Pakusa [9]; their proof is based on a variation of the CFI construction. They also suggested a stronger version IFPR* of rank logic that allows the parameter p in the operator rk_p to be given by a term. The question whether IFPR* captures PTIME was open for a few years, but it was recently settled in the negative by Lichter ([14]), who used the invertible map game mentioned above to show that a generalized CFI construction produces non-isomorphic structures that cannot be separated in the extension of $L^{k}_{\infty\omega}$ with the stronger rank operator of [9].

Another application of the CFI construction was given by Atserias, Bulatov and Dawar [1], who observed that the CFI structures $\mathfrak{A}_k$ and $\mathfrak{B}_k$ of [3] can be separated by PTIME-computable constraint satisfaction problems that arise from affine systems of equations. The same is true for the CFI construction in [10]: for each r there is a constraint satisfaction problem r-CFI that separates the $L^{k}_{\infty\omega}(\mathbf{Q}_r)$-equivalent structures obtained by the construction. The problem r-CFI is an instance of solving systems of equations over F_2, and hence it is PTIME-computable.

A constraint satisfaction problem $\mathsf{CSP}(\mathfrak{C})$ with a finite template structure $\mathfrak{C}$ is the class of all finite structures in the same vocabulary as $\mathfrak{C}$ that can be homomorphically mapped to $\mathfrak{C}$. By a CSP-quantifier we mean a generalized quantifier $Q_{\mathcal{K}}$ such that its defining class $\mathcal{K}$ is of the form $\mathsf{CSP}(\mathfrak{C})$. In this paper we use two numerical parameters for classifying CSP-quantifiers: the *arity* of $Q_{\mathsf{CSP}(\mathfrak{C})}$ is the maximum arity of relations in $\mathfrak{C}$, and the *size* of $Q_{\mathsf{CSP}(\mathfrak{C})}$ is the number of elements in the universe of $\mathfrak{C}$. Thus, the observation above concerning the CFI structures in [10] can be formulated as an *arity hierarchy* for CSP-quantifiers: for any positive integer r, there exists a PTIME-computable CSP-quantifier $Q_{\mathsf{CSP}(\mathfrak{C})}$ of arity $r+1$ and size 2 which is not definable in $L^{\omega}_{\infty\omega}(\mathbf{Q}_r)$, and a fortiori, not definable in the extension of $L^{\omega}_{\infty\omega}$ by all CSP-quantifiers of arity at most r.

In this paper, we turn attention to the size of CSP-quantifers. Given an integer $n \geq 2$, we define $\mathbf{CSP}_n$ to be the class of all CSP-quantifiers of size at most n. We introduce a pebble game CSPG$[\mathfrak{A}, \mathfrak{B}, n, k]$, and prove that it characterizes equivalence of the structures $\mathfrak{A}$ and $\mathfrak{B}$ with respect to $L^{k}_{\infty\omega}(\mathbf{CSP}^{+}_{n})$, where $\mathbf{CSP}^{+}_{n}$ is the union of $\mathbf{CSP}_n$ and the class of all unary quantifiers $\mathbf{Q}_1$. We also define another pebble game that we call *bijective colouring game*, BCG$[\mathfrak{A}, \mathfrak{B}, n, k]$. The game BCG has simpler rules than CSPG, but it can still be used for proving undefinability results for $L^{\omega}_{\infty\omega}(\mathbf{CSP}^{+}_{n})$: we prove that if Duplicator has a winning strategy in BCG$[\mathfrak{A}, \mathfrak{B}, n, k]$, then she has one also in CSPG$[\mathfrak{A}, \mathfrak{B}, n, k]$.

As our main result in the paper, we prove a *size hierarchy* theorem for CSP-quantifiers: for any $n \geq 2$ there exists a CSP-quantifier $Q_{\mathsf{CSP}(\mathfrak{C}_n)}$ of size $n+1$ which is not definable in $L^{\omega}_{\infty\omega}(\mathbf{CSP}^+_n)$. The proof of this result is based on a new variation of the CFI construction that produces for each input graph G two structures $\mathfrak{A}^{\mathrm{ev}}_n(G)$ and $\mathfrak{A}^{\mathrm{od}}_n(G)$ such that Duplicator has a winning strategy in the bijective colouring game $\mathrm{BCG}[\mathfrak{A}^{\mathrm{ev}}_n(G), \mathfrak{A}^{\mathrm{od}}_n(G), n, k]$, assuming that G is large enough. On the other hand, $\mathfrak{A}^{\mathrm{ev}}_n(G) \in \mathsf{CSP}(\mathfrak{C}_n)$ and $\mathfrak{A}^{\mathrm{od}}_n(G) \notin \mathsf{CSP}(\mathfrak{C}_n)$ for all graphs G.

The size hierarchy result differs from the arity hierarchy result in two important aspects. First, the membership problem of $\mathsf{CSP}(\mathfrak{C}_n)$ is NP-complete. It is an open problem whether $\mathsf{CSP}(\mathfrak{C}_n)$ can be replaced in the size hierarchy result by some PTIME-computable $\mathsf{CSP}(\mathfrak{D}_n)$. Second, the arity of $\mathsf{CSP}(\mathfrak{C}_n)$ depends on n; in fact, $\mathfrak{C}_n$ has a single relation which is $3n$-ary. We do not know if the size hierarchy result holds for CSP-quantifiers of arity r for some fixed r. It should also be mentioned here that we do not include vectorization in the definition of logics with generalized quantifiers, unlike is done in many recent papers on the topic. The pebble game CSPG could be adapted for vectorized quantifiers, but using such games would probably be too difficult to handle, as they involve existential second order quantification of ℓ-ary relations for $\ell \geq 2$. Furthermore, it seems quite possible that equivalence with respect to $L^k_{\infty\omega}$ with the second vectorization of CSP-quantifiers is just isomorphism.

The structure of the paper is the following. We explain the necessary background on logics, constraint satisfaction problems and generalized quantifiers in Sections 2 and 3. Section 4 is devoted to the definitions of the pebble games CSPG and BCG, and the corresponding characterizations of equivalence with respect to the logic $L^k_{\infty\omega}(\mathbf{CSP}^+_n)$. In Section 5, we describe our generalized CFI construction, and in Section 6, we introduce the CSP-quantifer that separates the constructed structures $\mathfrak{A}^{\mathrm{ev}}_n(G)$ and $\mathfrak{A}^{\mathrm{od}}_n(G)$. The proof that Duplicator has a winning strategy in the game $\mathrm{BCG}[\mathfrak{A}^{\mathrm{ev}}_n(G), \mathfrak{A}^{\mathrm{od}}_n(G), n, k]$ is given in Section 7. Finally, in Section 8, we give a brief overview of our contributions in the paper, and list some open problems.

2 Preliminaries

In this section we go briefly through definitions of basic notions concerning logics and constraint satisfaction problems. For a more comprehensive exposition on first-order logic and finite variable logic, we refer to the excellent textbooks [7] and [13]. Fo more information on constraint satisfaction problems from a logical point of view we refer to [12]. We start by giving some notational and other conventions.

2.1 Notation and conventions

For a positive integer n, we denote the set $\{1, \ldots, n\}$ by $[n]$. The cardinality of a (finite) set A is denoted by $|A|$. If $f\colon A \to B$ is a function, and $\vec{a} = (a_1, \ldots, a_r) \in A^r$, then we use the shorthand notation $f(\vec{a}) := (f(a_1), \ldots, f(a_r))$.

All vocabularies considered in this paper are *finite* and *relational*, i.e., they are of the form $\tau = \{R_1, \ldots, R_n\}$, where each R_i is a relation symbol. The arity of R_i is denoted by $\mathrm{ar}(R_i)$. A τ-structure is then a tuple $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \ldots, R_n^{\mathfrak{A}})$, where $R_i^{\mathfrak{A}} \subseteq A^{\mathrm{ar}(R_i)}$ for each $i \in [n]$. All our results are in the context of finite model theory; thus, we will assume without further notice that the universe A of any structure $\mathfrak{A}$ we consider is a finite set.

Given a τ-structure $\mathfrak{A}$, an *assignment on* $\mathfrak{A}$ is a function $\alpha\colon X \to A$, where $\mathrm{dom}(\alpha) = X$ is some finite set of variables. If y is a variable and $a \in A$, then $\alpha[a/y]$ denotes the modified assignment on $\mathfrak{A}$ with $\mathrm{dom}(\alpha[a/y]) = X \cup \{y\}$ such that $\alpha[a/y](x) = \alpha(x)$ for all $x \in X \setminus \{y\}$, and $\alpha[a/y](y) = a$.

Let $\mathfrak{A}$ and $\mathfrak{B}$ be τ-structures. A *partial isomorphism* from $\mathfrak{A}$ to $\mathfrak{B}$ is a bijection $p\colon C \to D$ such that $C \subseteq A$, $D \subseteq B$, and for all $R \in \tau$ and all $\vec{a} \in C^{\mathrm{ar}(R)}$, $\vec{a} \in R^{\mathfrak{A}} \iff p(\vec{a}) \in R^{\mathfrak{B}}$. We denote the set of all partial isomorphisms from $\mathfrak{A}$ to $\mathfrak{B}$ by $\mathrm{PI}(\mathfrak{A}, \mathfrak{B})$. If α is an assignment on $\mathfrak{A}$ and β is an assignment on $\mathfrak{B}$ such that $\mathrm{dom}(\alpha) = \mathrm{dom}(\beta)$, then we denote the relation $\{(\alpha(x), \beta(x)) \mid x \in \mathrm{dom}(\alpha)\} \subseteq A \times B$ by $\alpha \mapsto \beta$. Thus, $\alpha \mapsto \beta \in \mathrm{PI}(\mathfrak{A}, \mathfrak{B})$ if this relation is a bijection that preserves the relations in τ.

An *n-colouring* of a set T is a function $g\colon T \to [n]$. If $\mathfrak{A}$ is a τ-structure and g is an n-colouring of its universe A, we define $\mathfrak{A}^g$ to be the $\tau \cup \{S_1, \ldots, S_n\}$-structure with the same universe such that $R^{\mathfrak{A}^g} := R^{\mathfrak{A}}$ for all $R \in \tau$, and $S_i^{\mathfrak{A}^g} := \{a \in A \mid g(a) = i\}$ for each $i \in [n]$. Note that if $\mathfrak{B}^h$ is another τ-structure with an n-colouring h, and $p \in \mathrm{PI}(\mathfrak{A}, \mathfrak{B})$, then $p \in \mathrm{PI}(\mathfrak{A}^g, \mathfrak{B}^h)$ if and only if p preserves colouring: $g(a) = h(p(a))$ for all $a \in \mathrm{dom}(p)$.

We define graphs as ordered pairs $G = (V, E)$, where V is a nonempty set of vertices, and E is a set of *unordered pairs* of vertices, called edges; i.e., $E \subseteq \{\{u, v\} \mid u, v \in V,\ u \neq v\}$. Thus, a graph G *is not* a relational structure, but it can be represented as one by setting $\mathfrak{A}_G := (V, R_E)$, where $R_E := \{(u, v) \mid \{u, v\} \in E\}$.

2.2 Logics

First-order logic, FO, is defined as usually (see, e.g., [13]). The infinitary logic $L_{\infty\omega}$ is the extension of FO that allows disjunctions $\bigvee \Psi$ and conjunctions $\bigwedge \Psi$ of arbitrary sets Ψ of formulas. Since we assume that all structures are finite, it suffices to consider infinite disjunctions and conjunctions only over countable sets Ψ.

For each positive integer k, the *infinitary k-variable logic* $L^k_{\infty\omega}$ is the fragment of $L_{\infty\omega}$ consisting of formulas that contain at most k different variables; we assume throughout that the variables allowed in the formulas of $L^k_{\infty\omega}$ come from a fixed set $X_k := \{x_1, \ldots, x_k\}$. The k-variable first-order logic, FO^k, is defined analogously. The *finite variable logic* $L^\omega_{\infty\omega}$ is the union of $L^k_{\infty\omega}$ over positive integers k.

All logics L we consider are extensions of FO, $L^\omega_{\infty\omega}$, or their k-variable fragments with generalized quantifiers. Given a τ-structure $\mathfrak{A}$, an assignment $\alpha\colon X \to A$ on $\mathfrak{A}$, and a τ-formula $\varphi \in L$ with free variables in X, we write $(\mathfrak{A}, \alpha) \models \varphi$, if φ is true in $\mathfrak{A}$ under the interpretation α. If φ is a sentence and α is the empty assignment $\emptyset$, we write $\mathfrak{A} \models \varphi$ instead of $(\mathfrak{A}, \emptyset) \models \varphi$.

2.3 Constraint satisfaction problems

Let τ be a vocabulary, and let $\mathfrak{A}$ and $\mathfrak{B}$ be τ-structures. A function $h\colon A \to B$ is a *homomorphism* $\mathfrak{A} \to \mathfrak{B}$, if the implication

$$\vec{a} \in R^{\mathfrak{A}} \implies h(\vec{a}) \in R^{\mathfrak{B}}$$

holds for every relation symbol $R \in \tau$ and every tuple $\vec{a} \in A^{\mathrm{ar}(R)}$. The *uniform* Constraint Satisfaction Problem (CSP) on τ-structures asks whether there exists a homomorphism $h\colon \mathfrak{A} \to \mathfrak{B}$ for a pair $(\mathfrak{A}, \mathfrak{B})$ of input structures of vocabulary τ. It is well known that this problem is NP-complete, assuming that τ contains at least one relation symbol R with $\mathrm{ar}(R) \geq 2$.

We consider in this paper the *non-uniform* version of CSP, in which the second structure, called the template of the problem, is fixed, and only the first structure is given as input. For each template structure $\mathfrak{C}$ of vocabulary τ, the class of positive instances of the corresponding non-uniform CSP is denoted by $\mathsf{CSP}(\mathfrak{C})$. Thus, $\mathsf{CSP}(\mathfrak{C})$ consists of all τ-structures $\mathfrak{A}$ such that there exists a homomorphism $h\colon \mathfrak{A} \to \mathfrak{C}$.

We classify template structures $\mathfrak{C}$ by two numerical parameters:

- The *arity* of $\mathfrak{C}$ is $\mathrm{ar}(\mathfrak{C}) := \max\{\mathrm{ar}(R) \mid R \in \tau\}$, where τ is the vocabulary of $\mathfrak{C}$.
- The *size* of $\mathfrak{C}$ is $\mathrm{sz}(\mathfrak{C}) := |C|$.

We will henceforth assume without loss of generality that the universe of any template $\mathfrak{C}$ with $\mathrm{sz}(\mathfrak{C}) = n \geq 3$ is $[n]$; in case $\mathrm{sz}(\mathfrak{C}) = 2$, we may also use the Boolean universe $\{0, 1\}$.

As an example, consider the n-colourability problem n-COL of graphs. Clearly a graph G is n-colourable if and only if there is a homomorphism from $\mathfrak{A}_G$ to the structure $\mathfrak{C}_{n\text{-}\mathsf{COL}} := ([n], \{(i, j) \in [n]^2 \mid i \neq j\})$. Thus, we can identify n-COL with $\mathsf{CSP}(\mathfrak{C}_{n\text{-}\mathsf{COL}})$. The arity and size of $\mathfrak{C}_{n\text{-}\mathsf{COL}}$ are 2 and n, respectively. It is well known that for $n \geq 3$, n-COL is NP-complete, while for $n = 2$ it is in $\mathsf{LOGSPACE}$.

Another, highly relevant example is related to the generalization of the CFI construction given in [10]: a close inspection of the structures $\mathbf{A}(\mathbf{G})$ and $\mathbf{B}(\mathbf{G})$ constructed in Section 8 of that paper reveals that they can be separated by a CSP. More precisely, $\mathbf{A}(\mathbf{G}) \in \mathsf{CSP}(\mathfrak{C}_{n\text{-}\mathsf{CFI}})$ and $\mathbf{B}(\mathbf{G}) \notin \mathsf{CSP}(\mathfrak{C}_{n\text{-}\mathsf{CFI}})$, where $\mathfrak{C}_{n\text{-}\mathsf{CFI}} = (\{0, 1\}, R^{\mathrm{ev}}, \{(0, 0), (1, 1)\})$ for $R^{\mathrm{ev}} := \{(b_1, \ldots, b_{n+1}) \mid b_1 + \cdots + b_{n+1} = 0 \mod 2\}$.[1] The arity and size of $\mathfrak{C}_{n\text{-}\mathsf{CFI}}$ are $n + 1$ and 2, respectively. The problem n-$\mathsf{CFI} := \mathsf{CSP}(\mathfrak{C}_{n\text{-}\mathsf{CFI}})$ can be solved by Gaussian elimination, whence it is PTIME-computable.

The complexity of non-uniform CSP has been studied intensively for more than three decades. This is because a large variety of real-world problems can be formulated as CSPs, and hence it is important to understand the borderline between feasible and unfeasible cases. Research on the topic culminated recently in the proof by Bulatov [2] and Zhuk [16] of the Dichotomy Conjecture formulated by Feder and Vardi in [8]: for any template $\mathfrak{C}$, $\mathsf{CSP}(\mathfrak{C})$ is either in PTIME, or NP-complete.

3 Generalized quantifiers

The notion of generalized quantifier was originally defined by Lindström [15] in 1966. Thus, generalized quantifiers are often called *Lindström quantifiers*. We go here quickly through the definitions and notations concerning generalized quantifiers, and then introduce the special case of CSP-quantifiers. For more detailed treatment of quantifiers we refer to [7] and [10].

Let $\tau = \{R_1, \ldots, R_m\}$ be a relational vocabulary, and let $\mathrm{ar}(R_i) = r_i$ for each $i \in [m]$. To any class $\mathcal{K}$ of τ-structures that is closed under isomorphisms, we assign a *generalized quantifier* (or a *Lindström quantifier*) $Q_{\mathcal{K}}$. The extension $L(Q_{\mathcal{K}})$ of a logic L by $Q_{\mathcal{K}}$ is obtained by adding the following rules in the syntax and semantics of L:

- If $\psi_1, \ldots, \psi_m$ are formulas and $\vec{y}_1, \ldots, \vec{y}_m$ are tuples of variables with $|\vec{y}_i| = r_i$ for $i \in [m]$, then $\varphi = Q_{\mathcal{K}}\vec{y}_1, \ldots, \vec{y}_m\,(\psi_1, \ldots, \psi_m)$ is a formula. A variable is free in φ if, for some $i \in [m]$, it is free in ψ_i but does not occur in $\vec{y}_i$.
- $(\mathfrak{A}, \alpha) \models Q_{\mathcal{K}}\vec{y}_1, \ldots, \vec{y}_m\,(\psi_1, \ldots, \psi_m)$ if and only if $(A, \psi_1^{\mathfrak{A},\alpha,\vec{y}_1}, \ldots, \psi_m^{\mathfrak{A},\alpha,\vec{y}_m}) \in \mathcal{K}$, where $\theta^{\mathfrak{A},\alpha,\vec{y}} := \{\vec{a} \in A^r \mid (\mathfrak{A}, \alpha[\vec{a}/\vec{y}]) \models \theta\}$ for a formula θ and an r-tuple $\vec{y}$ of variables.

The extension $L(\mathbf{Q})$ of L by a class $\mathbf{Q}$ of generalized quantifiers is defined by adding the rules above to L for each $Q_{\mathcal{K}} \in \mathbf{Q}$.

[1] This was observed in hindsight after the paper [1]. At the time we wrote [10] we were not familiar with the literature on CSP.

Let $Q_{\mathcal{K}}$ and $Q_{\mathcal{K}'}$ be generalized quantifiers. We say that $Q_{\mathcal{K}}$ is *definable* in $L(Q_{\mathcal{K}'})$ if the defining class $\mathcal{K}$ is definable in $L(Q_{\mathcal{K}'})$, i.e., there is a sentence φ of $L(Q_{\mathcal{K}'})$ such that $\mathcal{K} = \{\mathfrak{A} \mid \mathfrak{A} \models \varphi\}$.

The *type* of the quantifier $Q_{\mathcal{K}}$ is $(r_1, \ldots, r_m)$, and the *arity* of $Q_{\mathcal{K}}$ is $\max\{r_1, \ldots, r_m\}$. For the sake of simplicity, we assume from now onwards that the type of $Q_{\mathcal{K}}$ is *uniform*, i.e., $r_i = r_j$ for all $i, j \in [m]$. This is no loss of generality, since for any quantifier $Q_{\mathcal{K}}$ there is another quantifier $Q_{\mathcal{K}'}$ of uniform type with the same arity such that $Q_{\mathcal{K}}$ is definable in $\mathrm{FO}(Q_{\mathcal{K}'})$ and $Q_{\mathcal{K}'}$ is definable in $\mathrm{FO}(Q_{\mathcal{K}})$.

Furthermore, we restrict the syntactic rule of $Q_{\mathcal{K}}$ by requiring that $\vec{y}_i = \vec{y}_j$ for all $i, j \in [m]$. Then we can denote the formula obtained by applying the rule simply by $\varphi = Q_{\mathcal{K}}\vec{y}\,(\psi_1, \ldots, \psi_m)$. Note however, that this convention disallows formulas of the type $\theta = Qx, y\,(R(x, y), R(y, x))$ in which both x and y remain free even though x is bound in $R(x, y)$ and y is bound in $R(y, x)$, and hence weakens the expressive power of $\mathrm{FO}^k(Q_{\mathcal{K}})$ and $L^k_{\infty\omega}(Q_{\mathcal{K}})$. Fortunately the loss can be compensated by using more variables (e.g., θ is equivalent with $Qz\,(R(z, y), R(z, x))$. Hence the restriction does not affect the expressive power of $\mathrm{FO}(Q_{\mathcal{K}})$ and $L^\omega_{\infty\omega}(Q_{\mathcal{K}})$.

▶ **Definition 1.** *Let r and $n \geq 2$ be positive integers.*

(a) *We denote the class of all generalized quantifiers $Q_{\mathcal{K}}$ of arity at most r by $\mathbf{Q}_r$.*

(b) *A generalized quantifier $Q_{\mathcal{K}}$ is a* CSP-quantifier *if its defining class $\mathcal{K}$ is $\mathsf{CSP}(\mathfrak{C})$ for some template structure $\mathfrak{C}$. We will denote $Q_{\mathsf{CSP}(\mathfrak{C})}$ simply by $Q_{\mathfrak{C}}$.*

(c) *We denote the class of all CSP-quantifiers $Q_{\mathfrak{C}}$ such that $\mathrm{sz}(\mathfrak{C}) \leq n$ by $\mathbf{CSP}_n$.*

(d) *We write $\mathbf{CSP}^+_n := \mathbf{Q}_1 \cup \mathbf{CSP}_n$.*

For example $Q_{n\text{-}\mathsf{COL}} \in \mathbf{Q}_2 \cap \mathbf{CSP}_n$ for each $n \geq 2$, and $Q_{r\text{-}\mathsf{CFI}} \in \mathbf{Q}_{r+1} \cap \mathbf{CSP}_2$ for each r. As mentioned in the previous section, r-CFI is PTIME-computable, and it separates the structures used in the proof of the main result in [10] (Theorem 8.6). Thus, the corresponding CSP-quantifiers form a strong hierarchy: $Q_{r\text{-}\mathsf{CFI}}$ is not definable in $L^\omega_{\infty\omega}(\mathbf{Q}_r)$ for any $r \geq 2$.

In the present paper we will prove a similar hierarchy result for CSP-quantifiers in terms of the size parameter: for every $n \geq 2$ we define a template structure $\mathfrak{C}_n$ with $\mathrm{sz}(\mathfrak{C}_n) = n + 1$ such that $Q_{\mathfrak{C}_n}$ is not definable in $L^\omega_{\infty\omega}(\mathbf{CSP}^+_n)$. It is well known that *inflationary fixed point logic* IFP is contained in $L^\omega_{\infty\omega}$, and this remains valid in the presence of any additional quantifiers. Moreover, *inflationary fixed point logic with counting* IFPC is contained in $L^\omega_{\infty\omega}(\mathbf{Q}_1)$. Thus, as a corollary, we obtain that $Q_{\mathfrak{C}_n}$ is not definable in $\mathrm{IFPC}(\mathbf{CSP}_n)$.

This corollary is the main reason for considering $\mathbf{CSP}^+_n$ instead of just $\mathbf{CSP}_n$. Without the class $\mathbf{Q}_1$ of all unary quantifiers we would only get the undefinability of $Q_{\mathfrak{C}_n}$ in $\mathrm{IFP}(\mathbf{CSP}_n)$, and $\mathrm{IFP}(\mathbf{CSP}_n)$ is strictly less expressive than $\mathrm{IFPC}(\mathbf{CSP}_n)$. For example, using a straightforward quantifier elimination argument we can show that no sets A and B with at least kn elements can be separated by a sentence of $L^\omega_{\infty\omega}(\mathbf{CSP}_n)$; thus, e.g. even cardinality can not be defined in $\mathrm{IFP}(\mathbf{CSP}_n)$.

Given two structures $\mathfrak{A}$ and $\mathfrak{B}$ of the same vocabulary, and assignments α and β on $\mathfrak{A}$ and $\mathfrak{B}$, respectively, such that $\mathrm{dom}(\alpha) = \mathrm{dom}(\beta)$, we write $(\mathfrak{A}, \alpha) \equiv^k_{\infty\omega,n} (\mathfrak{B}, \beta)$ if the equivalence

$$(\mathfrak{A}, \alpha) \models \varphi \iff (\mathfrak{B}, \beta) \models \varphi$$

holds for all formulas $\varphi \in L^k_{\infty\omega}(\mathbf{CSP}^+_n)$ with free variables in $\mathrm{dom}(\alpha)$. Similarly we write $(\mathfrak{A}, \alpha) \equiv^k_n (\mathfrak{B}, \beta)$ if the equivalence above holds for all $\mathrm{FO}^k(\mathbf{CSP}^+_n)$-formulas φ. If $\alpha = \beta = \emptyset$, we write simply $\mathfrak{A} \equiv^k_{\infty\omega,n} \mathfrak{B}$ instead of $(\mathfrak{A}, \emptyset) \equiv^k_{\infty\omega,n} (\mathfrak{B}, \emptyset)$, and similarly for $\equiv^k_n$.

4 Pebble games

In order to prove undefinability results for the logic $L^k_{\infty\omega}(\mathbf{CSP}^+_n)$, we introduce in this section two pebble games. The first game gives an exact characterization for the equivalence $\equiv^k_{\infty\omega,n}$. The second game corresponds to an at least as strong equivalence, but has simpler rules. Hence we use the second game in the proof of the main result in Section 7.

4.1 Game for CSP-quantifiers

Assume that $\mathfrak{A}$ and $\mathfrak{B}$ are τ-structures for a relational vocabulary τ. Furthermore, assume that α and β are assignments on $\mathfrak{A}$ and $\mathfrak{B}$, respectively, such that $\mathrm{dom}(\alpha) = \mathrm{dom}(\beta) \subseteq X_k$ (recall that $X_k = \{x_1, \ldots, x_k\}$). The *CSP game* for $(\mathfrak{A}, \alpha)$ and $(\mathfrak{B}, \beta)$ is played between *Spoiler* and *Duplicator*, and it has two integer parameters: $n \geq 1$ for the size of templates and $k \geq 1$ for the number of pebbles. The parameters n and k are kept fixed in each play of the game. As the structures $\mathfrak{A}$ and $\mathfrak{B}$ are also fixed in plays, but the assignments α and β are changed, we denote the game by $\mathrm{CSPG}[\mathfrak{A}, \mathfrak{B}, n, k](\alpha, \beta)$, and we use the shorthand notation $\mathrm{CSPG}(\alpha, \beta)$ whenever $\mathfrak{A}$, $\mathfrak{B}$, n and k are clear from the context.

▶ **Definition 2.** *The rules of the game* $\mathrm{CSPG}[\mathfrak{A}, \mathfrak{B}, n, k](\alpha, \beta)$ *are the following:*

(1) *If* $\alpha \mapsto \beta \notin \mathrm{PI}(\mathfrak{A}, \mathfrak{B})$*, then the game ends, and Spoiler wins.*

(2) *If (1) does not hold, there are three types of moves that Spoiler can choose to play:*

- **Bijection move:** *Spoiler starts by choosing a variable* $y \in X_k$*.*
  Assuming that $|A| = |B|$*, Duplicator answers by choosing a bijection* $f\colon A \to B$*. Spoiler completes the round by choosing an element* $a \in A$*. The players continue by playing* $\mathrm{CSPG}(\alpha[a/y], \beta[f(a)/y])$*.*
  On the other hand, if $|A| \neq |B|$*, then the game ends, and Spoiler wins.*
- **Left CSP-quantifier move:** *Spoiler starts by choosing* $r \in [k]$ *and an* r*-tuple* $\vec{y} \in X^r_k$ *of distinct variables and a colouring* $g\colon A \to [n]$*. Duplicator chooses next a colouring* $h\colon B \to [n]$ *such that* $\mathrm{rng}(h) \subseteq \mathrm{rng}(g)$*. Spoiler answers by choosing an* r*-tuple* $\vec{b} \in B^r$*. Duplicator completes the round by choosing an* r*-tuple* $\vec{a} \in A^r$ *such that* $g(a_j) = h(b_j)$ *for all* $j \in [r]$*.*[2] *The players continue by playing* $\mathrm{CSPG}(\alpha[\vec{a}/\vec{y}], \beta[\vec{b}/\vec{y}])$*.*
- **Right CSP-quantifier move:** *Spoiler starts by choosing* $r \in [k]$ *and an* r*-tuple* $\vec{y} \in X^r_k$ *of distinct variables and a colouring* $h\colon B \to [n]$*. Duplicator chooses next a colouring* $g\colon A \to [n]$ *such that* $\mathrm{rng}(g) \subseteq \mathrm{rng}(h)$*. Spoiler answers by choosing an* r*-tuple* $\vec{a} \in A^r$*. Duplicator completes the round by choosing an* r*-tuple* $\vec{b} \in B^r$ *such that* $g(a_j) = h(b_j)$ *for all* $j \in [r]$*. The players continue by playing* $\mathrm{CSPG}(\alpha[\vec{a}/\vec{y}], \beta[\vec{b}/\vec{y}])$*.*

(3) *Duplicator wins the game if Spoiler does not win it in a finite number of rounds.*

We prove now that the CSP game characterizes equivalence with respect to both of the logics $\mathrm{FO}^k(\mathbf{CSP}^+_n)$ and $L^k_{\infty\omega}(\mathbf{CSP}^+_n)$.

▶ **Theorem 3.** *The following conditions are equivalent:*

(1) *Duplicator has a winning strategy in the game* $\mathrm{CSPG}[\mathfrak{A}, \mathfrak{B}, n, k](\alpha, \beta)$*,*

(2) $(\mathfrak{A}, \alpha) \equiv^k_{\infty\omega,n} (\mathfrak{B}, \beta)$*,*

(3) $(\mathfrak{A}, \alpha) \equiv^k_n (\mathfrak{B}, \beta)$*.*

Proof. (1) ⇒ (2): We prove by induction on $\varphi \in L^k_{\infty\omega}(\mathbf{CSP}^+_n)$ that (for any assignments α and β) if Duplicator has a winning strategy in $\mathrm{CSPG}(\alpha, \beta)$, then $(\mathfrak{A}, \alpha) \models \varphi \iff (\mathfrak{B}, \beta) \models \varphi$.

[2] Note that since $\mathrm{rng}(h) \subseteq \mathrm{rng}(g)$, such a tuple $\vec{a}$ always exists.

- If φ is an atomic formula, the claim follows from the fact that Spoiler always wins the game $\mathrm{CSPG}(\alpha, \beta)$ immediately if $\alpha \mapsto \beta \notin \mathrm{PI}(\mathfrak{A}, \mathfrak{B})$.
- The cases $\varphi = \neg\psi$, $\varphi = \bigvee \Psi$ and $\varphi = \bigwedge \Psi$ are straightforward.
- Assume next that $\varphi = Qy\,(\psi_1, \ldots, \psi_\ell)$ for some unary generalized quantifer Q. Let Spoiler start the game $\mathrm{CSPG}(\alpha, \beta)$ by a bijection move with variable y. By our assumption Duplicator can answer by a bijection $f\colon A \to B$ such that for any $a \in A$, she has a winning strategy in the continuation $\mathrm{CSPG}(\alpha[a/y], \beta[f(a)/y])$ of the game. By the induction hypothesis we have

  $$(\mathfrak{A}, \alpha[a/y]) \models \psi_i \iff (\mathfrak{B}, \beta[f(a)/y]) \models \psi_i.$$

  This means that f is an isomorphism between the structures $(A, \psi_1^{\mathfrak{A},\alpha,y}, \ldots, \psi_\ell^{\mathfrak{A},\alpha,y})$ and $(B, \psi_1^{\mathfrak{B},\beta,y}, \ldots, \psi_\ell^{\mathfrak{B},\beta,y})$, whence it follows that $(\mathfrak{A}, \alpha) \models \varphi \iff (\mathfrak{B}, \beta) \models \varphi$.
- Consider finally the case $\varphi = Q_{\mathfrak{C}}\vec{y}\,(\psi_1, \ldots, \psi_\ell)$ for some r-ary CSP-quantifier $Q_{\mathfrak{C}}$ with template $\mathfrak{C} = ([n], R_1^{\mathfrak{C}}, \ldots, R_\ell^{\mathfrak{C}})$. We start by assuming that $(\mathfrak{A}, \alpha) \models \varphi$. Thus, there is a homomorphism g from the structure $(A, \psi_1^{\mathfrak{A},\alpha,\vec{y}}, \ldots, \psi_\ell^{\mathfrak{A},\alpha,\vec{y}})$ to $\mathfrak{C}$. Let Spoiler play in the game $\mathrm{CSPG}(\alpha, \beta)$ a left CSP-quantifier move with r, the tuple $\vec{y} \in X_k^r$ and the function g, and let $h\colon B \to [n]$ be the answer of Duplicator given by her winning strategy. We claim that h is a homomorphism $(B, \psi_1^{\mathfrak{B},\beta,\vec{y}}, \ldots, \psi_\ell^{\mathfrak{B},\beta,\vec{y}})$ to $\mathfrak{C}$, and consequently $(\mathfrak{B}, \beta) \models \varphi$.

  Assume that this is not the case. Then there is $i \in [\ell]$ and a tuple $\vec{b} \in B^r$ such that $\vec{b} \in \psi_i^{\mathfrak{B},\beta,\vec{y}}$ (i.e., $(\mathfrak{B}, \beta[\vec{b}/\vec{y}]) \models \psi_i$), but $h(\vec{b}) \notin R_i^{\mathfrak{C}}$. Let Spoiler play the tuple $\vec{b}$ after Duplicator has played h, and let $\vec{a} \in A^r$ be the answer to this move given by the winning strategy of Duplicator. Then $g(\vec{a}) = h(\vec{b})$, and Duplicator has a winning strategy in the continuation $\mathrm{CSPG}(\alpha[\vec{a}/\vec{y}], \beta[\vec{b}/\vec{y}])$ of the game, whence by the induction hypothesis $(\mathfrak{A}, \alpha[\vec{a}/\vec{y}]) \models \psi_i$, or equivalently, $\vec{a} \in \psi_i^{\mathfrak{A},\alpha,\vec{y}}$. This is in contradiction with the fact that g is a homomorphism, since $g(\vec{a}) = h(\vec{b}) \notin R_i^{\mathfrak{C}}$. Thus, h is a homomorphism, as we claimed.

  By using the right CSP-quantifier move in place of the left CSP-quantifier move, we can prove that $(\mathfrak{B}, \beta) \models \varphi$ implies $(\mathfrak{A}, \alpha) \models \varphi$. Thus, $(\mathfrak{A}, \alpha) \models \varphi \iff (\mathfrak{B}, \beta) \models \varphi$, as desired.

The implication (2) $\Rightarrow$ (3) is trivially true, as $\mathrm{FO}^k(\mathbf{CSP}_n^+)$ is contained in $L_{\infty\omega}^k(\mathbf{CSP}_n^+)$.

(3) $\Rightarrow$ (1): Observe first that since $\mathfrak{A}$ is finite, there are only finitely many formulas of $\mathrm{FO}^k(\mathbf{CSP}_n^+)$ that are non-equivalent on $\mathfrak{A}$. Thus, for each assignment γ of $\mathfrak{A}$ with $\mathrm{dom}(\gamma) \subseteq X_k$ there is a formula $\Psi_{\mathfrak{A}}^{n,k}(\gamma) \in \mathrm{FO}^k(\mathbf{CSP}_n^+)$ such that $(\mathfrak{A}, \gamma) \models \Psi_{\mathfrak{A}}^{n,k}(\gamma)$, and

$(*)$ $(\mathfrak{C}, \delta) \models \Psi_{\mathfrak{A}}^{n,k}(\gamma)$ implies $(\mathfrak{A}, \gamma) \equiv_n^k (\mathfrak{C}, \delta)$ for any structure $\mathfrak{C}$ and assignment δ.

Assume now that $(\mathfrak{A}, \alpha) \equiv_n^k (\mathfrak{B}, \beta)$. We show that Duplicator can play in the first round of the game $\mathrm{CSPG}(\alpha, \beta)$ in such a way that $(\mathfrak{A}, \alpha') \equiv_n^k (\mathfrak{B}, \beta')$ holds in the next position (α', β') of the game. Clearly playing this way in all rounds of the game, Duplicator is guaranteed to win, as the condition $(\mathfrak{A}, \alpha) \equiv_n^k (\mathfrak{B}, \beta)$ implies that $\alpha \mapsto \beta \in \mathrm{PI}(\mathfrak{A}, \mathfrak{B})$. There are three cases based on the type of the first move Spoiler chooses:

- Spoiler makes a bijection move, and picks a variable $y \in X_k$. For the sake of simplicity we use below the shorthand notation $\Psi_a := \Psi_{\mathfrak{A}}^{n,k}(\alpha[a/y])$ for each $a \in A$. The assumption $(\mathfrak{A}, \alpha) \equiv_n^k (\mathfrak{B}, \beta)$ implies that $(\mathfrak{B}, \beta) \models \exists^{=|A|} y\,(y = y)$, and for each $a \in A$, $(\mathfrak{B}, \beta) \models \exists^{=m_a} y\,\Psi_a$, where $m_a = |\{a' \in A \mid \Psi_{a'} \Leftrightarrow \Psi_a\}|$. This means that $|B| = |A|$ and $|\{b \in B \mid (\mathfrak{B}, \beta[b/y]) \models \Psi_a\}| = m_a$ for all $a \in A$, whence there is a bijection $f\colon A \to B$ such that for every $a \in A$, $(\mathfrak{B}, \beta[f(a)]) \models \Psi_a$. Thus, using this bijection f as her answer to the move of Spoiler, Duplicator makes sure that $(\mathfrak{B}, \beta[f(a)/y]) \models \Psi_a$, and hence by $(*)$, $(\mathfrak{A}, \alpha[a/y]) \equiv_n^k (\mathfrak{B}, \beta[f(a)/y])$ holds in the next position $(\alpha[a/y], \beta[f(a)/y])$ of the game.

- Spoiler makes a left CSP-quantifier move, and chooses $r \in [k]$, an r-tuple $\vec{y} \in X_k^r$ of variables and a colouring $g\colon A \to [n]$. To simplify notation, we denote $\Psi_{\mathfrak{A}}^{n,k}(\alpha[\vec{a}/\vec{y}])$ by $\Psi_{\vec{a}}$ for each $\vec{a} \in A^r$. Let $\mathfrak{C} = ([n], (R_{\vec{a}}^{\mathfrak{C}})_{\vec{a}\in A^r}, R_{\emptyset}^{\mathfrak{C}})$ be the canonical structure arising from the function g and the formulas $\Psi_{\vec{a}}$: $R_{\vec{a}}^{\mathfrak{C}} = \{g(\vec{a}') \mid \Psi_{\vec{a}'} \Leftrightarrow \Psi_{\vec{a}}\}$ and $R_{\emptyset}^{\mathfrak{C}} = \emptyset$. Then by the assumption $(\mathfrak{A}, \alpha) \equiv_n^k (\mathfrak{B}, \beta)$, $(\mathfrak{B}, \beta) \models Q_{\mathfrak{C}}\vec{y}((\Psi_{\vec{a}})_{\vec{a}\in A^r}, \bigwedge_{\vec{a}\in A^r} \neg\Psi_{\vec{a}})$. Thus, there is a homomorphism h from the structure $(B, (\Psi_{\vec{a}}^{\mathfrak{B},\beta,\vec{y}})_{\vec{a}\in A^r}, (\bigwedge_{\vec{a}\in A^r} \neg\Psi_{\vec{a}})^{\mathfrak{B},\beta,\vec{y}})$ to $\mathfrak{C}$. Let Duplicator use this function $h\colon B \to [n]$ as her move, and assume that Spoiler chooses next the tuple $\vec{b} \in B^r$. Since $R_{\emptyset}^{\mathfrak{C}} = \emptyset$, there exists $\vec{a} \in A^r$ such that $\vec{b} \in \Psi_{\vec{a}}^{\mathfrak{B},\beta,\vec{y}}$, i.e., $(\mathfrak{B}, \beta[\vec{b}/\vec{y}]) \models \Psi_{\vec{a}}$. Then $h(\vec{b}) \in R_{\vec{a}}^{\mathfrak{C}}$, whence by the definition of $R_{\vec{a}}^{\mathfrak{C}}$, there exists $\vec{a}' \in A^r$ such that $h(\vec{b}) = g(\vec{a}')$ and $\Psi_{\vec{a}'} \Leftrightarrow \Psi_{\vec{a}}$. We let Duplicator use this tuple $\vec{a}'$ as the final step of her move. The next position in the game is then $(\alpha[\vec{a}'/\vec{y}], \beta[\vec{b}/\vec{y}])$, and $(\mathfrak{B}, \beta[\vec{b}/\vec{y}]) \models \Psi_{\vec{a}'}$, whence $(*)$ implies that $(\mathfrak{A}, \alpha[\vec{a}'/\vec{y}]) \equiv_n^k (\mathfrak{B}, \beta[\vec{b}/\vec{y}])$, as desired.
- The case of right CSP-quantifier move is proved in the same way by switching the roles of the structure $(\mathfrak{A}, \alpha)$ and $(\mathfrak{B}, \beta)$. ◀

4.2 Bijective colouring game

As we mentioned in the beginning of Section 4, instead of the CSP game we will use another game with simpler rules in the proof of the size hierarchy theorem in Section 7. We will now introduce this game.

Assume again that $\mathfrak{A}$ and $\mathfrak{B}$ are τ-structures, and α and β are assignments on them with $\mathrm{dom}(\alpha) = \mathrm{dom}(\beta) \subseteq X_k$. Furthermore, let $g\colon A \to [n]$ and $h\colon B \to [n]$ be colourings. We define next the *bijective colouring game* $\mathrm{BCG}(\alpha, \beta, g, h) := \mathrm{BCG}[\mathfrak{A}, \mathfrak{B}, n, k](\alpha, \beta, g, h)$ for $(\mathfrak{A}, g, \alpha)$ and $(\mathfrak{B}, h, \beta)$ with parameters n and k.

▶ **Definition 4.** *The rules of the game* $\mathrm{BCG}[\mathfrak{A}, \mathfrak{B}, n, k](\alpha, \beta, g, h)$ *are the following:*

(1) *If* $\alpha \mapsto \beta \notin \mathrm{PI}(\mathfrak{A}^g, \mathfrak{B}^h)$*, then the game ends, and Spoiler wins.*

(2) *Otherwise Duplicator chooses a bijection* $f : A \to B$ *(if* $|A| \neq |B|$ *the game ends and Spoiler wins). Spoiler can now choose to play one of the two options:*

- **Element move:** *Spoiler chooses a variable* $y \in X_k$ *and an element* $a \in A$*. The players continue by playing* $\mathrm{BCG}(\alpha[a/y], \beta[f(a)/y], g, h)$*.*
- **Colouring move:** *Spoiler chooses a function* $g' : A \to [n]$*. The players continue by playing* $\mathrm{BCG}(\alpha, \beta, g', h')$*, where* h' *is the unique function* $B \to [n]$ *such that* $g' = h' \circ f$*.*

(3) *Duplicator wins the game if Spoiler does not win it in a finite number of rounds.*

The following result shows that the bijective colouring game corresponds to an at least as strong equivalence as the CSP game. We leave it as an open problem, whether the converse of this holds.

▶ **Theorem 5.** *If Duplicator has a winning strategy in the game* $\mathrm{BCG}[\mathfrak{A}, \mathfrak{B}, n, k](\alpha, \beta, g, h)$ *for some colourings* g *and* h*, then she has a winning strategy in the game* $\mathrm{CSPG}[\mathfrak{A}, \mathfrak{B}, n, k](\alpha, \beta)$*.*

Proof. Assume that $g\colon A \to [n]$ and $h\colon B \to [n]$ are colourings such that Duplicator has a winning strategy in $\mathrm{BCG}(\alpha, \beta, g, h)$. Then $\alpha \mapsto \beta \in \mathrm{PI}(\mathfrak{A}^g, \mathfrak{B}^h) \subseteq \mathrm{PI}(\mathfrak{A}, \mathfrak{B})$, and hence Spoiler does not win $\mathrm{CSPG}(\alpha, \beta)$ immediately in position (α, β). We will show that Duplicator can play in the game $\mathrm{CSPG}(\alpha, \beta)$ in such a way that the condition

(†) Duplicator has a winning strategy in $\mathrm{BCG}(\gamma, \delta, \tilde{g}, \tilde{h})$ for some $\tilde{g}\colon A \to [n]$, $\tilde{h}\colon B \to [n]$

holds in every position (γ, δ) during the play. Since (†) implies that $\gamma \mapsto \delta \in \mathrm{PI}(\mathfrak{A}, \mathfrak{B})$, playing this way Duplicator is guaranteed to win the game $\mathrm{CSPG}(\alpha, \beta)$.

Assume now that Spoiler and Duplicator have reached in the game $\mathrm{CSPG}(\alpha, \beta)$ a position (γ, δ) such that (†) holds. Let f be the bijection given by the winning strategy of Duplicator in $\mathrm{BCG}(\gamma, \delta, \tilde{g}, \tilde{h})$. We consider now the options Spoiler has for his move in $\mathrm{CSPG}(\gamma, \delta)$.

- Spoiler plays a bijection move, and picks a variable $y \in X_k$. We let Duplicator use the bijection f as her answer. Let Spoiler choose $a \in A$ to complete the round of the game. By the choice of f, Duplicator has then a winning strategy in the game $\mathrm{BCG}(\gamma[a/y], \delta[f(a)/y], \tilde{g}, \tilde{h})$. Thus (†) holds in the next position $(\gamma[a/y], \delta[f(a)/y])$ of the game $\mathrm{CSPG}(\alpha, \beta)$.
- Spoiler plays a right CSP-quantifier move, and picks a tuple $\vec{y} = (y_1, \ldots, y_r) \in X_k^r$ and a colouring $h' \colon B \to [n]$. Duplicator answers this by choosing $g' = h' \circ f$. Note that if Spoiler chooses g' in the game $\mathrm{BCG}(\gamma, \delta, \tilde{g}, \tilde{h})$, then the next position is (γ, δ, g', h'). Thus Duplicator has a winning strategy in $\mathrm{BCG}(\gamma, \delta, g', h')$. Let Spoiler choose next a tuple $\vec{a} = (a_1 \ldots, a_r) \in A^r$. We define the components b_i of the answer $\vec{b} \in B^r$ of Duplicator by induction on $i \in [r]$:
  - Assume that $b_1, \ldots, b_i$, $i < r$, are already defined, and Duplicator has a winning strategy in $\mathrm{BCG}(\gamma[\vec{a}_i/\vec{y}_i], \delta[\vec{b}_i/\vec{y}_i], g', h')$, where $\vec{a}_i := (a_1, \ldots, a_i)$, $\vec{b}_i := (b_1, \ldots, b_i)$ and $\vec{y}_i := (y_1, \ldots, y_i)$. Let f_{i+1} be the bijection given by Duplicators winning strategy in this game. Let Spoiler now play an element move and choose the component a_{i+1} as his move. Then we let $b_{i+1} = f_{i+1}(a_{i+1})$. By the choice of f_{i+1}, Duplicator has a winning strategy in the continuation $\mathrm{BCG}(\gamma[\vec{a}_i a_{i+1}/\vec{y}_i y_{i+1}], [\vec{b}_i b_{i+1}/\vec{y}_i y_{i+1}], g', h')$ of the game.

  Thus, choosing the tuple $\vec{b}$ defined above as her answer to $\vec{a}$, Duplicator guarantees that condition (†) holds in the next position $(\gamma[\vec{a}/\vec{y}], \delta[\vec{b}/\vec{y}])$.
- The case of left CSP-quantifier move is proved in the same way. ◀

▶ **Corollary 6.** *If Duplicator has a winning strategy in the game* $\mathrm{BCG}[\mathfrak{A}, \mathfrak{B}, n, k](\alpha, \beta, g, h)$ *for some colourings* g *and* h*, then* $(\mathfrak{A}, \alpha) \equiv^k_{\infty\omega,n} (\mathfrak{B}, \beta)$.

Thus, the bijective colouring game can be used for proving undefinability results for $L^\omega_{\infty\omega}(\mathbf{CSP}^+_n)$ in the usual way: a generalized quantifier $Q_\mathcal{K}$ is not definable in $L^\omega_{\infty\omega}(\mathbf{CSP}^+_n)$ if for all k there are structures $\mathfrak{A}_k \in \mathcal{K}$ and $\mathfrak{B}_k \notin \mathcal{K}$ such that Duplicator has a winning strategy in $\mathrm{BCG}[\mathfrak{A}_k, \mathfrak{B}_k, n, k](\emptyset, \emptyset, g, h)$ for some g and h.

5 Generalized CFI structures

Fix a natural number n, a $3n$-ary relation symbol R_n, and a connected 3-regular ordered graph $G = (V, E, <_G)$. We describe now the details of the construction of generalized CFI structures $\mathfrak{A}^{\mathrm{ev}}_n(G)$ and $\mathfrak{A}^{\mathrm{od}}_n(G)$. We start by defining structures $\mathfrak{A}^v_n = (A^v_n, R^v_n)$ and $\tilde{\mathfrak{A}}^v_n = (A^v_n, \tilde{R}^v_n)$ for $v \in V$ that will be used as building blocks of $\mathfrak{A}^{\mathrm{ev}}_n(G)$ and $\mathfrak{A}^{\mathrm{od}}_n(G)$. However, before this we introduce some useful notation.

We denote the set of all permutations of a set T by $\mathsf{S}(T)$ and the set of even permutations of T by $\mathsf{A}(T)$. If T is an indexed set of the form $\{t_i \mid i \in [n+1]\}$ and $\pi \in \mathsf{S}(T)$, then we use the notation $\vec{\pi}$ for the n-tuple $(\pi(t_1), \ldots, \pi(t_n))$ (note that $\pi(t_{n+1})$ is not included in $\vec{\pi}$, as π is completely determined by the values $\pi(t_i)$ for $i \in [n]$). The *parity* $\mathsf{p}(\pi) \in \{0, 1\}$ of a permutation $\pi \in \mathsf{S}(T)$ is defined as

$$\mathsf{p}(\pi) := \begin{cases} 0, & \text{if } \pi \in \mathsf{A}(T) \\ 1, & \text{if } \pi \notin \mathsf{A}(T). \end{cases}$$

For $a, b \in \{0, 1\}$, we denote their sum modulo 2 by $a \oplus b$.

For $v \in V$, we denote by $\vec{e}(v)$ the tuple (r, s, t) of edges adjacent to a vertex $v \in V$, where the components are listed in the order $<_G$. Furthermore, we denote by $E(v)$ the set $\{r, s, t\}$. For the definition of $\mathfrak{A}_n^v$ and $\tilde{\mathfrak{A}}_n^v$ we also fix distinct elements a_i^e for all $e \in E$ and $i \in [n+1]$.

▶ **Definition 7.** *For each $e \in E$, we define $A_n^e := \{a_i^e \mid i \in [n+1]\}$.*
- *For each $v \in V$, we define $A_n^v := A_n^r \cup A_n^s \cup A_n^t$, where $\vec{e}(v) = (r, s, t)$.*
- *Let $v \in V$ and $\vec{e}(v) = (r, s, t)$. We define the $\{R_n\}$-structures $\mathfrak{A}_n^v := (A_n^v, R_n^v)$ and $\tilde{\mathfrak{A}}_n^v := (A_n^v, \tilde{R}_n^v)$ by setting*
  - $R_n^v := \{\vec{\pi}\vec{\rho}\vec{\sigma} \in P_n^v \mid \mathsf{p}(\pi) \oplus \mathsf{p}(\rho) \oplus \mathsf{p}(\sigma) = 0\}$
  - $\tilde{R}_n^v := \{\vec{\pi}\vec{\rho}\vec{\sigma} \in P_n^v \mid \mathsf{p}(\pi) \oplus \mathsf{p}(\rho) \oplus \mathsf{p}(\sigma) = 1\}$,

  where we use the notation
  - $P_n^v := \{\vec{\pi}\vec{\rho}\vec{\sigma} \mid \pi \in \mathsf{S}(A_n^r), \rho \in \mathsf{S}(A_n^s), \sigma \in \mathsf{S}(A_n^t)\}$

We can now state the important characterization of the automorphisms of these structures and isomomorphisms between them. We say that a bijection $f\colon A_n^v \to A_n^v$ *preserves edges* if $f(a_i^e) \in A_n^e$ for all $e \in E(v)$ and $i \in [n]$. Note that if this holds, then $f = \bigcup_{e \in E(v)} f_e$, where $f_e := f \restriction A_n^e$ for each $e \in E(v)$.

▶ **Lemma 8.** *Let $f\colon A_n^v \to A_n^v$ be a bijection, and let $\vec{e}(v) = (r, s, t)$.*

(a) *f is an automorphism of $\mathfrak{A}_n^v$ and $\tilde{\mathfrak{A}}_n^v$ if and only if it* preserves edges *and $\mathsf{p}(f) := \mathsf{p}(f_r) \oplus \mathsf{p}(f_s) \oplus \mathsf{p}(f_t) = 0$;*

(b) *f is an isomorphism between $\mathfrak{A}_n^v$ and $\tilde{\mathfrak{A}}_n^v$ if and only if it preserves edges and $\mathsf{p}(f) = 1$.*

Proof. If f does not preserve edges, then at least for two of the edges $e \in E(v)$ there are permutations $\pi \in \mathsf{S}(A_n^e)$ such that $f(\vec{\pi}) \notin \{\vec{\rho} \mid \rho \in \mathsf{S}(A_n^e)\}$. Clearly this means that $\vec{\pi}$ can be extended to a tuple $\vec{a} \in R_n^v$ such that $f(\vec{a}) \notin R_n^v$ and $f(\vec{a}) \notin \tilde{R}_n^v$. Thus, f is neither an automorphism of $\mathfrak{A}_n^v$, nor an isomorphism from $\mathfrak{A}_n^v$ to $\tilde{\mathfrak{A}}_n^v$. Similarly, there is a tuple $\vec{b} \in \tilde{R}_n^v$ extending $\vec{\pi}$ such that $f(\vec{b}) \notin R_n^v$ and $f(\vec{b}) \notin \tilde{R}_n^v$, whence f is neither an automorphism of $\tilde{\mathfrak{A}}_n^v$, nor an isomorphism from $\tilde{\mathfrak{A}}_n^v$ to $\mathfrak{A}_n^v$.

Assume then that f is edge preserving. Clearly in order to determine whether f is an automorphism of $\mathfrak{A}_n^v$ and $\tilde{\mathfrak{A}}_n^v$ (or an isomorphism between them), it suffices to consider tuples of the form $\vec{a} = \vec{\pi}\vec{\rho}\vec{\sigma} \in P_n^v$. Observe that $f(\vec{a}) = f_r(\vec{\pi})f_s(\vec{\rho})f_t(\vec{\sigma}) = \overrightarrow{f_r \circ \pi}\,\overrightarrow{f_s \circ \rho}\,\overrightarrow{f_t \circ \sigma}$ and

$$\mathsf{p}(f_r \circ \pi) \oplus \mathsf{p}(f_s \circ \rho) \oplus \mathsf{p}(f_t \circ \sigma) = \mathsf{p}(\pi) \oplus \mathsf{p}(\rho) \oplus \mathsf{p}(\sigma) \oplus \mathsf{p}(f).$$

Note further that P_n^v is the disjoint union of R_n^v and $\tilde{R}_n^v$. Thus, if $\mathsf{p}(f) = 0$, we have

$$\vec{a} \in R_n^v \iff \vec{a} \notin \tilde{R}_n^v \iff f(\vec{a}) \notin \tilde{R}_n^v \iff f(\vec{a}) \in R_n^v,$$

and if $\mathsf{p}(f) = 1$, we have

$$\vec{a} \in R_n^v \iff \vec{a} \notin \tilde{R}_n^v \iff f(\vec{a}) \in \tilde{R}_n^v \iff f(\vec{a}) \notin R_n^v.$$

This completes the proof of both (a) and (b). ◀

We assign next an $\{R_n\}$-structure $\mathfrak{A}_n(G, U)$ to each subset U of V. The CFI structures $\mathfrak{A}_n^{\mathrm{ev}}(G)$ and $\mathfrak{A}_n^{\mathrm{od}}(G)$ are later defined as special cases of $\mathfrak{A}_n(G, U)$.

▶ **Definition 9.** *Let $U \subseteq V$. We define the $\{R_n\}$-structure $\mathfrak{A}_n(G, U) := (A_n(G), R_n(G, U))$ by setting*
- *$A_n(G) := \bigcup_{v \in V} A_n^v = \{a_i^e \mid i \in [n+1], e \in E\}$*
- *$R_n(G, U) := \bigcup_{v \in V} R_n^v(U)$,*

where $R_n^v(U) = \tilde{R}_n^v$ if $v \in U$ and $R_n^v(U) = R_n^v$ if $v \notin U$.

▶ **Lemma 10.** *If $U, U' \subseteq V$ are such that $|U| \equiv |U'| \pmod 2$, then $\mathfrak{A}_n(G,U)$ and $\mathfrak{A}_n(G,U')$ are isomorphic.*

Proof. It suffices to prove the following claim:

(∗) If $U, U' \subseteq V$ and $|(U \setminus U') \cup (U' \setminus U)| = 2$, then $\mathfrak{A}_n(G,U) \cong \mathfrak{A}_n(G,U')$.

The lemma follows from this, since by repeated use of (∗) we see that $\mathfrak{A}_n(G,U) \cong \mathfrak{A}_n(G,\emptyset)$ whenever $|U|$ is even, and $\mathfrak{A}_n(G,U) \cong \mathfrak{A}_n(G,\{v\})$ for any $v \in V$ whenever $|U|$ is odd.

To prove (∗), assume that $U, U' \subseteq V$ and $(U \setminus U') \cup (U' \setminus U) = \{u, u'\}$. Since G is connected, there is an E-path P consisting of edges $e_i = \{u_{i-1}, u_i\}$, $i \in [\ell]$, such that $u_0 = u$ and $u_\ell = u'$. For each $e \in E$, we define a function $f_e \colon A_n^e \to A_n^e$ as follows:

- If $e = e_i$ for some $i \in [\ell]$, then $f_e(a_1^e) = a_2^e$, $f_e(a_2^e) = a_1^e$ and $f_e(a_j^e) = a_j^e$ for $j > 2$.
- Otherwise $f_e = \mathrm{id}_{A_n^e}$ (the identity function of A_n^e).

Let $f : A_n(G) \to A_n(G)$ be the function determined by the functions f_e, $e \in E$: $f(a_i^e) = f_e(a_i^e)$ for all $i \in [n+1]$ and $e \in E$. We show next that f is an isomorphism from $\mathfrak{A}_n(G,U)$ to $\mathfrak{A}_n(G,U')$. Clearly it suffices to show that, for each $v \in V$, the restriction f_v of f to the set A_n^v is an isomorphism $\mathfrak{A}_n^v(U) \to \mathfrak{A}_n^v(U')$, where $\mathfrak{A}_n^v(U) := (A_n^v, R_n^v(U))$.

Note first that $f_v = f_r \cup f_s \cup f_t$, where $(r,s,t) = \vec{e}(v)$, whence f_v is an edge preserving bijection. We consider now the following three cases according to the number $n(P,v) := |\{i \in [\ell] \mid e_i \in E(v)\}|$ (note that the case $n(P,v) = 3$ is not possible):

(1) If $n(P,v) = 0$, then clearly $\mathsf{p}(f_v) = 0$ and $\mathfrak{A}_n^v(U) = \mathfrak{A}_n^v(U') \in \{\mathfrak{A}_n^v, \tilde{\mathfrak{A}}_n^v\}$. Thus, f_v is an isomorphism $\mathfrak{A}_n^v(U) \to \mathfrak{A}_n^v(U')$ by Lemma 8(a).

(2) If $n(P,v) = 1$, then clearly $\mathsf{p}(f_v) = 1$ and $v \in \{u, u'\}$, and hence either $\mathfrak{A}_n^v(U) = \mathfrak{A}_n^v$ and $\mathfrak{A}_n^v(U') = \tilde{\mathfrak{A}}_n^v$, or $\mathfrak{A}_n^v(U) = \tilde{\mathfrak{A}}_n^v$ and $\mathfrak{A}_n^v(U') = \mathfrak{A}_n^v$. Thus, the claim follows from Lemma 8(b) .

(3) If $n(P,v) = 2$, then $\mathsf{p}(f_v) = 0$ and $v \notin \{u, u'\}$, whence $\mathfrak{A}_n^v(U) = \mathfrak{A}_n^v(U') \in \{\mathfrak{A}_n^v, \tilde{\mathfrak{A}}_n^v\}$. The claim follows again from Lemma 8(a). ◀

By Lemma 10, there are at most two isomorphism types of the structures $\mathfrak{A}_n(G,U)$. We use $\mathfrak{A}_n^{\mathrm{ev}}(G) := \mathfrak{A}_n(G,\emptyset)$ and $\mathfrak{A}_n^{\mathrm{od}}(G) := \mathfrak{A}_n(G,\{v_0\})$ as representatives of these classes, where $v_0 \in V$ is the smallest vertex with respect to the order $<_G$. To simplify notation, we denote $R_n(G,\emptyset)$ and $R_n(G,\{v_0\})$ simply by R_n^{ev} and R_n^{od}, respectively.

6 Separating the CFI structures

We prove next that $\mathfrak{A}_n^{\mathrm{ev}}(G)$ and $\mathfrak{A}_n^{\mathrm{od}}(G)$ are indeed non-isomorphic. More precisely, we show that $\mathfrak{A}_n^{\mathrm{ev}}(G)$ and $\mathfrak{A}_n^{\mathrm{od}}(G)$ are separated by a constraint satisfaction problem $\mathsf{CSP}(\mathfrak{C}_n)$. The template structure $\mathfrak{C}_n$ is defined as follows:

▶ **Definition 11.** *We define $\mathfrak{C}_n := (C_n, \mathcal{R}_n)$ to be the $\{R_n\}$-structure such that*

- *$C_n := [n+1]$, and*
- *$\mathcal{R}_n = \{\vec{\pi}\vec{\rho}\vec{\sigma} \mid \pi, \rho, \sigma \in \mathsf{S}[n+1],\ \mathsf{p}(\pi) \oplus \mathsf{p}(\rho) \oplus \mathsf{p}(\sigma) = 0\}$.*

Here we consider C_n as the indexed set $\{c_i \mid i \in [n+1]\}$, where $c_i = i$ for each $i \in [n+1]$.

▶ **Proposition 12.** *$\mathfrak{A}_n^{\mathrm{ev}}(G) \in \mathsf{CSP}(\mathfrak{C}_n)$, but $\mathfrak{A}_n^{\mathrm{od}}(G) \notin \mathsf{CSP}(\mathfrak{C}_n)$.*

Proof. Given a permutation $\pi \in \mathsf{S}(A_n^e)$, $e \in E$, we denote its natural projection to $[n+1]$ by π_*: i.e., $\pi_*(i) = j$ if and only if $\pi(a_i^e) = a_j^e$. Note that clearly $\mathsf{p}(\pi_*) = \mathsf{p}(\pi)$.

We show that the function $g\colon A_n(G) \to C_n$ such that $g(a_i^e) = i$ for all $e \in E$ and $i \in [n+1]$, is a homomorphism $\mathfrak{A}_n^{\mathrm{ev}}(G) \to \mathfrak{C}_n$. Thus, assume that $\vec{a} \in R_n^{\mathrm{ev}}$. Then $\vec{a} \in R_n^v$ for some $v \in V$, whence there are permutations $\pi \in \mathsf{S}(A_n^r)$, $\rho \in \mathsf{S}(A_n^s)$ and $\sigma \in \mathsf{S}(A_n^t)$, such that $\mathsf{p}(\pi) \oplus \mathsf{p}(\rho) \oplus \mathsf{p}(\sigma) = 0$ and $\vec{a} = \vec{\pi}\vec{\rho}\vec{\sigma}$, where $\vec{e}(v) = (r,s,t)$. Then we have $g(\vec{a}) = g(\vec{\pi})g(\vec{\rho})g(\vec{\sigma}) = \vec{\pi}_*\vec{\rho}_*\vec{\sigma}_* \in \mathcal{R}_n$, since $\mathsf{p}(\pi_*) \oplus \mathsf{p}(\rho_*) \oplus \mathsf{p}(\sigma_*) = \mathsf{p}(\pi) \oplus \mathsf{p}(\rho) \oplus \mathsf{p}(\sigma) = 0$.

To prove that $\mathfrak{A}_n^{\mathrm{od}}(G) \notin \mathsf{CSP}(\mathfrak{C}_n)$, assume towards a contradiction that $h\colon A_n(G) \to C_n$ is a homomorphism $\mathfrak{A}_n^{\mathrm{od}}(G) \to \mathfrak{C}_n$. For each $e \in E$, we let $\chi_e\colon [n+1] \to [n+1]$ be the function such that $\chi_e(i) := h(a_i^e)$.

We observe first that χ_e is a bijection for all $e \in E$. Indeed, if χ_e is not a bijection, then there are $i, j \in [n+1]$, such that $i \neq j$ and $\chi_e(i) = \chi_e(j)$. Let $\pi \in \mathsf{S}(A_n^e)$ be a permutation such that a_i^e and a_j^e occur in $\vec{\pi}$. Then there are edges e' and e'' and permutations $\rho \in \mathsf{S}(A_n^{e'})$ and $\sigma \in \mathsf{S}(A_n^{e''})$ such that $\vec{a} \in R_n^{\mathrm{od}}$, where $\vec{a}$ is either $\vec{\pi}\vec{\rho}\vec{\sigma}$, $\vec{\rho}\vec{\pi}\vec{\sigma}$, or $\vec{\rho}\vec{\sigma}\vec{\pi}$ (depending on the order between e, e' and e''). On the other hand, $h(\vec{a}) \notin \mathcal{R}_n$, since $h(\vec{\pi})$ is not of the form $\vec{\eta}$ for any permutation $\eta \in \mathsf{S}[n+1]$. Hence h is not a homomorphism against our assumption.

Observe next that if $\pi \in \mathsf{S}(A_n^e)$, then $h(\pi(a_i^e)) = \chi_e(\pi_*(i))$ for all $i \in [n+1]$, whence $h(\vec{\pi}) = \overrightarrow{\chi_e \circ \pi_*}$. In particular, $h(\vec{\iota}) = \vec{\chi}_e$ for the identity permutation ι of A_n^e. Consider now a vertex $v \in V \setminus \{v_0\}$ with $\vec{e}(v) = (e_1, e_2, e_3)$ and the tuple $\vec{a} = \vec{\iota}_1\vec{\iota}_2\vec{\iota}_3$, where $\iota_j \in \mathsf{S}(A_n^{e_j})$ is the identity permutation of $A_n^{e_j}$ for $j \in [3]$. Then $\vec{a} \in R_n^{\mathrm{od}}$, whence we must have $h(\vec{a}) = \vec{\chi}_{e_1}\vec{\chi}_{e_2}\vec{\chi}_{e_3} \in \mathcal{R}_n$, or equivalently, $\mathsf{p}(\chi_{e_1}) \oplus \mathsf{p}(\chi_{e_2}) \oplus \mathsf{p}(\chi_{e_3}) = 0$.

On the other hand, if $\vec{e}(v_0) = (d_1, d_2, d_3)$, $\iota_j \in \mathsf{S}(A_n^{d_j})$ is the identity permutation of $A_n^{d_j}$ for $j \in [2]$, and $\pi \in \mathsf{S}(A_n^{d_3}) \setminus \mathsf{A}(A_n^{d_3})$, then $\vec{a} = \vec{\iota}_1\vec{\iota}_2\vec{\pi} \in R_n^{\mathrm{od}}$. Hence we must have $h(\vec{a}) = \vec{\chi}_{d_1}\vec{\chi}_{d_2}\overrightarrow{\chi_{d_3} \circ \pi_*} \in \mathcal{R}_n$, or equivalently, $\mathsf{p}(\chi_{d_1}) \oplus \mathsf{p}(\chi_{d_2}) \oplus \mathsf{p}(\chi_{d_3}) \oplus \mathsf{p}(\pi_*) = 0$. Since $\mathsf{p}(\pi_*) = 1$, it follows that $\mathsf{p}(\chi_{d_1}) \oplus \mathsf{p}(\chi_{d_2}) \oplus \mathsf{p}(\chi_{d_3}) = 1$.

For each $v \in V$, let $O(h,v)$ be the number $|\{e \in E(v) \mid \mathsf{p}(\chi_e) = 1\}|$. By the observations above, we see that $O(h,v)$ is even for all $v \in V \setminus \{v_0\}$ and $O(h,v_0)$ is odd. Thus we see that the sum $O(h) := \sum_{v \in V} O(h,v)$ of these numbers is odd. However this is impossible, since clearly $O(h) = 2 \cdot |\{e \in E \mid \mathsf{p}(\chi_e) = 1\}|$, as each $e \in E$ is adjacent to exactly two vertices. ◀

▶ **Proposition 13.** $\mathsf{CSP}(\mathfrak{C}_n)$ *is* NP*-complete.*

Proof. We give a reduction from 3-colourability to $\mathsf{CSP}(\mathfrak{C}_2)$; it is easy to generalize this reduction to the case of $\mathsf{CSP}(\mathfrak{C}_n)$ with $n > 2$. Given a graph $G = (V,E)$, we define an $\{R_2\}$-structure $\mathfrak{D} = (D, R_2^{\mathfrak{D}})$ as follows:

- $D := V \cup \{(u,v,i) \mid \{u,v\} \in E,\ i \in [2]\}$,
- $R_2^{\mathfrak{D}} := \{(u,v,u,v,(u,v,1),(u,v,2)) \mid \{u,v\} \in E\}$.

We show that G is 3-colourable if and only if $\mathfrak{D} \in \mathsf{CSP}(\mathfrak{C}_2)$.

Assume first that $h\colon V \to [3]$ is a 3-colouring of G. Let g be the extension of h to the set D obtained by setting $g(u,v,i) = i$ for all $\{u,v\} \in E$ and $i \in [2]$. If $\vec{a} = (u,v,u,v,(u,v,1),(u,v,2)) \in R_2^{\mathfrak{D}}$, then $g(\vec{a}) = (h(u),h(v),h(u),h(v),1,2) = \vec{\pi}\,\vec{\pi}\,\vec{\iota}$, where $\pi \in \mathsf{S}[3]$ is the permutation such that $\pi(1) = h(u)$ and $\pi(2) = h(v)$, and $\iota \in \mathsf{S}[3]$ is the identity permutation. Thus we see $g(\vec{a}) \in \mathcal{R}_2$, as clearly $\mathsf{p}(\pi) \oplus \mathsf{p}(\pi) \oplus \mathsf{p}(\iota) = 0$. Hence g is a homomorphism $\mathfrak{D} \to \mathfrak{C}_2$.

On the other hand, if $g\colon D \to [3]$ is a homomorphism $\mathfrak{D} \to \mathfrak{C}_2$, then necessarily $g(u) \neq g(v)$ whenever $\{u,v\} \in E$. Thus, the restriction of g to V is a 3-colouring of G. ◀

We do not know whether the structures $\mathfrak{A}_n^{\mathrm{ev}}(G)$ and $\mathfrak{A}_n^{\mathrm{od}}(G)$ can be separated by a PTIME-computable CSP-quantifier, but, in any case, there is a PTIME-property that separates them.

▶ **Proposition 14.** *There exists a* PTIME*-computable class $\mathcal{K}$ of $\{R_n\}$-structures that is closed under isomorphisms such that $\mathfrak{A}_n^{\mathrm{ev}}(G) \in \mathcal{K}$ and $\mathfrak{A}_n^{\mathrm{od}}(G) \notin \mathcal{K}$.*

Proof. Let $\mathcal{K}$ consist of all structures that are isomorphic to $\mathfrak{A}_n^{\mathrm{ev}}(G)$ for some ordered connected 3-regular graph $G = (V, E, <_G)$. Then $\mathfrak{A}_n^{\mathrm{ev}}(G) \in \mathcal{K}$ by definition, and $\mathfrak{A}_n^{\mathrm{od}}(G) \notin \mathcal{K}$ by Lemma 10 and Proposition 12.

We need to show that the membership problem of $\mathcal{K}$ is in PTIME. Let $\mathfrak{B} = (B, R_n^{\mathfrak{B}})$ be an $\{R_n\}$-structure. We describe now an algorithm for deciding whether $\mathfrak{B} \in \mathcal{K}$.

(1) We define first a binary relation P on B by the condition

$$(b, b') \in P \iff \text{there are tuples } \vec{b}_1, \vec{b}_2, \vec{b}_3 \in B^n \text{ such that } \vec{b}_1\vec{b}_2\vec{b}_3 \in R_n^{\mathfrak{B}},$$
$$b \text{ occurs in } \vec{b}_i \text{ and } b' \text{ occurs in } \vec{b}_j \text{ for some } 1 \le i \le j \le 3.$$

If $\mathfrak{B}$ is in the class $\mathcal{K}$, then the transitive closure $\preceq$ of P is a linear pre-order of B (i.e., it is transitive and satisfies the dichotomy law: $b \preceq b'$ or $b' \preceq b$ for all $b, b' \in B$). Thus, we compute the transitive closure $\preceq$ of P and reject $\mathfrak{B}$ if it is not a linear pre-order.

(2) Next we define the equivalence relation $\sim$ that corresponds to $\preceq$:

$$b \sim b' \iff b \preceq b' \text{ and } b' \preceq b.$$

We check that every $\sim$-equivalence class $[b] \in B/\!\sim$ has exactly $n+1$ elements; if this is not the case, then clearly $\mathfrak{B}$ is not in $\mathcal{K}$, and hence $\mathfrak{B}$ is rejected.

(3) We define an ordered graph $G = (V, E, <_G)$ related to $\mathfrak{B}$ as follows:

- $V := \{[b] \cup [c] \cup [d] \mid [b], [c], [d] \in B/\!\sim,\ [b]^n \times [c]^n \times [d]^n \cap R_n^{\mathfrak{B}} \neq \emptyset\}$,
- $E := \{\{u, v\} \mid u, v \in V, u \neq v, u \cap v \neq \emptyset\}$,
- $<_G$ is the strict version of the lexicographic linear order on V obtained from the linear pre-order $\preceq$ (note that $\preceq$ is a linear order on $B/\!\sim$).

If $\mathfrak{B}$ is isomorphic to $\mathfrak{A}_n^{\mathrm{ev}}(G')$ for some $G' = (V', E', <_{G'})$, then clearly $G \cong G'$, and the mapping $f(\{u, v\}) = u \cap v$ defines a bijection $E \to B/\!\sim$. Thus, $\mathfrak{B}$ is rejected if G is not an ordered connected 3-regular graph, or if f is not a bijection $E \to B/\!\sim$.

(4) For each $e \in E$, let $\{b_1^e, \dots, b_{n+1}^e\}$ be an arbitrary enumeration of the equivalence class $f(e)$. For each $v \in V$, let $\mathfrak{B}_v$ be the structure $(v, R_n^{\mathfrak{B}} \cap v^{3n})$, and let $h_v \colon v \to A_n^v$ be the function $h_v(b_i^e) = a_i^e$ for each $e \in E$ such that $f(e) \subseteq v$ and each $i \in [n+1]$. Define the sets $U^+, U^- \subseteq V$ as follows:

- $U^+ := \{v \in V \mid h_v \text{ is an isomorphism } \mathfrak{B}_v \to \mathfrak{A}_n^v\}$,
- $U^- := \{v \in V \mid h_v \text{ is an isomorphism } \mathfrak{B}_v \to \tilde{\mathfrak{A}}_n^v\}$.

If $\mathfrak{B}$ is in $\mathcal{K}$, then $U^+ \cup U^- = V$. Thus, $\mathfrak{B}$ is rejected, if $U^+ \cup U^- \neq V$. On the other hand, if $U^+ \cup U^- = V$, then the function $h = \bigcup_{v \in V} h_v$ is an isomorphism $\mathfrak{B} \to \mathfrak{A}_n(G, U^-)$. By Lemma 10, $\mathfrak{A}_n(G, U^-) \cong \mathfrak{A}_n^{\mathrm{ev}}$ if and only if $|U^-|$ is even. Thus, $\mathfrak{B}$ is accepted if $|U^-|$ is even, and rejected otherwise.

Clearly each of the steps (1)–(4) of the algorithm can be realized in polynomial time. In particular, since $|v| = 3(n+1)$ for all $v \in V$ and n is a constant, the sets U^+ and U^- can be computed in polynomial time with respect to $|B|$. ◀

7 Winning the bijective colouring game

Our aim is to prove that Duplicator has a winning strategy in the bijective colouring game $\mathrm{BCG}[\mathfrak{A}_n^{\mathrm{ev}}(G), \mathfrak{A}_n^{\mathrm{od}}(G), n, k]$ provided that G is large enough with respect to the parameter k. Here being large enough is defined in terms of a game, $\mathrm{CR}_k(G)$, that is a minor variation of the *cops&robber game* introduced in [10].

The game $\mathrm{CR}_k(G)$ uses G as a board, and is played between two players, Cop and Robber. Positions of the game are pairs (F, u), where $F \subseteq E$ is such that $|F| \leq k$ and $u \in V$. If $E(u) \subseteq F$, then the game ends immediately, and Cop wins. If this is not the case, then the players play a round as follows:

- Robber chooses a path $P\colon u_0, \ldots, u_\ell$ from $u = u_0$ to some vertex $u' = u_\ell$ such that $\{u_{i-1}, u_i\} \notin F$ for all $i \in [\ell]$.
- Cop chooses an edge $e \in E$ and a set $F' \subseteq F \cup \{e\}$ such that $|F'| \leq k$.
- The next position of the game is (F', u').

Robber wins the game if Cop does not win it in a finite number of rounds.

The intuitive idea of the game is that Cop has k cop pebbles that he moves on the edges of G, and Robber moves one pebble on the vertices of G. Cop tries to capture Robber by surrounding her pebble by his cop pebbles. In each round, Robber can escape along a path that does not contain cop pebbles, and after that Cop is allowed to either add one unused cop pebble on the board, or move one cop pebble to a new position. Robber wins if she can escape forever.

▶ **Definition 15.** *Let $F \subseteq E$ be a set of edges such that $|F| \leq k$. We denote by $\mathrm{W}_k(F, G)$ the set of all vertices $u \in V$ such that Robber has a winning strategy in the game $\mathrm{CR}_k(G)$ starting from position (F, u).*

If Robber has a winning strategy in the game $\mathrm{CR}_k(G)$ from a given position, then she can choose a move in the first round in such a way that she has a winning strategy from the next position after Cop's move. We formulate this simple principle in terms of the winning set $\mathrm{W}_k(F, G)$.

▶ **Lemma 16.** *Let $F \subseteq E$ be a set of edges such that $|F| \leq k$, and let $u \in V$. If $u \in \mathrm{W}_k(F, G)$, then there exists an E-path $u_0 \ldots, u_\ell$ from $u = u_0$ to a vertex $u' = u_\ell$ such that $\{u_{i-1}, u_i\} \notin F$ for all $i \in [\ell]$, and $u' \in \mathrm{W}_k(F', G)$ for all $e \in E$ and $F' \subseteq F \cup \{e\}$ such that $|F'| \leq k$.*

In order to win the bijective colouring game for $\mathfrak{A}_n^{\mathrm{ev}}(G)$ and $\mathfrak{A}_n^{\mathrm{od}}(G)$, Duplicator has to use *edge preserving* bijections $f\colon A_n(G) \to A_n(G)$, i.e., bijections f such that $f(a_i^e) \in A_n^e$ for all $e \in E$ and $i \in [n+1]$. We denote the restriction of an edge preserving f to the set A_n^e by f_e. Note that $f_e \in \mathsf{S}(A_n^e)$ for each $e \in E$, and f is the disjoint union of the bijections f_e. Furthermore, for each $v \in V$ we denote the restriction of f to the set A_n^v by f_v. Thus, if $\vec{e}(v) = (r, s, t)$, then $f_v = f_r \cup f_s \cup f_t$.

We formulate next another important property of bijections that Duplicator will use in her strategy.

▶ **Definition 17.** *An edge preserving bijection $f\colon A_n(G) \to A_n(G)$ is* good *if there is exactly one vertex $v \in V$ such that the following holds:*

(TW) *either $v \neq v_0$ and $\mathsf{p}(f_v) = 1$, or $v = v_0$ and $\mathsf{p}(f_v) = 0$.*

If in addition $g, h\colon A_n(G) \to [n]$ are colourings and $g = h \circ f$, then f is good for g and h.

We denote the set of all good bijections by $\mathrm{GB}(G)$, and the set of all bijections that are good for g and h by $\mathrm{GB}_{gh}(G)$. Furthermore, if $f \in \mathrm{GB}(G)$, then we denote the unique vertex v such that (TW) holds by $\mathrm{tw}(f)$.

The intuition behind condition (TW) is that $\mathrm{tw}(f)$ (the "twist of f") is the only vertex v such that f_v does not preserve the relation R_n.

Note that for any $v \in V$ there are bijections $f \in \mathrm{GB}(G)$ such that $\mathrm{tw}(f) = v$. In the case $v = v_0$, this is witnessed by the identity function $\mathrm{id}_{A_n(G)}$ of $A_n(G)$. For $v \neq v_0$, this follows from the proof of Lemma 10: in the case $U = \{v_0\}$ and $U' = \{v\}$ the isomorphism f from $\mathfrak{A}(G, U)$ to $\mathfrak{A}(G, U')$ constructed in the proof is in $\mathrm{GB}(G)$ and clearly $\mathrm{tw}(f) = v$.

It is also crucial in Duplicator's strategy that the partial functions $p = \alpha \mapsto \beta$ corresponding to positions (α, β, g, h) of the game $\mathrm{BCG}[\mathfrak{A}_n^{\mathrm{ev}}(G), \mathfrak{A}_n^{\mathrm{od}}(G), n, k]$ are restrictions of bijections f that are good for g and h, and whose twist $\mathrm{tw}(f)$ is far enough from $\mathrm{dom}(p)$.

▶ **Definition 18.** *A partial function $p\colon A_n(G) \to A_n(G)$ is* good for g and h *if $|\mathrm{dom}(p)| \leq k$ and there exists a bijection $f \in \mathrm{GB}_{gh}(G)$ such that $p \subseteq f$ and $\mathrm{tw}(f) \in \mathrm{W}_k(F^p, G)$, where $F^p := \{e \in E \mid \mathrm{dom}(p) \cap A_n^e \neq \emptyset\}$. We denote the set of all partial functions that are good for g and h by $\mathrm{GP}_{gh}(G)$.*

We prove next that all good partial functions are partial isomorphisms.

▶ **Lemma 19.** *If $p \in \mathrm{GP}_{gh}(G)$, then $p \in \mathrm{PI}(\mathfrak{A}_n^{\mathrm{ev}}(G)^g, \mathfrak{A}_n^{\mathrm{od}}(G)^h)$.*

Proof. Let f be a bijection in GB_{gh} such that $p \subseteq f$ and $\mathrm{tw}(f) \in \mathrm{W}_k(F^p, G)$. Then $g = h \circ f$, whence $g(a) = h(p(a))$ for all $a \in \mathrm{dom}(p)$. Furthermore, since f satisfies the condition (TW) and $\mathrm{tw}(f) \in \mathrm{W}_k(F^p, G)$, we have $\mathsf{p}(f_v) = 0$ for every $v \in V \setminus \{v_0\}$ such that $\mathrm{dom}(p)^{3n} \cap P_n^v \neq \emptyset$, and $\mathsf{p}(f_{v_0}) = 1$ if $\mathrm{dom}(p)^{3n} \cap P_n^{v_0} \neq \emptyset$. Thus, the claim follows from Lemma 8. ◀

The next lemma is the key for defining Duplicator's strategy in the bijective colouring game $\mathrm{BCG}[\mathfrak{A}^{\mathrm{ev}}(G), \mathfrak{A}^{\mathrm{od}}(G)]$: it shows that if $p = \alpha \mapsto \beta$ is a good partial function for g and h, then there is a suitable good bijection f' that Duplicator can play in the position (α, β, g, h).

▶ **Lemma 20.** *Let $g, h\colon A_n(G) \to [n]$ be colourings. Assume that $p \in \mathrm{GP}_{gh}(G)$. Then there exists $f' \in \mathrm{GB}_{gh}(G)$ such that $p \subseteq f'$ and $\mathrm{tw}(f') \in \mathrm{W}_k(F', G)$ for any $e \in E$ and $F' \subseteq F^p \cup \{e\}$ with $|F'| \leq k$.*

Proof. By the definition of GP_{gh} there is a bijection $f \in \mathrm{GB}_{gh}(G)$ such that $p \subseteq f$ and $\mathrm{tw}(f) \in \mathrm{W}_k(F^p, G)$. Hence, by Lemma 16, there exists an E-path $u_0, \ldots, u_\ell$ from $\mathrm{tw}(f) = u_0$ to some vertex $u' = u_\ell$ such that $e_i := \{u_{i-1}, u_i\} \notin F^p$ for all $i \in [\ell]$ and $u' \in \mathrm{W}_k(F', G)$ for any $e \in E$ and $F' \subseteq F^p \cup \{e\}$ with $|F'| \leq k$. We define the bijection f' we are looking for as the union of component bijections $f'_e : A_n^e \to A_n^e$.

For all edges $e \in E$ not on the path P, we let $f'_e := f_e$. Consider then an edge e on the path P. By the pigeon hole principle, there are elements $a, b \in A_n^e$ such that $a \neq b$ and $g(a) = g(b)$. We define now f'_e as follows:

$$f'_e(c) := \begin{cases} f_e(c), & \text{if } c \notin \{a, b\} \\ f_e(b) & \text{if } c = a \\ f_e(a) & \text{if } c = b. \end{cases}$$

Note that since $g = h \circ f$, we have $h(f_e(a)) = h(f_e(b))$, whence $g(c) = h(f'_e(c))$ for all $c \in A_n^e$. Note further that $\mathsf{p}(f'_e) = \mathsf{p}(f_e) \oplus 1$.

Clearly f' is an edge preserving bijection $A_n(G) \to A_n(G)$, and since $g(c) = h(f'_e(c))$ holds for all $e \in E$ and $c \in A_n^e$, we have $g = h \circ f'$. Moreover, since F^p contains no edges of the path P, we have $p = f \restriction \mathrm{dom}(p) = f' \restriction \mathrm{dom}(p) \subseteq f'$.

To show that $f' \in \mathrm{GB}_{gh}(G)$, we observe next that

- $\mathsf{p}(f'_v) = \mathsf{p}(f_v)$ for all $v \notin \{u_0, \ldots, u_\ell\}$,
- $\mathsf{p}(f'_{u_i}) = \mathsf{p}(f'_{e_i}) \oplus \mathsf{p}(f'_{e_{i+1}}) \oplus \mathsf{p}(f_e) = (\mathsf{p}(f_{e_i}) \oplus 1) \oplus (\mathsf{p}(f_{e_{i+1}}) \oplus 1) \oplus \mathsf{p}(f_e) = \mathsf{p}(f_{u_i})$ for each $i \in [\ell - 1]$, where e is the third edge adjacent to u_i, and
- $\mathsf{p}(f'_{u_i}) = \mathsf{p}(f'_{e_j}) \oplus \mathsf{p}(f_e) \oplus \mathsf{p}(f_{e'}) = (\mathsf{p}(f_{e_j}) \oplus 1) \oplus \mathsf{p}(f_e) \oplus \mathsf{p}(f_{e'}) = \mathsf{p}(f_{u_i}) \oplus 1$ for $(i, j) \in \{(0, 1), (\ell, \ell)\}$, where e and e' are the other two edges adjacent to u_i.

Thus we see that there is a unique $v \in V$ such that (TW) holds for f', and this unique vertex is $\mathrm{tw}(f') = u'$. Finally, by the choice of u', we have $\mathrm{tw}(f') \in \mathrm{W}_k(F', G)$ for any $e \in E$ and $F' \subseteq F^p \cup \{e\}$ with $|F'| \leq k$. ◀

Putting together the previous lemmas we can now prove that the strategy based on good bijections and good partial functions guarantees a win for Duplicator in the bijective colouring game.

▶ **Theorem 21.** *Assume that* $\mathrm{W}_k(\emptyset, G) \neq \emptyset$. *Then there are colourings* $g, h \colon A_n(G) \to [n]$ *such that Duplicator has a winning strategy in the game* $\mathrm{BCG}[\mathfrak{A}_n^{\mathrm{ev}}(G), \mathfrak{A}_n^{\mathrm{od}}(G), n, k](\emptyset, \emptyset, g, h)$.

Proof. Let g_0 and h_0 be the trivial colourings $A_n(G) \to [n]$ defined by $g_0(a) = h_0(a) = 1$ for all $a \in A_n(G)$. We will show that Duplicator has a winning strategy in the game $\mathrm{BCG}[\mathfrak{A}_n^{\mathrm{ev}}(G), \mathfrak{A}_n^{\mathrm{od}}(G), n, k](\emptyset, \emptyset, g_0, h_0)$. It suffices to show that Duplicator can guarantee that the condition

$$(*) \quad \alpha \mapsto \beta \in \mathrm{GP}_{gh}(G)$$

holds in every position (α, β, g, h) during the play. Indeed, by Lemma 19, this implies that $\alpha \mapsto \beta \in \mathrm{PI}(\mathfrak{A}^{\mathrm{ev}}(G)^g, \mathfrak{A}^{\mathrm{od}}(G)^h)$, and hence Spoiler cannot win the game.

Since $\mathrm{W}_k(\emptyset, G) \neq \emptyset$, and $g_0 = h_0 \circ f$ holds for any bijection $f : A_n(G) \to A_n(G)$, there is a bijection $f_0 \in \mathrm{GB}_{gh}(G)$ such that $\mathrm{tw}(f_0) \in \mathrm{W}_k(\emptyset, G)$. Clearly $\emptyset \mapsto \emptyset = \emptyset \subseteq f_0$, and $F^\emptyset = \emptyset$. Thus f_0 witnesses the fact that $\emptyset \mapsto \emptyset \in \mathrm{GP}_{g_0 h_0}(G)$, whence condition $(*)$ holds for the starting position $(\emptyset, \emptyset, g_0, h_0)$ of the game.

Assume then that Duplicator and Spoiler have reached a position (α, β, g, h) such that condition $(*)$ holds. Let $p := \alpha \mapsto \beta$. By Lemma 20 there exists a bijection $f' \in \mathrm{GB}_{gh}(G)$ such that $p \subseteq f'$ and

$$(\dagger) \quad \mathrm{tw}(f') \in \mathrm{W}_k(F', G) \text{ for any } e \in E \text{ and } F' \subseteq F^p \cup \{e\} \text{ with } |F'| \leq k.$$

Let Duplicator play f' as her move. We consider separately the two options Spoiler has for his answer:

(1) Spoiler plays an element move by choosing a variable $y \in X_k$ and an element $a \in A_n(G)$. The next position of the game is $(\alpha', \beta', g', h')$, where $\alpha' = \alpha[a/y]$, $\beta' = \beta[f'(a)/y]$, $g' = g$ and $h' = h$. Thus, $f' \in \mathrm{GB}_{g'h'}(G) = \mathrm{GB}_{gh}(G)$. Let $p' := \alpha' \mapsto \beta'$. Since $p \subseteq f'$ and $p' \subseteq p \cup \{(a, f'(a))\}$, we have $p' \subseteq f'$. Furthermore, $|F^{p'}| \leq k$ and $F^{p'} \subseteq F^p \cup \{e\}$, where e is the edge such that $a \in A_n^e$, whence by condition $(\dagger)$, we have $\mathrm{tw}(f') \in \mathrm{W}_k(F^{p'}, G)$. Thus f' witnesses that condtion $(*)$ holds in the position $(\alpha', \beta', g', h')$.

(2) Spoiler plays a colouring move by choosing a function $g' \colon A_n(G) \to [n]$. The next position is $(\alpha', \beta', g', h')$, where $\alpha' = \alpha$, $\beta' = \beta$ and $h' \colon A_n(G) \to [n]$ is the unique function such that $g' = h' \circ f'$. Since $f' \in \mathrm{GB}(G)$ and $g' = h' \circ f'$, we have $f' \in \mathrm{GB}_{g'h'}(G)$. Let $p' := \alpha' \mapsto \beta'$. Then $p' = p \subseteq f'$ by the choice of f'. Using condition $(\dagger)$ in the special case $F' = F^p = F^{p'}$, we see that $\mathrm{tw}(f') \in \mathrm{W}_k(F^{p'}, G)$. Thus we conclude that $\alpha' \mapsto \beta' = p' \in \mathrm{GP}_{g'h'}(G)$, as desired. ◀

By Proposition 12, $\mathfrak{A}_n^{\mathrm{ev}}(G) \in \mathsf{CSP}(\mathfrak{C}_n)$ and $\mathfrak{A}_n^{\mathrm{od}}(G) \notin \mathsf{CSP}(\mathfrak{C}_n)$. Furthermore, it is straightforward to show that for every k there is a graph G such that $\mathrm{W}_k(\emptyset, G) \neq \emptyset$.[3] Hence, by Theorem 21 and Corollary 6, for every k there are structures $\mathfrak{A}_k \in \mathsf{CSP}(\mathfrak{C}_n)$ and $\mathfrak{B}_k \notin \mathsf{CSP}(\mathfrak{C}_n)$ such that $\mathfrak{A}_k \equiv^k_{\infty\omega,n} \mathfrak{B}_k$. Thus we obtain the following size hierarchy result for CSP-quantifiers.

[3] Such graphs were constructed in Section 8 of [10] for the similar cops&robber game, which is more difficult for Robber to win.

▶ **Corollary 22.** *For any integer $n \geq 2$, there is a CSP-quantifier $Q_{\mathfrak{C}}$ with $\mathrm{sz}(\mathfrak{C}) = n + 1$ which is not definable in $L^{\omega}_{\infty\omega}(\mathbf{CSP}^{+}_{n})$.*

Although by Proposition 13, $\mathsf{CSP}(\mathfrak{C}_n)$ is NP-complete, we can still use the structures $\mathfrak{A}^{\mathrm{ev}}_n(G)$ and $\mathfrak{A}^{\mathrm{od}}_n(G)$ for proving that PTIME cannot be captured by adding CSP-quantifiers with bounded size templates to inflationary fixed point logic with counting. A suitable PTIME-computable property separating $\mathfrak{A}^{\mathrm{ev}}(G)$ and $\mathfrak{A}^{\mathrm{od}}(G)$ is provided by Proposition 14.

▶ **Corollary 23.** *For any integer $n \geq 2$, there is a* PTIME*-computable quantifier $Q_{\mathcal{K}}$ which is not definable in* $\mathrm{IFPC}(\mathbf{CSP}_n)$.

8 Conclusion

In this paper, we introduce two pebble games for extensions of the infinitary k-variable logic $L^{k}_{\infty\omega}$ by CSP-quantifiers. We prove that the first of these games, $\mathrm{CSPG}[\mathfrak{A}, \mathfrak{B}, n, k]$ characterizes equivalence of $\mathfrak{A}$ and $\mathfrak{B}$ with respect to $L^{k}_{\infty\omega}(\mathbf{CSP}^{+}_{n})$, where $\mathbf{CSP}^{+}_{n}$ is the union of the class of all CSP-quantifiers $\mathsf{CSP}(\mathfrak{C})$ with the size of $\mathfrak{C}$ at most n and the class $\mathbf{Q}_1$ of all unary quantifiers. Furthermore, we prove that the second game, BCG, corresponds to at least as strong equivalence as CSPG: if Duplicator has a winning strategy in $\mathrm{BCG}[\mathfrak{A}, \mathfrak{B}, n, k]$, then she has one in $\mathrm{CSPG}[\mathfrak{A}, \mathfrak{B}, n, k]$.

As our main contribution in the paper, we prove a size hierarchy result for CSP-quantifiers: for all $n \geq 2$ there is a CSP-quantifier $Q_{\mathfrak{C}_n}$ of size $n+1$ which is not definable in $L^{\omega}_{\infty\omega}(\mathbf{CSP}^{+}_{n})$. For proving this, we introduce a new variation of the CFI construction, and use the game BCG for showing that the structures $\mathfrak{A}^{\mathrm{ev}}_n(G)$ and $\mathfrak{A}^{\mathrm{od}}_n(G)$ obtained in the construction are equivalent with respect to $L^{k}_{\infty\omega}(\mathbf{CSP}^{+}_{n})$.

We conclude the paper by listing some open problems:

(1) The CSP-quantifiers $Q_{\mathfrak{C}_n}$ that we use in the proof of the hierarchy result, Corollary 22, are NP-complete. Is it possible to separate the structures $\mathfrak{A}^{\mathrm{ev}}_n(G)$ and $\mathfrak{A}^{\mathrm{od}}_n(G)$ by some PTIME-computable CSP-quantifier?

(2) The arity of the CSP-quantifier $Q_{\mathfrak{C}_n}$ is $3n$. Is it possible to find templates $\mathfrak{D}_n$, $n \geq 2$, such that $Q_{\mathfrak{D}_n}$ is not definable in $L^{\omega}_{\infty\omega}(\mathbf{CSP}^{+}_{n})$ and $\mathrm{ar}(\mathfrak{D}_n) = r$ for some constant r?

(3) We do not allow vectorization (i.e., interpreting ℓ-tuples as elements for some ℓ) in the definition of $L(Q_{\mathcal{K}})$, unlike is done in many other recent papers on the topic (e.g., the papers [5, 6, 9, 14, 4] on rank logic). It is not difficult to define a version on the CSP game that works for the ℓth vectorizations of quantifiers in $\mathbf{CSP}^{+}_{n}$. However, using such games would probably be extremely hard (if not impossible), since they involve existential second-order quantification of ℓ-ary relations. Furthermore, it seems quite plausible that for any $\ell, n \geq 2$, equivalence with respect to the extension of $L^{k}_{\infty\omega}$ by all ℓth vectorizations of the quantifiers in $\mathbf{CSP}^{+}_{n}$ is just isomorphism for large enough k. The challenge is to prove or disprove this.

(4) What is the relationship between $L^{\omega}_{\infty\omega}(\mathbf{CSP}^{+}_{n})$ and the extension $\mathrm{LA}^{\omega}(Q)$ of $L^{\omega}_{\infty\omega}$ with all linear algebraic operators (see [4])? Does equivalence with respect to one of these logics imply equivalence with respect to the other? We believe that the answer to the latter question is "no". This is because $\mathrm{LA}^{\omega}(Q)$ is closed under vectorizations, but it is not plausible that $L^{\omega}_{\infty\omega}(\mathbf{CSP}^{+}_{n})$ is, and closing it with respect to vectorizations could well lead to a logic whose equivalence is isomorphism (see the previous item).

References

1 Albert Atserias, Andrei A. Bulatov, and Anuj Dawar. Affine systems of equations and counting infinitary logic. *Theor. Comput. Sci.*, 410(18):1666–1683, 2009. doi:10.1016/j.tcs.2008.12.049.

2 Andrei A. Bulatov. A dichotomy theorem for nonuniform csps. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.

3 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Comb.*, 12(4):389–410, 1992. doi:10.1007/BF01305232.

4 Anuj Dawar, Erich Grädel, and Wied Pakusa. Approximations of isomorphism and logics with linear-algebraic operators. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 112:1–112:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.112.

5 Anuj Dawar, Martin Grohe, Bjarki Holm, and Bastian Laubner. Logics with rank operators. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 113–122. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.24.

6 Anuj Dawar and Bjarki Holm. Pebble games with algebraic rules. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 251–262. Springer, 2012. doi:10.1007/978-3-642-31585-5_25.

7 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

8 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

9 Erich Grädel and Wied Pakusa. Rank logic is dead, long live rank logic! In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 390–404. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.390.

10 Lauri Hella. Logical hierarchies in PTIME. *Inf. Comput.*, 129(1):1–19, 1996. doi:10.1006/inco.1996.0070.

11 Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In Alan L. Selman, editor, *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pages 59–81. Springer New York, New York, NY, 1990. doi:10.1007/978-1-4612-4478-3_5.

12 Phokion G. Kolaitis and Moshe Y. Vardi. A logical approach to constraint satisfaction. In Nadia Creignou, Phokion G. Kolaitis, and Heribert Vollmer, editors, *Complexity of Constraints – An Overview of Current Research Themes [Result of a Dagstuhl Seminar]*, volume 5250 of *Lecture Notes in Computer Science*, pages 125–155. Springer, 2008. doi:10.1007/978-3-540-92800-3_6.

13 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

14 Moritz Lichter. Separating rank logic from polynomial time. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 – July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470598.

15 Per Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, 1966. doi:10.1111/j.1755-2567.1966.tb00600.x.

16 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

Complexity of Polyadic Boolean Modal Logics: Model Checking and Satisfiability

Reijo Jaakkola
Tampere University, Finland

Abstract

We study the computational complexity of model checking and satisfiability problems of polyadic modal logics extended with permutations and Boolean operators on accessibility relations. First, we show that the combined complexity of the model checking problem for the resulting logic is PTIME-complete. Secondly, we show that the satisfiability problem of polyadic modal logic extended with negation on accessibility relations is EXPTIME-complete. Finally, we show that the satisfiability problem of polyadic modal logic with permutations and Boolean operators on accessibility relations is EXPTIME-complete, under the assumption that both the number of accessibility relations that can be used and their arities are bounded by a constant. If NEXPTIME is not contained in EXPTIME, then this assumption is necessary, since already the satisfiability problem of modal logic extended with Boolean operators on accessibility relations is NEXPTIME-hard.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Polyadic modal logics, Boolean modal logics, Model checking, Satisfiability

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.26

Acknowledgements I want to thank Antti Kuusisto for suggesting this topic to me and for discussing it with me on numerous occasions. I also want to thank the anonymous reviewers for their detailed comments which greatly improved the presentation of this paper.

1 Introduction

In recent years there has been increasing interest in generalizing complexity results for logics that only have access to relation symbols of arity at most two, to logics that have access to relations of arbitrary high arity, which we will call their *polyadic* extensions, see for example [2, 12, 13, 14, 15, 18, 19, 21]. In [12] the authors introduced the uniform one-dimensional logic U_1, which is a polyadic extension of the two-variable logic FO^2. It was proved in [18] that the complexity of U_1 satisfiability problem is NEXPTIME-complete, which is the same as for FO^2 [11]. In the very recent work [2] the forward guarded fragment FGF was introduced, which is a polyadic extension of the description logic $\mathcal{ALC}$ with global diamond. In the same work it was established that its satisfiability and conjunctive query entailment problems have the same complexity as the corresponding problems in the case of $\mathcal{ALC}$, i.e., both are EXPTIME-complete. In the two aforementioned examples the complexities of the base logic and its polyadic extension coincided, but there are also examples where the polyadic extension has a higher complexity. For instance, in [17] it is proved that the satisfiability problem of guarded U_1, which is a polyadic extension of guarded FO^2, is NEXPTIME-complete, while the satisfiability problem of guarded FO^2 is EXPTIME-complete [8].

While lifting complexity results from base logics to their polyadic extensions is, at least to the author, already intrinsically interesting, it also has more applied motivations stemming from, say, database theory, since polyadic relations occur naturally in various contexts and having access to them can be advantageous. For instance, a simple relation such as "Alice received a message M from Bob" is best viewed as a ternary relation. One application area where the potential need for polyadic logics has been recognized is the very active field of *description logics* [1]. Indeed, given that several description logics used in knowledge

representation can only access binary roles, it should not come as a surprise that several polyadic *description logics* have been already suggested in the literature [5, 26, 29]. Finding polyadic extensions of modal logics can be seen as attempts at finding natural polyadic description logics.

To make the research around polyadic extensions more systematic, in [14] the authors outlined a research program for systematically extending complexity results for description logics – and modal logics more generally – into their polyadic counterparts, following the ideas presented in [20]. The main idea is the following: since description logics can be seen as standard modal logic extended with *relation operators* – such as inverse roles and counting – and the standard modal logic has a canonical extension into a polyadic modal logic (namely the extension in which diamonds can bind multiple formulas), one can easily obtain quite canonical extensions of known description logics[1]. To demonstrate their research program in action, the polyadic extension of $\mathcal{ALCQI}$, i.e., the extension of $\mathcal{ALC}$ with inverse roles and counting, was studied in [14]. The authors proved that the concept satisfiability problem for this logic is PSPACE-complete, which is the same as in the case of $\mathcal{ALCQI}$.

The main purpose of the present work is to contribute to the above research program in the context of Boolean modal logics, which are modal logics extended with Boolean operators on accessibility relations [23]. In these logics one can for example write down formulas such as $\langle \neg R \rangle p$, which expresses that one can reach a world in which p is true via the complement of the accessibility relation R. While being very natural modal logics as such, they are also closely related to other interesting logics such as modal logics extended with the window operator [7, 23], the two-variable logic [27] and the uniform one-dimensional fragment [20]. They have also been considered in the context of description logics [25].

Boolean modal logics present novel and intriguing technical challenges, especially when studying the complexity of their satisfiability problems. Indeed, the most common explanation for the good algorithmic properties of modal logics is that they often enjoy some variant of the tree-model property [9, 28, 30], which Boolean modal logics lack. This is a consequence of the fact that Boolean modal logics have access to *complements* of accessibility relations. To clarify this important point, we point out that in Boolean modal logics one can write statements such as $[R][\neg R]\bot \wedge [\neg R][\neg R]\bot$, which enforce that R needs to be interpreted in the Kripke models of this sentence as the total binary relation. Clearly such sentences do not have a tree-model property.

Satisfiability problems of Boolean modal logics were first studied in [23], where it was proved that modal logic with negations of accessibility relations ML($\neg$) has EXPTIME-complete satisfiability problem, while the complexity of full Boolean modal logic ML($\neg, \cap$) is NEXPTIME-complete. As a follow-up to this work, in [27] authors studied the complexity of the satisfiability problem for Boolean modal logic extended with inverses of accessibility relations and equality ML($I, s, \neg, \cap$), and proved that if the number of accessibility relations is bounded by a constant, then its satisfiability problem is EXPTIME-complete. Since FO^2 has the same expressive power as ML($I, s, \neg, \cap$) [27], the aforementioned complexity result should be contrasted with the fact that the satisfiability problem of FO^2 is NEXPTIME-hard already over unary vocabularies.

In this work we partially extend these results to the polyadic case. First, we will prove that the satisfiability problem of polyadic ML($\neg$), which we denote by PML($\neg$), has EXPTIME-complete satisfiability problem. Secondly, we will partially generalize the complexity result

[1] Of course, some conceptual work is needed to figure out what are the canonical polyadic extensions of the relevant role operators.

of [27] by proving that polyadic Boolean modal logic extended with arbitrary permutations of accessibility relations (as opposed to just inverses of binary accessibility relations), which we denote by PML($p, s, \neg, \cap$), has ExpTime-complete satisfiability problem, if the number of accessibility relations in the underlying vocabulary and *their arities* are bounded by a constant. We emphasize again that, if ExpTime $\neq$ NExpTime, then the aforementioned assumption is necessary, since in general the satisfiability problem of PML($p, s, \neg, \cap$) is NExpTime-complete (see Proposition 3).

An important technical contribution of the present work is that we prove these two results in a *unified manner*. Originally, the complexity of ML($\neg$) was established using automata theory [23], while the complexity of ML($I, s, \neg, \cap$) over vocabularies with at most a constant number of accessibility relations was established via a poly-time reduction to the satisfiability problem of modal logic with *difference operator* over a restricted class of Kripke frames. In contrast, we establish upper bounds on the satisfiability problems of PML($\neg$) and PML($p, s, \neg, \cap$) by simply using elementary (albeit in the second case quite technical) reductions to the satisfiability problem of polyadic ML extended with global diamond $\langle \mathrm{E} \rangle$, which we denote by PML $+ \langle \mathrm{E} \rangle$. These reductions were inspired by the reduction used in [27] to establish the complexity of ML($I, s, \neg, \cap$) over vocabularies with at most a constant number of accessibility relations. The overall structure of these reductions is quite robust and hence we expect that similar reductions will yield in the future several extensions of the results presented in this paper.

In addition to studying satisfiability problems, we also study the combined complexity of the model checking problem of PML($p, s, \neg, \cap$), which is the variant where both the model and the formula itself are received as part of the input, and prove that it is PTime-complete, the non-trivial part being the upper bound. It is well-known that model checking problems of various modal logics with tree-model property – even quite expressive ones such as the guarded fragment – are PTime-complete [3, 10]. However, besides these modal logics and finite variable fragments of first-order logic [10], it seems that there are not that many natural logics known for which the complexity of the model checking problem lies in PTime. Note that the formulas of PML($p, s, \neg, \cap$) are neither guarded nor are they expressible in any finite variable fragment of first-order logic. Thus, even though proving that the model checking problem of PML($p, s, \neg, \cap$) is in PTime turns out to be quite straightforward, we still believe that the result is interesting since the logic PML($p, s, \neg, \cap$) is quite distinct from other logics known in the literature with PTime-complete combined complexity. Indeed, it would be interesting to understand how much PML($p, s, \neg, \cap$) could be extended while keeping its combined complexity feasible.

The structure of this paper is as follows. First, in Section 2 we will formally define the logics that are studied in this paper. Then, in Sections 3 and 4 we prove that the satisfiability problems of PML($\neg$) and PML($p, s, \neg, \cap$) respectively are ExpTime-complete, the latter under the assumption that the number of accessibility relations and their arities are bounded by a constant, which, as pointed out earlier, is necessary if ExpTime $\neq$ NExpTime. Finally, in Section 5 we prove that the combined complexity of PML($p, s, \neg, \cap$) is PTime-complete.

2 Preliminaries

In this paper we will only consider relation symbols of arity at least two. Given a relation symbol R, we will use $ar(R)$ to denote its arity. If τ is a set of relation symbols and Φ is a set of propositional symbols, then a *Kripke-model over* (τ, Φ) is a tuple $\mathfrak{M} = (W, (R^{\mathfrak{M}})_{R \in \tau}, V)$, where

1. W is a non-empty set (the set of possible worlds),
2. for every $R \in \tau$ we have that $R^{\mathfrak{M}} \subseteq W^{ar(R)}$ and
3. $V : \Phi \to \mathcal{P}(W)$.

Members of τ are also called *accessibility relations*. In what follows we will never specify explicitly what the underlying set of propositional symbols is and we will only occasionally specify the set of accessibility relations.

In this paper we consider extensions of standard polyadic (multimodal) modal logic via *relation operators*, which are essentially mappings that map relational structures to relational structures and which are invariant under isomorphisms. Since we will focus our attention only on a specific set of relation operators, we will omit the formal definition of a relation operator here, which can be found in [16].

The main relation operators that we are going to consider are $p, s, \neg, \cap$, where p and s are called *cyclic permutation* and *swap permutation* respectively. Furthermore, for technical reasons we are going to need two additional relation operators $\setminus, \cup$. Let τ denote a set of relation symbols. Given $k \geq 2$, the set of *k-ary terms* $\mathrm{GRA}_k(p, s, \neg, \cap, \setminus, \cup)[\tau]$ is generated by the following grammar

$$\mathcal{R} ::= R \mid p\mathcal{R} \mid s\mathcal{R} \mid \neg\mathcal{R} \mid \mathcal{R} \cap \mathcal{R} \mid \mathcal{R} \setminus \mathcal{R} \mid \mathcal{R} \cup \mathcal{R},$$

where $R \in \tau$ is a k-ary relation. We use $\mathrm{GRA}(p, s, \neg, \cap, \setminus, \cup)[\tau]$ to denote

$$\bigcup_{k \geq 2} \mathrm{GRA}_k(p, s, \neg, \cap, \setminus, \cup)[\tau],$$

which is the set of all terms. Arity of a term $\mathcal{R}$ is denoted by $ar(\mathcal{R})$.

Let $\mathfrak{M}$ be a Kripke model over τ and let $\mathcal{R} \in \mathrm{GRA}(p, s, \neg, \cap, \setminus, \cup)$ be a k-ary term. We define the *interpretation* of $\mathcal{R}$ over $\mathfrak{M}$ recursively as follows.

1. If $\mathcal{R} = R \in \tau$, then we define $[\![R]\!]_{\mathfrak{M}} = R^{\mathfrak{M}}$.
2. If $\mathcal{R} = p\mathcal{R}'$, then we define
$$[\![\mathcal{R}]\!]_{\mathfrak{M}} = \{(a_k, a_1, \ldots, a_{k-1}) \in W^k \mid (a_1, \ldots, a_k) \in [\![\mathcal{R}']\!]_{\mathfrak{M}}\}$$
3. If $\mathcal{R} = s\mathcal{R}'$, then we define
$$[\![\mathcal{R}]\!]_{\mathfrak{M}} = \{(a_1, \ldots, a_{k-2}, a_k, a_{k-1}) \in W^k \mid (a_1, \ldots, a_k) \in [\![\mathcal{R}']\!]_{\mathfrak{M}}\}$$
4. If $\mathcal{R} = \neg\mathcal{R}'$, then we define $[\![\mathcal{R}]\!]_{\mathfrak{M}} = W^k \setminus [\![\mathcal{R}']\!]_{\mathfrak{M}}$
5. If $\mathcal{R} = \mathcal{R}' \cap \mathcal{R}''$, then we define $[\![\mathcal{R}]\!]_{\mathfrak{M}} = [\![\mathcal{R}']\!]_{\mathfrak{M}} \cap [\![\mathcal{R}'']\!]_{\mathfrak{M}}$
6. If $\mathcal{R} = \mathcal{R}' \setminus \mathcal{R}''$, then we define $[\![\mathcal{R}]\!]_{\mathfrak{M}} = [\![\mathcal{R}']\!]_{\mathfrak{M}} \setminus [\![\mathcal{R}'']\!]_{\mathfrak{M}}$
7. If $\mathcal{R} = \mathcal{R}' \cup \mathcal{R}''$, then we define $[\![\mathcal{R}]\!]_{\mathfrak{M}} = [\![\mathcal{R}']\!]_{\mathfrak{M}} \cup [\![\mathcal{R}'']\!]_{\mathfrak{M}}$

Given $k \geq 2$, we let S_k denote the set of all bijections $\{1, \ldots, k\} \to \{1, \ldots, k\}$. It is a well-known group theoretic fact that every permutation $\sigma \in S_k$ can be obtained by composing cyclic permutation with swap permutation. Given this, we will adopt the notational convention that we will use $\sigma \in S_k$ to denote some fixed (possibly empty) sequence consisting of operators p and s that generates it. The only explicit requirement that we impose on this sequence is that it should not be possible to rewrite it into a smaller one. Thus, for example, we use the identity permutation to denote the empty sequence.

The following lemma collects some elementary algebraic identities for terms.

▶ **Lemma 1.** *Let $\mathcal{R}_1, \mathcal{R}_2 \in \mathrm{GRA}(p, s, \neg, \cap, \setminus, \cup)[\tau]$ be k-ary terms and let $\sigma \in S_k$. Let $\mathfrak{M}$ be a Kripke model over τ.*

1. $[\![\neg\neg\mathcal{R}_1]\!]_{\mathfrak{M}} = [\![\mathcal{R}_1]\!]_{\mathfrak{M}}$
2. $[\![\sigma\neg\mathcal{R}_1]\!]_{\mathfrak{M}} = [\![\neg\sigma\mathcal{R}_1]\!]_{\mathfrak{M}}$
3. $[\![\sigma(\mathcal{R}_1 \cap \mathcal{R}_2)]\!]_{\mathfrak{M}} = [\![(\sigma\mathcal{R}_1 \cap \sigma\mathcal{R}_2)]\!]_{\mathfrak{M}}$
4. $[\![\sigma(\mathcal{R}_1 \cup \mathcal{R}_2)]\!]_{\mathfrak{M}} = [\![(\sigma\mathcal{R}_1 \cup \sigma\mathcal{R}_2)]\!]_{\mathfrak{M}}$
5. $[\![\neg(\mathcal{R}_1 \cap \mathcal{R}_2)]\!]_{\mathfrak{M}} = [\![(\neg\mathcal{R}_1 \cup \neg\mathcal{R}_2)]\!]_{\mathfrak{M}}$
6. $[\![\neg(\mathcal{R}_1 \cup \mathcal{R}_2)]\!]_{\mathfrak{M}} = [\![(\neg\mathcal{R}_1 \cap \neg\mathcal{R}_2)]\!]_{\mathfrak{M}}$
7. $[\![(\mathcal{R}_1 \cap \neg\mathcal{R}_2)]\!]_{\mathfrak{M}} = [\![(\mathcal{R}_1 \setminus \mathcal{R}_2)]\!]_{\mathfrak{M}}$
8. $[\![(\mathcal{R}_1 \cup \neg\mathcal{R}_2)]\!]_{\mathfrak{M}} = [\![\neg(\mathcal{R}_2 \setminus \mathcal{R}_1)]\!]_{\mathfrak{M}}$

Let $\mathcal{R} \in \mathrm{GRA}(p, s, \neg)[\tau]$ be a k-ary term. We say that $\mathcal{R}$ is a *k-literal over τ*, if it is either of the form $\neg\sigma R$ or σR, for a k-ary relation symbol $R \in \tau$. A maximally consistent set ρ of k-literals over σ is called a *k-table* over τ. We identify tables ρ with terms $\bigcap_{\alpha\in\rho} \alpha$.

Given a Kripke model $\mathfrak{M}$ over τ and $(w_1, \ldots, w_k) \in W^k$ we say that $(w_1, \ldots, w_k)$ *realizes* a k-table ρ, if for every k-literal α we have that

$$(w_1, \ldots, w_k) \in [\![\alpha]\!]_{\mathfrak{M}} \Leftrightarrow \alpha \in \rho.$$

Note that since tables are maximally consistent, each tuple in a given Kripke model realizes a unique table. Conversely, the accessibility relations of a Kripke model can be described completely by specifying what tables different tuples realize.

Given a k-table ρ and $\sigma \in S_k$ we define

$$\sigma[\rho] := \bigcap_{\sigma' R \in \rho} (\sigma \circ \sigma')R \cap \bigcap_{\neg\sigma' R \in \rho} \neg(\sigma \circ \sigma')R.$$

Note that for every k-table ρ and $\sigma_1, \sigma_2 \in S_k$ we have that $\sigma_1[\sigma_2[\rho]] = (\sigma_1 \circ \sigma_2)[\rho]$. Furthermore, we note that it can be the case that $\sigma[\rho] = \rho$, even if σ is not the identity permutation.[2]

Let Φ be a set of propositional symbols, τ a set of relation symbols and $\mathcal{F} \subseteq \{p, s, \neg, \cap, \setminus, \cup\}$. The set of formulas $\mathrm{PML}(\mathcal{F})[\tau, \Phi]$ is generated by the following grammar

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \langle\mathcal{R}\rangle(\underbrace{\varphi, \ldots, \varphi}_{k\text{-times}})$$

where $p \in \Phi$ and $\mathcal{R} \in \mathrm{GRA}(\mathcal{F})[\tau]$ is a $(k+1)$-ary term. We will use $\mathrm{PML}(\mathcal{F})[\tau, \Phi] + \langle\mathrm{E}\rangle$ to denote the set of formulas generated by the grammar of $\mathrm{PML}(\mathcal{F})[\tau, \Phi]$ extended with the rule

$$\varphi ::= \langle\mathrm{E}\rangle\varphi$$

If τ contains only binary relation symbols, then we emphasize this by writing $\mathrm{ML}(\mathcal{F})[\tau, \Phi]$.

Given a natural number c a vocabulary τ is called *c-bounded*, if $|\tau| \leq c$ and for every $R \in \tau$ we have that $ar(R) \leq c$. For example, the empty vocabulary is 0-bounded and a binary vocabulary with five relation symbols is 5-bounded. If we are only considering c-bounded vocabularies, then we emphasize this by writing PML_c. Thus $\mathrm{PML}_c[\tau, \Phi]$ entails that τ is c-bounded. We emphasize that c should be thought of as a fixed constant.

[2] Consider for example the permutation s and the 2-table $\{R, sR\}$ over $\{R\}$, where R is a binary relation.

As in the case of Kripke models, for the rest of this paper we will never explicitly mention the underlying set of propositional symbols. We will use standard shorthand notations: $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$, $[\mathcal{R}] := \neg\langle\mathcal{R}\rangle\neg$ and $\langle \mathrm{A}\rangle = \neg\langle \mathrm{E}\rangle\neg$. Given a formula φ we let $\mathrm{Subf}(\varphi)$ denote the set of subformulas of φ.

The semantics of our logics are standard and we will only recall here the semantic clauses of $\langle\mathcal{R}\rangle(\psi_1, \ldots, \psi_k)$ and $\langle \mathrm{E}\rangle\psi$. Given a Kripke model $\mathfrak{M}$ over τ and $w \in W$ we define

$\mathfrak{M}, w \Vdash \langle\mathcal{R}\rangle(\psi_1, \ldots, \psi_k) \Leftrightarrow$

there exists $(w, w_1, \ldots, w_k) \in [\![\mathcal{R}]\!]_{\mathfrak{M}}$ such that $\mathfrak{M}, w_\ell \Vdash \psi_\ell$, for every $1 \leq \ell \leq k$

and

$\mathfrak{M}, w \Vdash \langle \mathrm{E}\rangle\psi \Leftrightarrow$ there exists some $w' \in W$ such that $\mathfrak{M}, w' \Vdash \psi$.

We will conclude this section by stating the complexity of the satisfiability problem of $\mathrm{PML} + \langle \mathrm{E}\rangle$.

▶ **Proposition 2.** *The satisfiability problem of* $\mathrm{PML} + \langle \mathrm{E}\rangle$ *is* ExpTime*-complete.*

Proof. Lower bound follows from the well-known fact that the satisfiability problem of $\mathrm{ML} + \langle \mathrm{E}\rangle$ is ExpTime-complete [4, Exercise 6.8.1]. Upper bound follows, say, from the recent result that the satisfiability problem of the so-called *guarded forward fragment* FGF is ExpTime-complete [2]. Indeed, by using variables carefully, one can guarantee that the standard translation of $\mathrm{PML} + \langle \mathrm{E}\rangle$ into FO produces sentences of FGF. ◀

As mentioned in the introduction, existing results in the literature imply that the satisfiability problem of $\mathrm{PML}(p, s, \neg, \cap)$ is NExpTime-complete.

▶ **Proposition 3.** *The satisfiability problem of* $\mathrm{PML}(p, s, \neg, \cap)$ *is* NExpTime*-complete.*

Proof. Lower bound follows from the fact that the satisfiability problem of $\mathrm{ML}(\neg, \cap)$ is already NExpTime-hard [24]. For upper bound we note that $\mathrm{PML}(p, s, \neg, \cap)$ is contained in U_1 [20, Theorem 7], for which the satisfiability problem is NExpTime-complete [18]. ◀

3 Satisfiability problem of polyadic modal logic with negation

In this section we will show how the satisfiability problem of $\mathrm{PML}(\neg)$ can be reduced in polynomial time to that of $\mathrm{PML} + \langle \mathrm{E}\rangle$, which will yield the following result.

▶ **Theorem 4.** *The satisfiability problem of* $\mathrm{PML}(\neg)$ *is* ExpTime*-complete.*

It seems most probable that the above complexity result continues to hold for $\mathrm{PML}(p, s, \neg)$, but we have not yet been able to show this.

Before presenting the reduction from $\mathrm{PML}(\neg)$ to $\mathrm{PML} + \langle \mathrm{E}\rangle$, we first describe one brief application of Theorem 4, which reflects one original motivation for the study of Boolean modal logics [24]. Namely, we show that the logic PML extended with *polyadic window operator*, which we denote by $\mathrm{PML} + \nabla$, has an ExpTime-complete satisfiability problem. Consider a vocabulary τ. The grammar for generating the formulas of $\mathrm{PML}[\tau] + \nabla$ is the grammar of $\mathrm{PML}[\tau]$ extended with the rule

$$\varphi ::= \nabla_R(\underbrace{\varphi, \ldots, \varphi}_{k\text{-times}}),$$

where R is a $(k+1)$-ary accessibility relation, for every $R \in \tau$. The semantics of formulas of the form $\nabla_R(\psi_1, \ldots, \psi_k)$ is defined as follows:

$\mathfrak{M}, w \Vdash \nabla_R(\psi_1, \ldots, \psi_k) \Leftrightarrow$ For every $(w_1, \ldots, w_k) \in W^k$ we have that if

$\mathfrak{M}, w_\ell \Vdash \psi_\ell$, for every $1 \leq \ell \leq k$, then $(w, w_1, \ldots, w_k) \in R^{\mathfrak{M}}$.

It is easy to see that $\mathfrak{M}, w \Vdash \nabla_R(\psi_1, \ldots, \psi_k)$ is equivalent with

$\mathfrak{M}, w \Vdash [\neg R](\neg\psi_1, \ldots, \neg\psi_k)$

and hence Theorem 4 immediately implies the following complexity results.

▶ **Corollary 5.** *The satisfiability problem of* $\mathrm{PML} + \nabla$ *is* ExpTime*-complete.*

Now we will present the reduction. Fix a formula $\varphi \in \mathrm{PML}(\neg)$ and let τ denote the set of accessibility relations occuring in φ. For every symbol $R \in \tau$, we will introduce two fresh symbols of the same arity, R_1 and R_2. Given $\psi \in \mathrm{Subf}(\varphi)$, we let $t(\psi)$ denote the formula obtained from ψ by replacing each $\langle R\rangle$ with $\langle R_1\rangle$ and each $\langle \neg R\rangle$ with $\langle R_2\rangle$. Consider then the following formula $\theta := t(\varphi) \wedge \eta$, where

$$\eta := \bigwedge_{\substack{\langle R_1\rangle(\psi_1,\ldots,\psi_k),\\ \langle R_2\rangle(\chi_1,\ldots,\chi_k)\\ \in \mathrm{Subf}(t(\varphi))}} \Big(\langle \mathrm{E}\rangle(\neg\langle R_1\rangle(\psi_1, \ldots, \psi_k) \wedge \neg\langle R_2\rangle(\chi_1, \ldots, \chi_k))$$

$$\to \bigvee_{1\leq \ell \leq k} \langle \mathrm{A}\rangle(\neg\psi_\ell \vee \neg\chi_\ell)\Big).$$

Intuitively speaking, in every model of η we can extend the interpretations of R_1 and R_2, for $R \in \tau$, in such a way that they cover $W^{ar(R)}$, i.e., every tuple of length $ar(R)$ belongs either to the interpretation of R_1 or to the interpretation of R_2, while maintaining that the resulting model is a model of $t(\varphi)$, if the original model was.

Since the big conjunction in η ranges over only those formulas that occur as subformulas in φ, the size of η is $O(|\varphi|^2)$, i.e., polynomial with respect to $|\varphi|$. The rest of this section is devoted to proving that the above reduction is correct, i.e., φ is satisfiable iff θ is. We will start with the left to right direction.

▶ **Lemma 6.** *If* φ *is satisfiable, then so is* θ*.*

Proof. Let $\mathfrak{M} = (W, (R)_{R\in\tau}, V)$ be a Kripke model and let $w \in W$ be a world so that $\mathfrak{M}, w \Vdash \varphi$. We then define the Kripke model $\mathfrak{N} = (W, (R_1)_{R\in\tau}, (R_2)_{R\in\tau}, V)$ by setting that for every $R \in \tau$, $R_1^{\mathfrak{N}} = R^{\mathfrak{M}}$ and $R_2^{\mathfrak{N}} = W^{ar(R)} \setminus R^{\mathfrak{M}}$. Clearly $\mathfrak{N}, w \Vdash t(\varphi)$. To verify that $\mathfrak{N}$ satisfies η, suppose that there exists $\langle R\rangle(\psi, ..., \psi_k), \langle \neg R\rangle(\chi_1, ..., \chi_k) \in \mathrm{Subf}(t(\varphi))$ and $w_0 \in W$ so that

$\mathfrak{N}, w_0 \Vdash \neg\langle R_1\rangle(\psi_1, \ldots, \psi_k) \wedge \neg\langle R_2\rangle(\chi_1, \ldots, \chi_k)$

but

$\mathfrak{N}, w \nVdash \bigvee_{1\leq \ell \leq k} \langle \mathrm{A}\rangle(\neg\psi_\ell \vee \neg\chi_\ell).$

Thus for every $1 \leq \ell \leq k$ there exists $w_\ell \in W$ so that $\mathfrak{N}, w_\ell \Vdash \psi_\ell \wedge \chi_\ell$. By construction, we must either have that $(w_0, w_1, ..., w_k) \in R_1^{\mathfrak{N}}$ or $(w_0, w_1, ..., w_k) \in R_2^{\mathfrak{N}}$, but clearly both of these cases lead to a contradiction. ◀

Suppose then that θ is satisfiable. $\mathfrak{M}$ be a Kripke model and let $w \in W$ be a world so that $\mathfrak{M}, w \Vdash \theta$.

▶ **Lemma 7.** *For every $(w_0, w_1, \ldots, w_k) \in W^{k+1}$ and $R \in \tau$ there exists $i \in \{1, 2\}$ so that for every $\langle R_i \rangle(\psi_1, \ldots, \psi_k) \in \mathrm{Subf}(t(\varphi))$ we have that if $\mathfrak{M}, w_\ell \Vdash \psi_\ell$, for every $1 \leq \ell \leq k$, then $\mathfrak{M}, w_0 \Vdash \langle R_i \rangle(\psi_1, \ldots, \psi_k)$.*

Proof. Suppose that this is not the case. Thus there exists

$$\langle R_1 \rangle(\psi_1, \ldots, \psi_k), \langle R_2 \rangle(\chi_1, \ldots, \chi_k) \in \mathrm{Subf}(t(\varphi))$$

so that $\mathfrak{M}, w_\ell \Vdash \psi_\ell \wedge \chi_\ell$, for every $1 \leq \ell \leq k$, but

$$\mathfrak{M}, w_0 \Vdash \neg\langle R_1 \rangle(\psi_1, \ldots, \psi_k) \wedge \neg\langle R_2 \rangle(\chi_1, \ldots, \chi_k).$$

Since $\mathfrak{M}, w \Vdash \eta$, we have that

$$\mathfrak{M}, w \Vdash \bigvee_{1 \leq \ell \leq k} \langle \mathrm{A} \rangle(\neg\psi_\ell \vee \neg\chi_\ell),$$

which is a clear contradiction. ◀

Using Lemma 7, we can extend the model $\mathfrak{M}$ as follows. For every $(w_1, \ldots, w_k) \notin R_1^{\mathfrak{M}} \cup R_2^{\mathfrak{M}}$, we choose $i \in \{1, 2\}$ with the properties described in Lemma 7, and add $(w_1, \ldots, w_k)$ to $R_i^{\mathfrak{M}}$. We still use $\mathfrak{M}$ to denote the resulting model. We emphasize that $\mathfrak{M}$ has now the property that for every $(w_1, \ldots, w_k) \in W^k$ and $R \in \tau$, either $(w_1, \ldots, w_k) \in R_1^{\mathfrak{M}}$ or $(w_1, \ldots, w_k) \in R_2^{\mathfrak{M}}$.

▶ **Lemma 8.** $\mathfrak{M}, w \Vdash t(\varphi)$

Proof. A routine induction. ◀

We now define a Kripke model $\mathfrak{N} = (W^*, (R^{\mathfrak{N}})_{R \in \tau}, V^*)$ over τ as follows. First, we specify that $W^* := W \times \{0, 1\}$ and that for every $(w, i) \in W^*$ we have that $(w, i) \in V^*(p)$ iff $w \in V(p)$. Next we need to define interpretations of relation symbols $R \in \tau$. Fix such a relation symbol R. For every $(w_0, i) \in W^*$ we define that if $(w_0, w_1, \ldots, w_k) \in R_1^{\mathfrak{M}}$, then

$$((w_0, i), (w_1, i + 1 \mod 2), \ldots, (w_k, i + 1 \mod 2)) \in R^{\mathfrak{N}}.$$

Then, for every $(w_0, w_1, \ldots, w_k) \in R_2^{\mathfrak{M}}$ we define that

$$((w_0, i), (w_1, i), \ldots, (w_k, i)) \notin R^{\mathfrak{N}}.$$

Finally, for every $((w_0, i_0), \ldots, (w_k, i_k))$ for which we have not specified whether they belong to $R^{\mathfrak{N}}$, we define that if $(w_0, \ldots, w_k) \notin R_2^{\mathfrak{M}}$ then $((w_0, i_0), \ldots, (w_k, i_k)) \in R^{\mathfrak{N}}$.

▶ **Lemma 9.** *For every $\psi \in \mathrm{Subf}(\varphi)$ and $w_0 \in W$ we have that*

$$\mathfrak{N}, (w_0, i_0) \Vdash \psi \Leftrightarrow \mathfrak{M}, w_0 \Vdash t(\psi).$$

Proof. A routine induction. ◀

In particular $\mathfrak{N}$ is a model of φ, since $\mathfrak{N}, (w, 0) \Vdash \varphi$, and hence φ is satisfiable. Thus we can conclude that φ is satisfiable iff θ is.

4 Satisfiability problem of polyadic Boolean modal logic with permutations over bounded vocabularies

Recall that $\mathrm{PML}_c(p, s, \neg, \cap)$ denotes the restriction of $\mathrm{PML}(p, s, \neg, \cap)$ where we consider only c-bounded vocabularies, c being a natural number which should be thought of as a fixed constant. In this section we will give a polynomial time reduction from the satisfiability problem of $\mathrm{PML}_c(p, s, \neg, \cap)$ to that of $\mathrm{PML} + \langle \mathrm{E} \rangle$, which will yield the following result.

▶ **Theorem 10.** *The satisfiability problem of* $\mathrm{PML}_c(p, s, \neg, \cap)$ *is* ExpTime*-complete.*

The reduction used in the proof of Theorem 10 is very similar to the reduction that was used in the previous section to prove Theorem 4. The reader is encouraged to keep this in mind when parsing the reduction, since even though the underlying ideas are again elementary, the resulting reduction is quite technical.

An important property of $\mathrm{PML}(\neg)$ is that it can not speak about intersections of accessibility relations. In the case of $\mathrm{PML}(p, s, \neg, \cap)$ this is obviously no longer the case, but it is still possible to convert each sentence of $\mathrm{PML}(p, s, \neg, \cap)$ into an equi-satisfiable sentence with an analogous property. If the number of underlying accessibility relations and their arities are bounded by some constant, then this translation can also be carried out in polynomial time.

▶ **Lemma 11.** *Let* $\varphi \in \mathrm{PML}_c(p, s, \neg, \cap)[\tau]$ *be a formula. Then we can transform* φ *in polynomial time to a formula* $\varphi^* \in \mathrm{PML}_c(p, s, \neg, \cap)[\tau]$*, which has the following properties.*

1. φ *is satisfiable if and only if* φ^* *is.*
2. *For every* $\langle \mathcal{R} \rangle(\psi_1, \ldots, \psi_k) \in \mathrm{Subf}(\varphi^*)$ *the term* $\mathcal{R}$ *is a* k*-table over* τ*.*

Proof. Note that since τ is c-bounded, the number of tables over τ is bounded by a constant. By applying repeatedly Lemma 1, we can assume that in every formula $\langle \mathcal{R} \rangle(\psi_1, \ldots, \psi_k) \in \mathrm{Subf}(\varphi)$ the term $\mathcal{R}$ is a boolean combination of $(k+1)$-literals over τ. Pick an innermost such subformula of φ. The term $\mathcal{R}$ is clearly equivalent with the term

$$\bigcup_{\rho \models \mathcal{R}} \rho,$$

where each ρ is a $(k+1)$-table over τ. Let $p_{\psi_1}, \ldots, p_{\psi_k}$ denote fresh propositional symbols. In φ we replace $\langle R \rangle(\psi_1, \ldots, \psi_k)$ with the following formula

$$\bigvee_{\rho \models \mathcal{R}} \langle \rho \rangle(p_{\psi_1}, \ldots, p_{\psi_k}).$$

Note the above formula is essentially of constant size, since the number of tables over τ is bounded by a constant. Now, let φ' denote the resulting formula. Without loss of generality we will assume that τ contains at least one binary relation symbol. With this technical assumption it is clear that φ is equisatisfiable with the formula

$$\varphi' \wedge \bigwedge_{1 \leq \ell \leq k} \bigwedge_{\text{2-table } \rho} [\rho](p_{\psi_\ell} \leftrightarrow \psi_\ell).$$

Since the size of τ is bounded by a constant, the number of 2-tables over τ is also bounded by a constant. Thus the size of the above formula is polynomial with respect to the size of the original formula φ. It should be clear that by repeating the above procedure sufficiently many times, we will eventually reach the desired formula. ◀

▶ Remark 12. In Lemma 11 it is not enough to assume that there is a constant bound on the *arities* of relations in τ to guarantee that the translation is efficient, i.e., polynomial time computable, since already in the case of binary vocabularies there are exponentially many tables. Likewise, it is not enough to assume that only the size of τ is bounded by a constant, since the number of tables over a vocabulary which consists of a single relation R of arity n is bounded from below by 2^n.

For the rest of this section we will assume that φ satisfies property (ii) of Lemma 11. Now, for $2 \leq k \leq m$, where $m = \max\{ar(R) \mid R \in \tau\}$, let $\mathfrak{T}_k$ denote the set of all tables over τ. Each k-table $\rho \in \mathfrak{F}_k$ can be seen as a k-ary accessibility relation and hence we will consider the vocabulary $\tau_{\mathfrak{T}} := \bigcup_{2 \leq k \leq m} \mathfrak{T}_k$. We let $t(\varphi)$ denote the sentence in $\mathrm{PML}_c[\tau_{\mathfrak{T}}]$ which is obtained from φ by replacing each table $\rho \in \mathrm{GRA}(p, s, \neg, \cap)[\tau]$ with the corresponding relation symbol $\rho \in \tau_{\mathfrak{T}}$.

We next describe sentences of $\mathrm{PML}[\tau_{\mathfrak{T}}] + \langle \mathrm{E} \rangle$ which together play the same role that η did in the previous section. We start with the following sentence, which we denoted by ξ_1.

$$\xi_1 := \langle \mathrm{A} \rangle \bigwedge_{1 \leq k < m} \bigwedge_{\substack{g : \mathfrak{T}_{k+1} \to S_{k+1} \\ \sigma_\rho := g(\rho)}} \bigwedge_{\substack{h : \mathfrak{T}_{k+1} \to (\mathrm{Subf}(t(\varphi)))^k \\ h(\rho) = (\psi_1^{\sigma_\rho}, \ldots, \psi_k^{\sigma_\rho})}} \Bigg(\Big(\bigwedge_{\rho \in X_0} \neg \langle \sigma_\rho[\rho] \rangle (\psi_1^{\sigma_\rho}, \ldots, \psi_k^{\sigma_\rho})$$

$$\wedge \bigwedge_{1 \leq \ell \leq k} \langle \mathrm{E} \rangle \Big(\bigwedge_{\rho \in X_\ell} \neg \langle \sigma_\rho[\rho] \rangle (\psi_1^{\sigma_\rho}, \ldots, \psi_k^{\sigma_\rho}) \wedge \bigwedge_{\rho \notin X_\ell} \psi^{\sigma_\rho}_{\sigma_\rho^{-1}(\ell)} \Big) \Big) \to \bigvee_{\rho \notin X_0} \neg \psi^{\sigma_\rho}_{\sigma_\rho^{-1}(0)} \Bigg)$$

In the above sentence we use X_ℓ to denote the set $\{\rho \mid \sigma_\rho(0) = \ell\}$. Note that since τ is c-bounded, ξ_1 is only of size at most polynomial with respect to the size of φ.

In addition to ξ_1, we will need the following sentences, which will be denoted by ξ_2 and ξ_3 respectively.

$$\xi_2 := \langle \mathrm{A} \rangle \bigwedge_{1 \leq k < m} \bigwedge_{\rho \in \mathfrak{T}_{k+1}} \bigwedge_{\substack{\sigma \in S_{k+1} \\ \sigma(0) \neq 0}} \bigwedge_{\psi_1, \ldots, \psi_k \in \mathrm{Subf}(t(\varphi))}$$

$$\Big(\neg \psi_{\sigma^{-1}(0)} \vee \neg \langle \rho \rangle (\neg \psi_{\sigma^{-1}(1)}, \ldots, \underbrace{\langle \sigma[\rho] \rangle (\psi_1, \ldots, \psi_k)}_{\sigma(0)\text{:th formula}}, \ldots, \neg \psi_{\sigma^{-1}(k)}) \Big)$$

$$\xi_3 := \langle \mathrm{A} \rangle \bigwedge_{1 \leq k < m} \bigwedge_{\rho \in \mathfrak{T}_{k+1}} \bigwedge_{\substack{\sigma \in S_{k+1} \\ \sigma(0) = 0}} \bigwedge_{\psi_1, \ldots, \psi_k \in \mathrm{Subf}(t(\varphi))}$$

$$\Big(\langle \rho \rangle (\psi_{\sigma^{-1}(1)}, \ldots, \psi_{\sigma^{-1}(k)}) \to \langle \sigma[\rho] \rangle (\psi_1, \ldots, \psi_k) \Big)$$

Again, since τ is c-bounded, ξ_1 and ξ_2 are only of size at most polynomial with respect to the size of φ. We let $\Theta := t(\varphi) \wedge \xi_2 \wedge \xi_2 \wedge \xi_3$.

The sentences ξ_1, ξ_2, ξ_3 might look rather complicated, but we emphasize again that they are simply playing essentially the same role that η did in the previous section. Namely, they axiomatize enough properties of tables so that in any model of Θ we can enlarge the interpretations of the accessibility relations ρ in such a way that they cover all the tuples of relevant length, while maintaining that the resulting model is still a model of $t(\varphi)$.

The rest of this section is devoted towards proving that φ is satisfiable iff Θ. We start with the easy direction.

▶ **Lemma 13.** *If φ is satisfiable, then so is Θ.*

Proof. Let $\mathfrak{M} = (W, (R^{\mathfrak{M}})_{R\in\tau}, V)$ be a Kripke model and let $w \in W$ be a world so that $\mathfrak{M}, w \Vdash \varphi$. We then define the Kripke model $\mathfrak{N} = (W, (\rho^{\mathfrak{N}})_{\rho\in\tau_{\mathfrak{T}}}, V)$ by setting that for every $\rho \in \tau_{\mathfrak{T}}$, $\rho^{\mathfrak{N}} = [\![\rho]\!]_{\mathfrak{M}}$. Clearly $\mathfrak{N}, w \Vdash t(\varphi)$.

Let us then verify that $\mathfrak{N}, w_0 \Vdash \xi_1$, for any $w_0 \in W$. Fix k, g and h. Suppose that

$$\mathfrak{N}, w_0 \Vdash \bigwedge_{\rho\in X_0} \neg\langle\sigma_\rho[\rho]\rangle(\psi_1^{\sigma_\rho}, \ldots, \psi_k^{\sigma_\rho})$$

and that for every $1 \leq \ell \leq k$ there exists w_ℓ so that

$$\mathfrak{N}, w_\ell \Vdash \bigwedge_{\rho\in X_\ell} \neg\langle\sigma_\rho[\rho]\rangle(\psi_1^{\sigma_\rho}, \ldots, \psi_k^{\sigma_\rho}) \wedge \bigwedge_{\rho\notin X_\ell} \psi^{\sigma_\rho}_{\sigma_\rho^{-1}(\ell)}.$$

Our goal is to show that

$$\mathfrak{N}, w_0 \Vdash \bigvee_{\rho\notin X_0} \neg\psi^{\sigma_\rho}_{\sigma_\rho^{-1}(0)}.$$

Aiming for a contradiction, suppose that this is not the case. Since every tuple realizes a table in $\mathfrak{M}$, there exists $\rho \in \tau_{\mathfrak{T}}$ so that $(w_0, w_1, \ldots, w_k) \in \rho^{\mathfrak{N}}$. Recall that the function g associates a permutation σ_ρ with ρ. Now $(w_{\sigma_\rho(0)}, \ldots, w_{\sigma_\rho(k)}) \in [\![\sigma_\rho\rho]\!]_{\mathfrak{M}}$. Let $\ell_0 := \sigma_\rho(0)$. Then, for every $\ell \neq \ell_0$ we have that

$$\mathfrak{N}, w_\ell \Vdash \psi^{\sigma_\rho}_{\sigma_\rho^{-1}(\ell)},$$

since $\rho \notin X_\ell$. This implies that

$$\mathfrak{N}, w_{\sigma_\rho(\ell)} \Vdash \psi_\ell^{\sigma_\rho},$$

for every $1 \leq \ell \leq k$. On the other hand

$$\mathfrak{N}, w_{\sigma_\rho(0)} \Vdash \neg\langle\sigma_\rho[\rho]\rangle(\psi_1^{\sigma_\rho}, \ldots, \psi_k^{\sigma_\rho}),$$

which is a contradiction, since $\sigma_\rho[\rho]^{\mathfrak{N}} = [\![\sigma_\rho\rho]\!]_{\mathfrak{M}}$.

Next, we will verify that $\mathfrak{N}, w_0 \Vdash \xi_2$, for any $w_0 \in W$. Fix k, σ and $\psi_1, \ldots, \psi_k \in \mathrm{Subf}(t(\varphi))$. Aiming for a contradiction, suppose that

$$\mathfrak{N}, w_0 \Vdash \Big(\psi_{\sigma^{-1}(0)} \wedge \langle\rho\rangle(\psi_{\sigma^{-1}(1)}, \ldots, \neg\langle\sigma[\rho](\psi_1, \ldots, \psi_k), \ldots, \psi_{\sigma^{-1}(k)})\Big)$$

Thus there exists $(w_0, w_1, \ldots, w_k) \in \rho^{\mathfrak{N}}$ so that $\mathfrak{M}, w_\ell \Vdash \psi_{\sigma^{-1}(\ell)}$, for every $\ell \neq \sigma(0)$ (including $\ell = 0$), and

$$\mathfrak{M}, w_{\sigma(0)} \Vdash \neg\langle\sigma[\rho]\rangle(\psi_1, \ldots, \psi_k).$$

Now $(w_{\sigma(0)}, \ldots, w_{\sigma(k)}) \in (\sigma[\rho])^{\mathfrak{M}}$ and furthermore $\mathfrak{M}, w_{\sigma(\ell)} \Vdash \psi_\ell$, for every $1 \leq \ell \leq k$, which is a contradiction.

Finally, we will verify that $\mathfrak{N}, w_0 \Vdash \xi_3$, for any $w_0 \in W$. Fix k, σ and $\psi_1, \ldots, \psi_k \in \mathrm{Subf}(t(\varphi))$. Aiming for a contradiction, suppose that

$$\mathfrak{M}, w_0 \Vdash \langle\rho\rangle(\psi_{\sigma^{-1}(1)}, \ldots, \psi_{\sigma^{-1}(k)}),$$

but

$$\mathfrak{M}, w_0 \Vdash \neg\langle\sigma[\rho]\rangle(\psi_1, \ldots, \psi_k).$$

Now there exists $(w_0, w_1, \ldots, w_k) \in \rho^{\mathfrak{N}}$ so that $\mathfrak{M}, w_\ell \Vdash \psi_{\sigma^{-1}(\ell)}$. Hence $(w_0, w_{\sigma(1)}, \ldots, w_{\sigma(k)}) \in [\![\sigma\rho]\!]_{\mathfrak{M}} = (\sigma[\rho])^{\mathfrak{N}}$, which is a contradiction, since $\mathfrak{M}, w_{\sigma(\ell)} \Vdash \psi_\ell$, for every $1 \leq \ell \leq k$. ◀

Suppose now that $\mathfrak{M}$ is a Kripke model and that there is a $w \in W$ so that $\mathfrak{M}, w \Vdash \Theta$. To obtain a model for φ, we start by extending $\mathfrak{M}$ as follows.

1. For every $2 \leq k \leq m, \rho \in \mathfrak{T}_k, \sigma \in S_k$ and $(w_1, \dots, w_k) \in \rho^{\mathfrak{M}}$, we will add $(w_{\sigma(1)}, \dots, w_{\sigma(k)})$ to $(\sigma[\rho])^{\mathfrak{M}}$.
2. For every $2 \leq k \leq m$ and $(w_1, \dots, w_k) \in W^k$, for which there does not exists $\rho \in \mathfrak{T}_k$ so that $(w_1, \dots, w_k) \in \rho^{\mathfrak{M}}$, we let ρ denote the relation promised by Lemma 14 (see below) and we will add the tuple $(w_{\sigma(1)}, \dots, w_{\sigma(k)})$ to $(\sigma[\rho])^{\mathfrak{M}}$, for every $\sigma \in S_k$.

Letting $\mathfrak{M}^*$ denote the resulting model, we need to show that $\mathfrak{M}^*, w \Vdash t(\varphi)$. We start with the following lemma which guarantees that we can extend the interpretations of k-ary accessibility relations in $\mathfrak{M}$ in such a way that every tuple will belong to the interpretation of at least one such relation.

▶ **Lemma 14.** *For every $1 \leq k < m$ and $(w_0, w_1, \dots, w_k) \in W^{k+1}$ there exists $\rho \in \mathfrak{T}_{k+1}$ so that for all $\sigma \in S_{k+1}$ and $\psi_1, \dots, \psi_k \in \mathrm{Subf}(t(\varphi))$ we have that if $\mathfrak{M}, w_{\sigma(\ell)} \Vdash \psi_\ell$, for every $1 \leq \ell \leq k$, then $\mathfrak{M}, w_{\sigma(0)} \Vdash \langle \sigma[\rho] \rangle (\psi_1, \dots, \psi_k)$.*

Proof. Fix $(w_0, w_1, \dots, w_k) \in W^{k+1}$. Aiming for a contradiction, suppose that for every $\rho \in \mathfrak{T}_{k+1}$ there exists $\sigma_\rho \in S_{k+1}$ and $\psi_1^{\sigma_\rho}, \dots, \psi_k^{\sigma_\rho} \in \mathrm{Subf}(t(\varphi))$ so that

$$\mathfrak{M}, w_{\sigma_\rho(\ell)} \Vdash \psi_\ell^{\sigma_\rho},$$

for every $1 \leq \ell \leq k$, but

$$\mathfrak{M}, w_{\sigma_\rho(0)} \Vdash \neg \langle \sigma_\rho[\rho] \rangle (\psi_1^{\sigma_\rho}, \dots, \psi_k^{\sigma_\rho}).$$

It is simple to verify that this entails that

$$\mathfrak{M}, w_0 \Vdash \bigwedge_{\rho \in X_0} \neg \langle \sigma_\rho[\rho] \rangle (\psi_1^{\sigma_\rho}, \dots, \psi_k^{\sigma_\rho})$$

$$\wedge \bigwedge_{1 \leq \ell \leq k} \langle \mathrm{E} \rangle \Big(\bigwedge_{\rho \in X_\ell} \neg \langle \sigma_\rho[\rho] \rangle (\psi_1^{\sigma_\rho}, \dots, \psi_k^{\sigma_\rho}) \wedge \bigwedge_{\rho \notin X_\ell} \psi^{\sigma_\rho}_{\sigma_\rho^{-1}(\ell)} \Big)$$

and since $\mathfrak{M}, w_0 \Vdash \xi_1$, we have that

$$\mathfrak{M}, w_0 \Vdash \bigvee_{\rho \notin X_0} \neg \psi^{\sigma_\rho}_{\sigma_\rho^{-1}(0)},$$

which is a contradiction, since by assumption $\mathfrak{M}, w_0 \Vdash \bigwedge_{\rho \notin X_0} \psi^{\sigma_\rho}_{\sigma_\rho^{-1}(0)}$. ◀

Secondly, we will need a lemma which guarantees that we can close the interpretations of accessibility relations under permutations.

▶ **Lemma 15.** *For every $1 \leq k < m$, $\rho \in \mathfrak{T}_{k+1}$, $\sigma \in S_{k+1}$ we have that if $(w_0, w_1, \dots, w_k) \in \rho^{\mathfrak{M}}$, then for every $\psi_1, \dots, \psi_k \in \mathrm{Subf}(t(\varphi))$ we have that if $\mathfrak{M}, w_{\sigma(\ell)} \Vdash \psi_\ell$, for every $1 \leq \ell \leq k$, then $\mathfrak{M}, w_{\sigma(0)} \Vdash \langle \sigma[\rho] \rangle (\psi_1, \dots, \psi_k)$.*

Proof. Fix k, ρ, σ and $(w_0, w_1, \dots, w_k) \in \rho^{\mathfrak{M}}$. Aiming for a contradiction, suppose that there exists $\psi_1, \dots, \psi_k \in \mathrm{Subf}(t(\varphi))$ so that

$$\mathfrak{M}, w_{\sigma(\ell)} \Vdash \psi_\ell,$$

for every $1 \leq \ell \leq k$, but

$$\mathfrak{M}, w_{\sigma(0)} \Vdash \neg \langle \sigma[\rho] \rangle (\psi_1, \dots, \psi_k).$$

We have now two cases based on whether or not $\sigma(0) = 0$. First, if $\sigma(0) = 0$, then since $\mathfrak{M}, w_0 \Vdash \xi_3$, we have that

$$\mathfrak{M}, w_0 \Vdash \langle\rho\rangle(\psi_{\sigma^{-1}(1)}, \ldots, \psi_{\sigma^{-1}(k)}) \to \langle\sigma[\rho]\rangle(\psi_1, \ldots, \psi_k).$$

By assumption, $(w_0, w_1, \ldots, w_k) \in \rho^{\mathfrak{M}}$ and $\mathfrak{M}, w_{\sigma(\sigma^{-1}(\ell))} \Vdash \psi_{\sigma^{-1}(\ell)}$, for every $1 \leq \ell \leq k$, and hence

$$\mathfrak{M}, w_0 \Vdash \langle\rho\rangle(\psi_{\sigma^{-1}(1)}, \ldots, \psi_{\sigma^{-1}(k)}),$$

which implies that

$$\mathfrak{M}, w_0 \Vdash \langle\sigma[\rho]\rangle(\psi_1, \ldots, \psi_k),$$

a contradiction.

Consider then the case $\sigma(0) \neq 0$. Since $\mathfrak{M}, w_0 \Vdash \xi_2$, we have that

$$\mathfrak{M}, w_0 \Vdash \left(\neg\psi_{\sigma^{-1}(0)} \vee \neg\langle\rho\rangle(\neg\psi_{\sigma^{-1}(1)}, \ldots, \underbrace{\langle\sigma[\rho]\rangle(\psi_1, \ldots, \psi_k)}_{\sigma(0)\text{:th formula}}, \ldots, \neg\psi_{\sigma^{-1}(k)})\right)$$

By assumption $\mathfrak{M}, w_{\sigma(\sigma^{-1}(0))} \Vdash \psi_{\sigma^{-1}(0)}$. Furthermore, since $(w_0, w_1, \ldots, w_k) \in \rho^{\mathfrak{M}}$ and $\mathfrak{M}, w_{\sigma(\sigma^{-1}(\ell))} \Vdash \psi_{\sigma^{-1}(\ell)}$, for every $\ell \neq \sigma(0)$, we must have that

$$\mathfrak{M}, w_{\sigma(0)} \Vdash \langle\sigma[\rho]\rangle(\psi_1, \ldots, \psi_k),$$

which is a clear contradiction. ◀

A routine induction, which uses the previous two lemmas in the case of formulas of the form $\langle\rho\rangle(\psi_1, \ldots, \psi_k)$, can be used to establish that $\mathfrak{M}^*, w \Vdash t(\varphi)$. For the rest of this section we will use $\mathfrak{M}$ to denote $\mathfrak{M}^*$.

Now we are ready to use $\mathfrak{M}$ to construct a model for φ. We define a Kripke model $\mathfrak{N} = (W^*, (R^{\mathfrak{N}})_{R\in\tau}, V^*)$ over τ as follows. First, we define that

$$W^* := W \times \{2, \ldots, m\} \times \mathbb{N},$$

where $\mathbb{N}$ is the set of natural numbers. In what follows we will adopt the convention that we will associate to every k-table ρ an unique index from the set $\mathbb{N}$, which we simply denote by ρ. We start our model construction by specifying that for every $(w, \ell, r) \in W^*$ we have that $(w, \ell, r) \in V^*(p)$ iff $w \in V(p)$. Next we will assign tables to tuples. We first define that for every $(w_0, \ell, r) \in W^*$ and $(w_0, w_1, \ldots, w_k) \in \rho^{\mathfrak{M}}$ the tuple

$$((w_0, \ell, r), (w_1, 2, r + \rho), \ldots, (w_k, k + 1, r + \rho))$$

realizes the type ρ. Observe that each such tuple consists of $k + 1$ distinct elements. Indeed, the first element is distinct from the remaining elements because $r \neq r + \rho$, while the remaining elements in the tuple are pairwise distinct because they differ with respect to their second coordinate.

Notice that if we force a tuple to realize a table ρ, then for every $\sigma \in S_{k+1}$ the permutation of this tuple under σ realizes the type $\sigma[\rho]$. Hence, it is not obvious that the above procedure does not assign different tables to some tuples.

▷ Claim 16. In the above procedure, no tuple is assigned a table more than once.

Proof. Suppose that we have assigned tables ρ_1 and ρ_2 to a tuple $(w_0, w_1, \ldots, w_k)$. We want to show that $\rho_1 = \rho_2$. By construction, our assumption implies that there are tuples

$$((w'_0, \ell', r'), (w'_1, 2, r' + \rho'), \ldots, (w'_k, k+1, r' + \rho'))$$

and

$$((w''_0, \ell'', r''), (w''_1, 2, r'' + \rho''), \ldots, (w''_k, k+1, r'' + \rho''))$$

and permutations σ_1 and σ_2 so that σ_1 (respectively σ_2) applied to the first (respectively the second) tuple gives $(w_0, w_1, \ldots, w_k)$, and furthermore that $\sigma_1[\rho'] = \rho_1$ and $\sigma_2[\rho''] = \rho_2$. Since for every $2 \leq \ell \leq k+1$ there exists an unique element in the tuple $(w_0, w_1, \ldots, w_k)$ which has ℓ as its second coordinate, we must have that the two above tuples are in fact the same tuples, since they are both permutations of $(w_0, w_1, \ldots, w_k)$. In particular, $\rho' = \rho''$, since $r' = r''$. Finally, because this single tuple consists of $k+1$ distinct elements and permutating it with either σ_1 or σ_2 gives the same result – namely $(w_0, w_1, \ldots, w_k)$ – we must have that $\sigma_1 = \sigma_2$, and hence $\rho_1 = \rho_2$. ◁

Finally, for every tuple $((w_0, \ell_0, r_0), \ldots, (w_k, \ell_k, r_k)$ for which we have not yet assigned a table, we will pick some $\rho \in \mathfrak{T}_{k+1}$ for which $(w_0, \ldots, w_k) \in \rho^{\mathfrak{M}}$ and assign the corresponding table to our tuple. This completes the definition of $\mathfrak{N}$.

▶ **Lemma 17.** *For every $\psi \in \mathrm{Subf}(\varphi)$ and $w_0 \in W$ we have that*

$$\mathfrak{N}, (w_0, \ell, r) \Vdash \psi \Leftrightarrow \mathfrak{M}, w_0 \Vdash t(\psi).$$

Proof. A routine induction. ◀

In particular $\mathfrak{N}$ is a model of φ and hence φ is satisfiable. Thus we can conclude that φ is satisfiable iff Θ is.

5 Model checking problem of polyadic Boolean modal logic with permutations

In this section we prove that the combined complexity of $\mathrm{PML}(p, s, \neg, \cap)$ is PTIME-complete. Note that the corresponding lower bound follows already from the fact that the combined complexity of standard modal logic is PTIME-complete [10]. Throughout this section we will assume that the domains of the models are equipped with some (arbitrary) linear order.

We start by defining precisely how we will encode Kripke models. In fact, we will describe how we will encode arbitrary relational models, since it avoids some notational clutter. Given two strings x and y, we will use $x\#y$ to denote their concatenation. The *database encoding* of $\mathfrak{A}$ is the sequence

$$1^{|A|} \rhd \mathrm{lenc}(R_1) \rhd \cdots \rhd \mathrm{lenc}(R_m)$$

where $\rhd$ is a separator character (the use of which could be of course avoided) and each $\mathrm{lenc}(R_i)$ is a sequence

$$r_1 \# r_2 \# \ldots \# r_{|R_i|},$$

where each r_j is a sequence consisting of $ar(R_i)$-many binary strings of length $\log_2(|A|)$ which encodes the jth tuple in $R_i^{\mathfrak{A}}$. We note that the length of the database encoding of a model $\mathfrak{A}$, denoted by $||\mathfrak{A}||$, is

$$O\left(|A| + \sum_{1 \leq i \leq m} |R_i| ar(R_i) \log_2(|A|)\right).$$

▶ Remark 18. The encoding of models that we have presented here is not the only encoding of relational structures one encounters in the literature. Another standard choice of encoding can be found in [22], where the encoding essentially describes the "adjacency" matrix of each relation, i.e., for every relation R and every tuple of length $ar(R)$ there is a single bit which indicates whether that tuple belongs to the interpretation of R. Note that if the arities of relation symbols are bounded by a constant, then this encoding of models, which we call the *matrix encoding*, is essentially equivalent with the database encoding of models.

If matrix encoding of models is used, then the PTime upper bound on the model checking problem of $\mathrm{PML}(p, s, \neg, \cap)$ becomes somewhat trivial. Indeed, one can then compute complements of relations in linear time, which allows one to easily reduce the model checking problem of $\mathrm{PML}(p, s, \neg, \cap)$ to the model checking problem of, say, the guarded fragment, which is known to be PTime-complete [3].

Next we will present our model checking algorithm. As an important preliminary step, the following lemma will allow us to restrict our attention to formulas that contain only terms which use negation in a very restricted way.

▶ **Lemma 19.** *Suppose that $\mathcal{R} \in \mathrm{GRA}(p, s, \neg, \cap)[\tau]$. We can compute in polynomial time a term $\mathcal{R}' \in \mathrm{GRA}(p, s, \neg, \setminus, \cap, \cup)$ so that $\mathcal{R}$ is equivalent with $\mathcal{R}'$ and $\mathcal{R}'$ is either of the form $\mathcal{R}''$ or of the form $\neg\mathcal{R}''$, for some $\mathcal{R}'' \in \mathrm{GRA}(p, s, \setminus, \cap, \cup)[\tau]$.*

Proof. Using Lemma 1, we can bring all the negations occurring in the input term $\mathcal{R}$ to the start of the term, which – after eliminating consecutive negations – results in a term which is either of the form $\mathcal{R}'$ or $\neg\mathcal{R}'$, where $\mathcal{R}' \in \mathrm{GRA}(p, s, \setminus, \cap, \cup)[\tau]$. ◀

Suppose now that $\varphi \in \mathrm{PML}(p, s, \neg, \cap)[\tau]$ and $\mathfrak{M} = (W, (R^{\mathfrak{M}})_{R\in\tau}, V)$. Our goal is to compute the set of worlds in $\mathfrak{M}$ where φ is true. By applying Lemma 19, we can assume that in each subformula $\langle\mathcal{R}\rangle(\psi_1, \ldots, \psi_k)$ the term $\mathcal{R}$ is either of the form $\mathcal{R}'$ or $\neg\mathcal{R}'$, where $\mathcal{R}' \in \mathrm{GRA}(p, s, \setminus, \cap, \cup)[\tau]$. Using induction, one can show that the size of $[\![\mathcal{R}']\!]_{\mathfrak{M}}$ is only polynomial with respect to $||\mathfrak{M}||$.

Now we will describe the model checking algorithm, which extends the standard labeling algorithm that is often used in the context of modal logics [30]. The algorithm will use some enumeration of the subformulas $\varphi_1, \ldots, \varphi_n$ of φ which satisfies the requirement that if φ_j is a proper subformula of φ_i, then $j < i$.

$\nu = \varnothing$
for $i = 1$ **through** n **do**:
  if $\varphi_i = p$ **then** $\nu := \nu \cup \{(p, V(p))\}$
  if $\varphi_i = \neg\varphi_j$ **then** $\nu := \nu \cup \{(\varphi_i, W \setminus \nu(\varphi_j))\}$
  if $\varphi_i = \varphi_j \wedge \varphi_k$ **then** $\nu := \nu \cup \{(\varphi_i, \nu(\varphi_j) \cap \nu(\varphi_k))\}$
  if $\varphi_i = \langle\mathcal{R}\rangle(\varphi_{i_1}, \ldots, \varphi_{i_k})$ **then**

$$\nu := \nu \cup \left(\varphi_i, \left\{ w \in W \,\middle|\, \left(w \times \nu(\varphi_{i_1}) \times \cdots \times \nu(\varphi_{i_k})\right) \cap [\![\mathcal{R}]\!]_{\mathfrak{M}} \neq \varnothing \right\}\right)$$

  if $\varphi_i = \langle\neg\mathcal{R}\rangle(\varphi_{i_1}, \ldots, \varphi_{i_k})$ **then**
    $U := \varnothing$
    for $w \in W$ **do**:
      for $(w_1, \ldots, w_k) \in \nu(\varphi_{i_1}) \times \cdots \times \nu(\varphi_{i_k})$ **do**:
        if $(w, w_1, \ldots, w_k) \notin [\![\mathcal{R}]\!]_{\mathfrak{M}}$ **then** $U := U \cup \{w\}$ and **break**
    $\nu := \nu \cup \{(\varphi_i, U)\}$
return $\nu(\varphi)$

It is straightforward to check that the above algorithm is correct. We note that in certain places the description of the algorithm is, for ease of exposition, somewhat informal. In particular, we have not specified how in the case of $\langle \neg \mathcal{R} \rangle(\varphi_{i_1}, \ldots, \varphi_{i_k})$ the for-loop going through the set $\nu(\varphi_{i_1}) \times \ldots \nu(\varphi_{i_k})$ should be implemented, which is in fact a somewhat important detail, since we can not always construct this set explicitly as in the worst case its size is proportional to $|W|^k$ (which is exponential in the size of the input, since k is not bounded by a constant). However, this explicit construction can be avoided by maintaining k pointers to elements of the sets $\nu(\varphi_{i_\ell})$. More precisely, we can initialize k pointers which at the beginning point at the first element of A and which we will then increment in a lexicographical manner.

We are now left with the easy task of proving that our algorithm runs in polynomial time.

▶ **Lemma 20.** *The above algorithm runs in time polynomial with respect to*

$$|\varphi| \times ||\mathfrak{M}||.$$

Proof. The outermost for-loop is executed $|\mathrm{Subf}(\varphi)| \leq |\varphi|$ times, so it suffices to argue that each case within the loop can be done in time polynomial with respect to $|\varphi| \times ||\mathfrak{M}||$. The most non-trivial case is the case of $\langle \neg \mathcal{R} \rangle(\varphi_{i_1}, \ldots, \varphi_{i_k})$, where we can make the simple observation that the running time of the for-loop going through $\nu(\varphi_{i_1}) \times \ldots \nu(\varphi_{i_k})$ is bounded above by the size of $[\![\mathcal{R}]\!]_{\mathfrak{M}} \leq ||\mathfrak{M}||$, since it stops after we have encountered a tuple which does not belong to $[\![\mathcal{R}]\!]_{\mathfrak{M}}$ (or after we have went through all the relevant k-tuples). ◀

Since the model checking problem of standard modal logic is PTime-complete, we have the desired result.

▶ **Theorem 21.** *The model checking problem of* $\mathrm{PML}(p, s, \neg, \cap)$ *is PTime-complete.*

6 Conclusions

We have studied the computational complexity of model checking and satisfiability problems of polyadic modal logics extended with permutations and Boolean operators on accessibility relations. Concerning satisfiability problems, we have proved that the satisfiability problems of both polyadic modal logic extended with negations of accessibility relations $\mathrm{PML}(\neg)$ and full polyadic Boolean modal logic extended with permutations over $\mathrm{PML}(p, s, \neg, \cap)$ are ExpTime-complete, the latter under the assumption that the underlying set of accessibility relations and their arities are bounded by a constant, which is necessary if ExpTime $\neq$ NExpTime, since the satisfiability problem of $\mathrm{PML}(p, s, \neg, \cap)$ is in general NExpTime-complete. We have also established that the model checking problem for full polyadic Boolean modal logic extended with permutations $\mathrm{PML}(p, s, \neg, \cap)$ is PTime-complete. Our results contribute to the research program outlined in [14] and extend the results of [24, 27] to polyadic context.

Concerning future research directions, the reductions that we used in establishing complexity bounds on satisfiability problems seem to be quite robust, and hence we expect that in the future they can be used to extend the results presented here. For instance, one can most likely show that $\mathrm{PML}(p, s, \neg)$ has an ExpTime-complete satisfiability problem, which we have not yet been able to do. In this direction a natural intermediate problem would be to establish that $\mathrm{PML}(p, s) + \langle \mathrm{E} \rangle$ has an ExpTime-complete satisfiability problem, since then one might be able to adapt the techniques used in this paper to reduce the satisfiability problem of $\mathrm{PML}(p, s, \neg)$ to that of $\mathrm{PML}(p, s) + \langle \mathrm{E} \rangle$. Following [14], we are also quite confident that $\mathrm{PML}(p, s, \neg)$ could be extended with counting without affecting its ExpTime-completeness.

Another obvious direction would be to show that if the number of accessibility relations and their arities are bounded by a constant, then the satisfiability problem of $\mathrm{PML}(p, s, \neg, \cap)$ extended with an equality operator I is ExpTime-complete. Indeed, such a result would fully generalize the main result of [27] to polyadic context, which states the satisfiability problem of $\mathrm{ML}(I, s, \neg, \cap)$ is ExpTime-complete, when the number of accessibility relations is bounded by a constant. Here the main technical difficulty is that it seems that one needs to reduce the satisfiability problem of $\mathrm{PML}(I, p, s, \neg, \cap)$ to that of PML extended with the *difference modality* $\langle d \rangle$ [6] (as opposed to just reducing it to $\mathrm{PML} + \langle \mathrm{E} \rangle$). $\langle d \rangle \psi$ states that there exists a world which is *different* from the current world and in which ψ is true. Roughly speaking, the need for $\langle d \rangle$ arises from the fact that one needs to encode basic properties of equality I in the reduction.

References

1 Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

2 Bartosz Bednarczyk. Exploiting forwardness: Satisfiability and query-entailment in forward guarded fragment. In *Logics in Artificial Intelligence*, pages 179–193. Springer, 2021.

3 Dietmar Berwanger and Erich Grädel. Games and model checking for guarded logics. In *Proceedings of the Artificial Intelligence on Logic for Programming*, pages 70–84. Springer-Verlag, 2001.

4 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001. doi:10.1017/CBO9781107050884.

5 Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In Alberto O. Mendelzon and Jan Paredaens, editors, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 149–158, 1998. doi:10.1145/275487.275504.

6 Maarten de Rijke. The modal logic of inequality. *Journal of Symbolic Logic*, 57(2):566–584, 1992. doi:10.2307/2275293.

7 G Gargov, S Passy, and T Tinchev. Modal environment for boolean speculations. In *Mathematical logic and its applications*, pages 253–263, 1987.

8 Erich Grädel. On the restraining power of guards. *J. Symb. Log.*, 64(4):1719–1742, 1999. doi:10.2307/2586808.

9 Erich Grädel. *Why Are Modal Logics so Robustly Decidable?*, pages 393–408. World Scientific Publishing Co., Inc., 2001.

10 Erich Grädel. Finite model theory and descriptive complexity. In *Finite Model Theory and Its Applications*, pages 125–230. Springer, 2007.

11 Erich Grädel, Phokion Kolaitis, and Moshe Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

12 Lauri Hella and Antti Kuusisto. One-dimensional fragment of first-order logic. In *Proceedings of Advances in Modal Logic*, pages 274–293, 2014.

13 Jonne Iso-Tuisku and Antti Kuusisto. Uniform one-dimensional fragment over ordered structures. *arXiv*, abs/1812.08732, 2018. arXiv:1812.08732.

14 Jonne Iso-Tuisku and Antti Kuusisto. Description logics as polyadic modal logics. *arXiv*, abs/2108.08838, 2021. arXiv:2108.08838.

15 Reijo Jaakkola. Ordered fragments of first-order logic. In *46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 62:1–62:14, 2021.

16 Reijo Jaakkola and Antti Kuusisto. Algebraic classifications for fragments of first-order logic and beyond. *arXiv*, abs/2005.01184, 2020. arXiv:2005.01184.

17 Emanuel Kieronski. One-dimensional guarded fragments. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS*, pages 16:1–16:14, 2019. doi:10.4230/LIPIcs.MFCS.2019.16.

18 Emanuel Kieronski and Antti Kuusisto. Complexity and expressivity of uniform one-dimensional fragment with equality. In *39nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 365–376, 2014.

19 Emanuel Kieronski and Antti Kuusisto. One-dimensional fragment over words and trees. *Journal of Logic and Computation*, February 2022.

20 Antti Kuusisto. On the uniform one-dimensional fragment. In *International Workshop on Description Logics (DL)*, 2016.

21 Antti Kuusisto and Carsten Lutz. Weighted model counting beyond two-variable logic. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 619–628, 2018.

22 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

23 Carsten Lutz and Ulrike Sattler. The complexity of reasoning with boolean modal logics. In *Proceedings of Advances in Modal Logic*, pages 329–348, 2000.

24 Carsten Lutz and Ulrike Sattler. The complexity of reasoning with boolean modal logics. In *Advances in Modal Logic*, 2000.

25 Carsten Lutz and Ulrike Sattler. Mary likes all cats. In *Proceedings of the 2000 International Workshop on Description Logics (DL2000)*, pages 213–226, 2000.

26 Carsten Lutz, Ulrike Sattler, and Stephan Tobies. A suggestion for an n-ary description logic. In Patrick Lambrix, Alexander Borgida, Maurizio Lenzerini, Ralf Möller, and Peter F. Patel-Schneider, editors, *Proceedings of the 1999 International Workshop on Description Logics (DL'99)*, 1999.

27 Carsten Lutz, Ulrike Sattler, and Frank Wolter. Modal logic and the two-variable fragment. In *Proceedings of the 15th International Workshop on Computer Science Logic (CSL)*, pages 247–261, 2001.

28 Maarten Marx and Yde Venema. Local variations on a loose theme: Modal logic and decidability. In *Finite Model Theory and Its Applications*, pages 371–429. Springer, 2007.

29 James G. Schmolze. Terminological knowledge representation systems supporting n-ary terms. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, 1989.

30 Moshe Y. Vardi. Why is modal logic so robustly decidable? In *Descriptive Complexity and Finite Models*, pages 149–183. American Mathematical Society, 1996.

Complexity Classifications via Algebraic Logic

Reijo Jaakkola
Tampere University, Finland

Antti Kuusisto
Tampere University, Finland
University of Helsinki, Finland

Abstract

Complexity and decidability of logics is an active research area involving a wide range of different logical systems. We introduce an algebraic approach to complexity classifications of computational logics. Our base system GRA, or general relation algebra, is equiexpressive with first-order logic FO. It resembles cylindric algebra but employs a finite signature with only seven different operators, thus also giving a very succinct characterization of the expressive capacities of first-order logic. We provide a comprehensive classification of the decidability and complexity of the systems obtained by limiting the allowed sets of operators of GRA. We also discuss variants and extensions of GRA, and we provide algebraic characterizations of a range of well-known decidable logics.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Decidability, complexity, algebraic logic, fragments of first-order logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.27

Related Version *Full Version*: https://arxiv.org/abs/2005.01184

Funding The authors were funded by the Academy of Finland project Theory of computational logics, grant numbers 324435, 328987 (to December 2021) and 352419, 352420, 353027 (2022 onwards).
Reijo Jaakkola: Academy of Finland project Theory of Computational Logics, grant 328987
Antti Kuusisto: Academy of Finland project Theory of Computational Logics, grants 324435, 328987, 352419, 352420, 353027

1 Introduction

The fall of Hilbert's program and the realization of the undecidability of first-order logic FO put an end to the most prestigious plans of automating mathematical reasoning. However, research with more modest aims continued right away. Perhaps the most direct descendant of Hilbert's program was the work on the *classical decision problem*, i.e., the initiative to classify the quantifier prefix classes of FO according to whether they are decidable or not. This major program was successfully completed in the 1980's, see [7] for an overview.

Subsequent work has been more scattered but active. The current state of the art on decidability and complexity of fragments of FO divides roughly into two main branches: research on variants of *two-variable logic* FO^2 and the *guarded fragment* GF. Two-variable logic FO^2 is the fragment of FO that allows only two variable symbols. It was shown decidable in [33] and NExpTime-complete in [12]. The extension of FO^2 with counting quantifiers, or C^2, was proved decidable in [13, 35] and NExpTime-complete in [36]. Research on variants of FO^2 is currently very active, see, e.g., [5, 8, 21, 22, 29, 43] for some recent work. See also [14, 20] where the *uniform one-dimensional fragment* U_1 is defined. This system extends FO^2 to a logic that allows an arbitrary number of variables but remains NExpTime-complete.

The guarded fragment GF was initially conceived as an extension of modal logic, being a system where quantification is similarly localized as in the Kripke semantics for modal logic. After its introduction in [1], it was soon shown 2ExpTime-complete in [11]. The guarded fragment has proved successful in relation to applications, and it has been extended in several

ways, see, e.g., the *loosely guarded* [6], *clique guarded* [10] and *packed* [30] fragments. All these logics have the same 2ExpTime-complete complexity as GF (see, e.g., [4]). The more recently introduced *guarded negation fragment* GNFO [4] is a very expressive extension of GF based on restricting the use of negation in the same way GF restricts quantification. The logic GNFO also extends the *unary negation fragment* UNFO [42] which is orthogonal to GF in expressive power. GNFO shares the 2ExpTime-completeness of GF, and so does UNFO.

Other known decidable fragments of FO include the *monadic fragment* of FO [27]; the *Maslov class* [31]; the *fluted logic* [40], [37]; the *binding form* systems [32] and generalizations of prefix classes in, e.g., [44]. Moreover, in the highly active field of *description logics* [2], complexities of FO-fragments are classified in great detail, operator by operator.

The current state of the art involves a *huge number* of different logical frameworks, tailored for different purposes. Consequently the related research can be a bit scattered, and could benefit from more systematic approaches. The current article suggests an algebraic framework that enables a possible systematic approach to related studies.

Our contributions. We introduce a research method for classifying complexity and decidability of fragments of FO (and beyond) within an algebraic framework. We believe the setting nicely enables reasonably general and fine-grained studies of related questions. One of the key ideas is to always identify a *finite* collection of operators to capture the expressive power of FO or some other logic of interest – so our algebras have finite signatures. In FO, there are essentially infinitely many quantifiers $\exists x_i$ due to the different variable symbols x_i, and this issue gives rise to the infinite signature of *cylindric set algebras*, which are the principal algebraic formulation of FO. Basing our investigations on finite signatures leads to a nicely controlled setting that elucidates how the expressive power of FO arises. This is achieved by listing a finite set of operators that the expressivity of FO is based on.

The principal system we introduce is built on the algebraic signature $(e, p, s, I, \neg, J, \exists)$. The atomic terms of the related algebra are simply relation symbols (of any arity), and complex algebraic terms are built from atoms by applying the operators in the signature in the usual way. This defines an algebra over every relational structure $\mathfrak{M}$. The atomic terms R are interpreted as the corresponding relations $R^{\mathfrak{M}}$.

The operators correspond to functions that modify relations into new relations over $\mathfrak{M}$ as follows. e is the constant operator denoting the equality relation over $\mathfrak{M}$. p is a cyclic permutation operator while s is a swap operator (swapping the last two elements of tuples). I is a substitution operator, which deletes tuples whose last two members are not identical and then projects away the last coordinate of the remaining tuples (this correspond to variable substitutions in FO). $\neg$ and J are the complementation and join operators, respectively. Finally, $\exists$ is the projection operator (projects the last element away from tuples).

We let $\mathrm{GRA}(e, p, s, I, \neg, J, \exists)$ refer to the system based on these operators, with GRA standing for *general relation algebra*. To simplify notation, we also let GRA stand for $\mathrm{GRA}(e, p, s, I, \neg, J, \exists)$ in the current article. Furthermore, by $\mathrm{GRA} \setminus f_1, \ldots, f_k$ we refer to GRA with the operators $f_1, \ldots, f_k \in \{e, p, s, I, \neg, J, \exists\}$ removed.

We begin our study by proving that GRA *and* FO *are equiexpressive*. The next aim is to classify the decidability and complexity properties of the principal subsystems of GRA. Firstly, $\mathrm{GRA} \setminus \neg$ is trivially decidable, every term being satisfiable. Nevertheless, $\mathrm{GRA} \setminus \neg$ is interesting, as we show it can define precisely all conjunctive queries with equality. Then we establish that $\mathrm{GRA} \setminus \exists$ is NP-complete. We then show that satisfiability of $\mathrm{GRA} \setminus J$ can be checked by a finite automaton, and futhermore, we prove $\mathrm{GRA} \setminus I$ to be NP-complete. We thereby identify new decidable low-complexity fragments of FO. Including

the discovery of the algebraic systems, we identify, e.g., the NP-complete fragment $\mathcal{F}$ of FO based on the restriction that when forming conjunctions $\varphi(x_1,\dots,x_m)\wedge\psi(y_1,\dots,y_n)$, the sets $\{x_1,\dots,x_m\}$ and $\{y_1,\dots,y_n\}$ of variables should be disjoint.

On the negative side, we show that $\mathrm{GRA}(p,I,\neg,J,\exists)$ is Π^0_1-complete, so removing either e or s (or both) from GRA does *not* lead to decidability. Thus we have the following close to complete first classification: removing any of the operators $\neg,\exists,I,J$ gives a decidable system, while dropping e or s (or both) keeps the system undecidable. The only open case concerning the removal of a single operator is $\mathrm{GRA}\setminus p$. We leave the study of the complexity and decidability of subsystems of GRA there in this introductory article.

To push our approach further, we define a general notion of a *relation operator* which puts connectives and (generalized) quantifiers under the same umbrella concept. The definition is a slight generalization of the notion of a generalized quantifier due to Mostowski [34] and Lindström [26]. We then study variants of GRA with different sets of relation operators.

In particular, we characterize the guarded fragment, two-variable logic, fluted logic and unary negation fragment by algebras. The guarded fragment corresponds to the algebra $\mathrm{GRA}(e,p,s,\setminus,\dot\cap,\exists)$ where the symbol $\setminus$ denotes relative complementation and $\dot\cap$ the *suffix intersection* operator. The suffix intersection is similar to standard intersection but makes sense also when intersecting relations of different arities. Two-variable logic – over vocabularies with at most binary relation symbols – corresponds to $\mathrm{GRA}(e,s,\neg,\dot\cap,\exists)$, and fluted logic to $\mathrm{GRA}(\neg,\dot\cap,\exists)$. Finally, the unary negation fragment corresponds to $\mathrm{GRA}(e,p,s,\neg_1,J,\overline{J},\exists)$, where $\neg_1$ denotes 1-*dimensional negation* and $\overline{J}$ the dual operator of J.

Observe that the algebras for fluted logic and two-variable logic are clearly rather intimately related (note that we do not impose restrictions on relation symbol arities for fluted logic). Also, as the guarded fragment is characterized by $\mathrm{GRA}(e,p,s,\setminus,\dot\cap,\exists)$, and as $\mathrm{GRA}(e,s,\neg,\dot\cap,\exists)$ and $\mathrm{GRA}(e,p,s,\neg,\dot\cap,\exists)$ can be shown equiexpressive over vocabularies with at most binary relations, also two-variable logic and the guarded fragment are very nicely and closely linked. These kinds of results demonstrate the explanatory power and potential usefulness of comparing FO-fragments *under the same umbrella framework* based on different kinds of *finite signature algebras*. Indeed, *each finite operator set specifies precisely what the building blocks of the related logic are,* giving a nice, compact expressivity characterization.

The contributions of this article can be summarized by the following four points. **(1)** The main objective is to introduce an algebraic approach for classifying the complexity and decidability properties of logics via different finite signature algebras. **(2)** Concretely, we provide a comprehensive classification of the principal subsystems of $\mathrm{GRA}(e,p,s,I,\neg,J,\exists)$ (which itself characterizes FO). In each solved case, we also pinpoint the complexity of the system. In the process, we also identify interesting new decidable fragments of FO. **(3)** We find algebraic characterizations for FO and some of its main decidable fragments such as FO^2, guarded fragment, fluted logic and unary negation fragment. (Furthermore, we additionally find algebras that characterize conjunctive queries, equality-free FO, quantifier-free FO and the set of first-order atoms.) We also provide a 2ExpTime-completeness result for the algebra for the guarded fragment GF. This turns out to require some nice proof techniques and new notions (e.g., the notion of a *term guard*) to keep the translations between GF and the algebra polynomial. **(4)** We define a general notion of a relation operator.

Further notes on related work. We already extensively surveyed the *related work* concerning our program. We now give further related information on algebraic issues.

There are various algebraic approaches to FO, e.g., Tarski's *cylindric algebras*, their semantic counterparts *cylindric set algebras* and the *polyadic algebras* of Halmos. The book [16] gives a comprehensive and relatively recent account of the these systems. Also,

variants of Codd's *relational algebra* [9] are very important. The main systems studied within that setting relate to domain independent first-order logic. The closest approach to our system is Quine's *predicate functor logic* [38, 39, 41]. This system comes in several variants, with different sets of operators used. Variants of predicate functor logic can be naturally considered to be within the scope of our research program. Predicate functor logic has been studied very little, and we are not aware of any work relating to complexity theory that predates our current work. The notable works within this framework include the complete axiomatizations given in [23] and [3]. Concerning further algebraic settings, Tarski's *relation algebra* (see [16]) is also related to our work, but focuses on binary relations. For some recent results on algebras of relations, see, e.g., [15].

The article [19] is the arXiv-preprint of the current article with full proofs of all results, with the exception of Theorem 6 which will be proved in the final full version. [19] has already been followed-up by [17], where several natural extensions of so-called *ordered logic*, fluted logic and FO^2 were studied within the algebraic framework introduced below. [18] is the first version of the prerint [19] of the current article. The article [18] contains many of the results below, but using a slightly different set of algebraic operators. The research program realized in the current article was proposed in [24]. That article discusses the FO-equivalence of the operators of [18] and suggests, for example, studying systems with limited permutations.

2 Preliminaries

Let A be an arbitrary set. As usual, a ***k*-tuple** over A is an element of A^k. When $k = 0$, we let ϵ denote the unique **0-tuple** in $A^k = A^0$. Note that $A^0 = B^0 = \varnothing^0 = \{\epsilon\}$ for all sets A and B. Note also that $\varnothing^k = \varnothing$ for all positive integers k. If k is a non-negative integer, then a k-ary **AD-relation** over a set A is a pair (R, k) where $R \subseteq A^k$ is a k-ary relation in the usual sense, i.e., simply a set of k-tuples. We call $(\varnothing, k)$ the **empty *k*-ary AD-relation**. For a non-negative integer k, we let $\top_k$ (respectively, $\bot_k$) denote an operator that maps any set A to the AD-relation $\top_k(A) := (A^k, k)$ (respectively, $\bot_k(A) := (\varnothing, k)$). We may write $\top_k^A$ for $\top_k(A)$ and simply $\top_0$ for $\top_0(A)$, and we may write $\bot_k^A$ or $\bot_k$ for $\bot_k(A)$. We note that $\bot_k^{\varnothing} = \top_k^{\varnothing}$ iff $k \neq 0$. When $T = (R, k)$ is a k-ary AD-relation, we let $rel(T)$ denote R and write $ar(T) = k$ to refer to the arity of T. If T is an AD-relation, $t \in T$ *always means that* $t \in rel(T)$.

The notion of a model is defined as usual in model theory, assuming domains are never empty. For simplicity, we restrict attention to relational models, i.e., vocabularies of models do not contain function or constant symbols. The domain of a model $\mathfrak{A}$ is denoted by A, the domain of $\mathfrak{B}$ by B, et cetera. We let $\hat{\tau}$ denote the **full relational vocabulary** containing countably infinitely many relation symbols of each arity $k \geq 0$. We let $\mathrm{VAR} = \{v_1, v_2, \dots\}$ denote the countably infinite set of exactly all variables used in first-order logic FO. We also use metavariables (e.g., $x, y, z, x_1, x_2 \dots$) to refer to symbols in VAR. The syntax of FO is built as usual, starting from the set of atoms consisting of **equality atoms** (i.e., atoms with the equality symbol $=$) and **relation atoms** $R(x_1, \dots, x_n)$ where $R \in \hat{\tau}$. By an FO-formula $\varphi(x_1, \dots, x_k)$ we refer to a formula whose free variables are exactly $x_1, \dots, x_k$. An FO-formula φ (without a list of variables) may or may not have free variables. $Free(\varphi)$ denotes the set of free variables of φ. Now, let $(x_1, \dots, x_k)$ be a tuple of pairwise distinct variables and consider a formula $\varphi(x_1, \dots, x_k)$. Let also $(y_1, \dots, y_k)$ be a tuple of pairwise distinct variables. Then we let $\varphi(y_1, \dots, y_k)$ be the formula obtained from $\varphi(x_1, \dots, x_k)$ by simultaneously replacing each free variable x_i by y_i for all $i \leq k$ (and avoiding variable capture by renaming bounded variables, if necessary).

Let $k \geq 0$ and consider an FO-formula $\varphi(v_{i_1}, \ldots, v_{i_k})$ where $i_1 < \cdots < i_k$. The formula $\varphi(v_{i_1}, \ldots, v_{i_k})$ **defines** the AD-relation $\big(\{(a_1, \ldots, a_k) \in A^k \mid \mathfrak{A} \models \varphi(a_1, \ldots, a_k)\}, k\big)$ in the model $\mathfrak{A}$. Notice that we make crucial use of the linear ordering of the subindices of the variables $v_{i_1}, \ldots, v_{i_k}$ in VAR. We let $\varphi^{\mathfrak{A}}$ denote the AD-relation defined by φ in $\mathfrak{A}$. Notice – to give an example – that $\varphi(v_1, v_2, v_3)$ and $\varphi(v_6, v_8, v_9)$ define the same AD-relation over any model. It is important to recall this phenomenon below. When using the six metavariables x, y, z, u, v, w, we henceforth always assume $(x, y, z, u, v, w) = (v_{i_1}, v_{i_2}, v_{i_3}, v_{i_4}, v_{i_5}, v_{i_6})$ for some indices $i_1 < i_2 < i_3 < i_4 < i_5 < i_6$. Now, to clarify a further technical issue, let us consider the formulas $R(v_1, v_2)$ and $R(v_1, v_1)$. Observe that while $R(v_1, v_2)$ defines a binary AD-relation, the atom $R(v_1, v_1)$ defines a unary one since the only variable in it is v_1.

Consider the formulas $\varphi := v_1 \neq v_1$ and $\psi := v_1 \neq v_1 \wedge v_2 \neq v_2$. Now $\varphi^{\mathfrak{A}}$ is the empty unary AD-relation and $\psi^{\mathfrak{A}}$ the empty binary AD-relation. The negated formulas $\neg\varphi$ and $\neg\psi$ then define the universal unary and binary AD-relations $(\neg\varphi)^{\mathfrak{A}} = (A, 1)$ and $(\neg\psi)^{\mathfrak{A}} = (A \times A, 2)$, respectively. This demonstrates why we consider AD-relations rather than ordinary relations: if φ and ψ both defined the ordinary empty relation $\varnothing$ in $\mathfrak{A}$, then the action of $\neg$ in $\mathfrak{A}$ on the input $\varnothing$ would appear ambiguous.

A **conjunctive query** (CQ) is a *formula* $\exists x_1 \ldots \exists x_k \, \psi$ where ψ is a conjunction of atoms $R(y_1, \ldots, y_n)$. For example $\exists y \exists z (R(x, y, z) \wedge S(y, z, u, v))$ is a CQ with the free variables x, u, v. A **conjunctive query with equality** (CQE) is like a CQ but also allows equality atoms in addition to relation atoms $R(y_1, \ldots, y_n)$.

3 An algebra for first-order logic

In this section we define an algebra equiexpressive with FO. To this end, consider the algebraic signature $\big(e, p, s, I, \neg, J, \exists\big)$ where e is an algebraically nullary symbol (i.e., a constant), the symbols $p, s, I, \neg, \exists$ have arity one, and J has arity two. Let τ be a vocabulary, i.e., a set of relation symbols. The vocabulary τ defines a set of **terms** (or **τ-terms**) built by starting from the the symbols e and $R \in \tau$ and composing terms by using the symbols $p, s, I, \neg, J, \exists$ in the usual way. Thereby e and each $R \in \tau$ are terms, and if $\mathcal{T}$ and $\mathcal{T}'$ are terms, then so are $p(\mathcal{T})$, $s(\mathcal{T})$, $I(\mathcal{T})$, $\neg(\mathcal{T})$, $J(\mathcal{T}, \mathcal{T}')$, $\exists(\mathcal{T})$. We often leave out brackets when using unary operators and write, for example, IpR instead of $I(p(R))$. Each term $\mathcal{T}$ is associated with an arity $ar(\mathcal{T})$ which, as we will see later on, equals the arity of the AD-relation that $\mathcal{T}$ defines on a model. We define that $ar(R)$ is the arity of the relation symbol R, and we define $ar(e) = 2$; $ar(p\mathcal{T}) = ar(\mathcal{T})$; $ar(s\mathcal{T}) = ar(\mathcal{T})$; $ar(\neg\mathcal{T}) = ar(\mathcal{T})$; $ar(J(\mathcal{T}, \mathcal{T}')) = ar(\mathcal{T}) + ar(\mathcal{T}')$. Finally, for I and $\exists$, we define and $ar(I\mathcal{T}) = ar(\exists\mathcal{T}) = ar(\mathcal{T}) - 1$ if $ar(\mathcal{T}) \geq 1$ and $ar(I\mathcal{T}) = ar(\exists\mathcal{T}) = 0$ when $ar(\mathcal{T}) = 0$.

Given a model $\mathfrak{A}$ of vocabulary τ, each τ-term $\mathcal{T}$ defines an AD-relation $\mathcal{T}^{\mathfrak{A}}$ over A, and the arity of $\mathcal{T}^{\mathfrak{A}}$ will indeed be equal to the arity of $\mathcal{T}$. Consider terms $\mathcal{T}$ and $\mathcal{S}$ and assume we have defined AD-relations $\mathcal{T}^{\mathfrak{A}}$ and $\mathcal{S}^{\mathfrak{A}}$. Then the below conditions hold.

R) Let R be a k-ary relation symbol in τ, so R is a constant term in the algebra. We define $R^{\mathfrak{A}} = \big(\{(a_1, \ldots, a_k) \,|\, \mathfrak{A} \models R(a_1, \ldots, a_k)\}, k\big)$.

e) We define $e^{\mathfrak{A}} = \big(\{(a, a) | \, a \in A\}, 2\big)$. We call e the **equality** constant.

p) If $ar(\mathcal{T}) = k \geq 2$, we define $(p(\mathcal{T}))^{\mathfrak{A}} = \big(\{(a_k, a_1, \ldots, a_{k-1}) \,|\, (a_1, \ldots, a_k) \in \mathcal{T}^{\mathfrak{A}}\}, k\big)$, where $(a_k, a_1, \ldots, a_{k-1})$ is the k-tuple obtained from the k-tuple $(a_1, \ldots, a_k)$ by moving the last element a_k to the beginning of the tuple. If $ar(\mathcal{T})$ is 1 or 0, we define $(p(\mathcal{T}))^{\mathfrak{A}} = \mathcal{T}^{\mathfrak{A}}$. We call p the **cyclic permutation** operator.

s) If $ar(\mathcal{T}) = k \geq 2$, we define $\big(s(\mathcal{T})\big)^{\mathfrak{A}} = \big(\{(a_1, \ldots, a_{k-2}, a_k, a_{k-1}) \,|\, (a_1, \ldots, a_k) \in \mathcal{T}^{\mathfrak{A}}\}, k\big)$, where $(a_1, \ldots, a_{k-2}, a_k, a_{k-1})$ is the k-tuple that is obtained from the k-tuple $(a_1, \ldots, a_k)$ by swapping the last two elements a_{k-1} and a_k but keeping the other elements as they are. If $ar(\mathcal{T})$ is 1 or 0, we define $\big(s(\mathcal{T})\big)^{\mathfrak{A}} = \mathcal{T}^{\mathfrak{A}}$. We call s the **swap** operator.

I) If $ar(\mathcal{T}) = k \geq 2$, we let $\big(I(\mathcal{T})\big)^{\mathfrak{A}}$ be the AD-relation

$\big(\{(a_1, \ldots, a_{k-1}) \,|\, (a_1, \ldots, a_{k-1}, a_k) \in \mathcal{T}^{\mathfrak{A}} \text{ and } a_{k-1} = a_k\}, k-1\big)$.

If $ar(\mathcal{T})$ is 1 or 0, then $\big(I(\mathcal{T})\big)^{\mathfrak{A}} = \mathcal{T}^{\mathfrak{A}}$. We call I the **substitution** operator.

¬) Let $ar(\mathcal{T}) = k$. We define $\big(\neg(\mathcal{T})\big)^{\mathfrak{A}} = \big(\{(a_1, \ldots, a_k) \,|\, (a_1, \ldots, a_k) \in A^k \setminus rel(\mathcal{T}^{\mathfrak{A}})\}, k\big)$. Note, in particular, that if $\mathcal{T}^{\mathfrak{A}} = (\varnothing, 0) = \bot_0^A$, then $\big(\neg(\mathcal{T})\big)^{\mathfrak{A}} = (\{\epsilon\}, 0) = \top_0^A$, and vice versa, if $\mathcal{T}^{\mathfrak{A}} = \top_0^A$, then $\big(\neg(\mathcal{T})\big)^{\mathfrak{A}} = \bot_0^A$. We call $\neg$ the **complementation** operator.

J) Let $ar(\mathcal{T}) = k$ and $ar(\mathcal{S}) = \ell$. We define $\big(J(\mathcal{T}, \mathcal{S})\big)^{\mathfrak{A}}$ to be the AD-relation

$\big(\{(a_1, \ldots, a_k, b_1, \ldots, b_\ell) \,|\, (a_1, \ldots, a_k) \in \mathcal{T}^{\mathfrak{A}} \text{ and } (b_1, \ldots, b_\ell) \in \mathcal{S}^{\mathfrak{A}}\}, k + \ell\big)$.

Note that ϵ is interpreted as the identity of concatenation, so if $rel(\mathcal{T}^{\mathfrak{A}}) = \{\epsilon\}$, then $\big(J(\mathcal{T}, \mathcal{S})\big)^{\mathfrak{A}} = \big(J(\mathcal{S}, \mathcal{T})\big)^{\mathfrak{A}} = \mathcal{S}^{\mathfrak{A}}$ and $\big(J(\mathcal{T}, \mathcal{T})\big)^{\mathfrak{A}} = (\{\epsilon\}, 0)$. We call J the **join** operator.

∃) If $ar(\mathcal{T}) = k \geq 1$, we let $\big(\exists(\mathcal{T})\big)^{\mathfrak{A}}$ be the AD-relation

$\big(\{(a_1, \ldots, a_{k-1}) \,|\, (a_1, \ldots, a_k) \in \mathcal{T}^{\mathfrak{A}} \text{ for some } a_k \in A\}, k-1\big)$

where $(a_1, \ldots, a_{k-1})$ is the $(k-1)$-tuple obtained by removing the last element of $(a_1, \ldots, a_k)$. When $ar(\mathcal{T}) = 0$, then $\big(\exists(\mathcal{T})\big)^{\mathfrak{A}} = \mathcal{T}^{\mathfrak{A}}$. We call $\exists$ the **projection** operator.

We denote this algebra by $\mathrm{GRA}(e, p, s, I, \neg, J, \exists)$ where GRA stands for **general relation algebra**. A set $\{f_1, \ldots, f_k\}$ of operators defines the general relation algebra $\mathrm{GRA}(f_1, \ldots, f_k)$. In this paper – only to simplify notation – we write GRA for $\mathrm{GRA}(e, p, s, I, \neg, J, \exists)$. We identify $\mathrm{GRA}(f_1, \ldots, f_k)$ with the set of $\hat{\tau}$-terms of this algebra, where $\hat{\tau}$ is the full relational vocabulary. On the logic side, we similarly identify FO with the set of $\hat{\tau}$-formulas.

Let $\mathcal{G}$ be some set of terms of some general relation algebra $\mathrm{GRA}(f_1, \ldots, f_k)$. Formally, the satisfiability problem for $\mathcal{G}$ takes as input a term $\mathcal{T} \in \mathcal{G}$ and returns '*yes*' iff there exists a model $\mathfrak{A}$ such that $\mathcal{T}^{\mathfrak{A}}$ is not the empty AD-relation of arity $ar(\mathcal{T})$.

An FO-formula φ and term $\mathcal{T}$ are **equivalent** if $\varphi^{\mathfrak{A}} = \mathcal{T}^{\mathfrak{A}}$ for every τ-model $\mathfrak{A}$ (where τ is an arbitrary vocabulary that is large enough so that φ is a τ-formula and $\mathcal{T}$ a τ-term). For example, the formula $R(v_1, v_2)$ is equivalent to R, while $R(v_2, v_1) \wedge (P(v_1) \vee \neg P(v_1))$ is equivalent to sR. Note that under our definition, $R(v_3, v_6)$ and $R(v_1, v_2)$ are both equivalent to the term R while the formulas are not equivalent to each other. This causes no ambiguities as long as we use the terminology carefully. Also, $R(v_1, v_2) \wedge v_3 = v_3$ is *not* equivalent to the term R as it defines a ternary rather than a binary relation. Furthermore, recall that in our setting, the formula $T(v_1, v_1, v_2)$ defines a binary relation and $v_8 = v_8$ a unary relation.

It is useful to remember below how the use of the operator p is reflected to corresponding FO-formulas: if $rel\big(R^{\mathfrak{A}}\big) = \{(a, b, c, d)\} = rel\big((Rxyzu)^{\mathfrak{A}}\big)$, then $rel\big((pR)^{\mathfrak{A}}\big) = \{(d, a, b, c)\} = rel\big((Ryzux)^{\mathfrak{A}}\big)$, so the tuple (a, b, c, d) has its last element moved to the beginning of the tuple, while $Rxyzu$ has the first variable x moved to the end of the tuple of variables. It is also useful to understand how the operator I works. For example, if $rel\big(R^{\mathfrak{A}}\big) = \{(a, b, c, d)\} = rel\big((Rxyzu)^{\mathfrak{A}}\big)$, then $rel\big((IR)^{\mathfrak{A}}\big) = \{(a, b, c)\}$ if $c = d$, and else $rel\big((IR)^{\mathfrak{A}}\big) = \varnothing$. Thus IR is equivalent to $Rxyzz$ which is obtained from $Rxyzu$ by the variable substitution $u \mapsto z$.

Let S_1 be a set of terms of our algebra and S_2 a set of FO-formulas. Then S_1 and S_2 are **equiexpressive** if each term in S_1 has an equivalent formula in S_2 and conversely each formula in S_2 an equivalent term in S_1. The sets S_1 and S_2 are called **sententially equiexpressive** if each *sentence* of S_2 has an equivalent 0-ary term in S_1 and conversely each 0-ary term S_1 has an equivalent sentence in S_2.

▶ **Theorem 1.** FO *and* GRA *are equiexpressive.*

Proof. Translating terms to FO formulas is straightforward. Suppose then that $\varphi \in$ FO. Consider first the cases where φ is one of the atoms $x = x$, $x = y$. Then the corresponding terms are, respectively, Ie and e. Assume then that φ is $R(v_{i_1}, \ldots, v_{i_k})$ for $k \geq 0$. Suppose first that no variable symbol gets repeated in the tuple $(v_{i_1}, \ldots, v_{i_k})$ and that $i_1 < \cdots < i_k$. Then the term R is equivalent to φ. We then consider the cases where $(v_{i_1}, \ldots, v_{i_k})$ may have repetitions and the variables may not be linearly ordered (i.e., $i_1 < \cdots < i_k$ does not necessarily hold). We can permute any relation in every possible way by using the operators p and s; for the sake of completeness, we present the following steps that prove this claim.

Consider a tuple $(a_1, \ldots, a_i, \ldots, a_k)$ of the relation $R^{\mathfrak{A}}$ in a model $\mathfrak{A}$. Now, we can move the element a_i an arbitrary number n of steps to the left (while keeping the rest of the tuple otherwise in the same order) by doing the following: **(1)** repeatedly apply p to the term R, making a_i the rightmost element of the tuple; **(2)** apply then the *composed* function ps (so s first and then p) precisely n times; **(3)** Apply p repeatedly to put the tuple into the ultimate desired order. Moving a_i to the right is similar. Intuitively, we keep moving a_i to the *left* and continue even when it has gone past the leftmost element of the original tuple. Formally, we can move a_i by n steps to the right by performing the above three steps so that in step 2, we apply the composed function ps exactly $k - n - 1$ times.

This shows that we can move an arbitrary element anywhere in the tuple, and thereby it is clear that with p and s, we can permute a relation in all possible ways. Since we indeed can permute tuples without restrictions, we can also deal with the possible repetitions of variables in $R(v_{i_1}, \ldots, v_{i_k})$. Indeed, we can bring any two elements to the right end of a tuple and then use I. We discussed this phenomenon already above, but for extra clarity, we once more illustrate the issue by providing a related, concrete example. So let us consider the formula $R(v_1, v_2, v_1)$ (which defines a *binary* relation). We observe that $R(v_1, v_2, v_1)$ is equivalent to the term $pIpp(R)$, so we first use p twice to permute R, then we use I to identify coordinates, and finally we use p once more.

So, to sum up, we permute tuples by p and s and we use I for identifying variables. Thus, using p, s, I, we can find an equivalent term for every quantifier-free formula $R(v_{i_1}, \ldots, v_{i_k})$.

Now suppose we have equivalent terms $\mathcal{S}$ and $\mathcal{T}$ for formulas φ and ψ, respectively. We will discuss how to translate $\neg\varphi$, $\varphi \wedge \psi$ and $\exists v_i \varphi$. Firstly, clearly $\neg\varphi$ can be translated to $\neg\mathcal{S}$. Translating $\varphi \wedge \psi$ is done in two steps. Suppose φ and ψ have, respectively, the free variables $v_{i_1}, \ldots, v_{i_k}$ and $v_{j_1}, \ldots, v_{j_\ell}$. We first write the term $J(\mathcal{S}, \mathcal{T})$ which is equivalent to $\chi(v_1, \ldots, v_{k+\ell}) := \varphi(v_1, \ldots, v_k) \wedge \psi(v_{k+1}, \ldots, v_{k+\ell})$; note here the new lists of variables. We then deal with the possible overlap in the original sets $\{v_{i_l}, \ldots, v_{i_k}\}$ and $\{v_{j_l}, \ldots, v_{j_\ell}\}$ of variables of φ and ψ. This is done by repeatedly applying p, s and I to $J(\mathcal{S}, \mathcal{T})$ in the same way as used above when dealing with atomic formulas. Indeed, we above observed that we can arbitrarily permute relations and identify variables by using p, s, I. Finally, translating $\exists v_i \varphi$ is easy. We first repeatedly apply p to the term $\mathcal{S}$ corresponding to φ to bring the element to be projected away to the right end of each tuple. Then we use $\exists$. After this we again use p repeatedly to put the term into the final wanted form. ◀

The following corollary is now easy to extract from the above proof. It gives a nice, *algebraic characterization of atomic formulas* of FO.

▶ **Corollary 2.** GRA(p, s, I) *is equiexpressive with the set of relation atoms of* FO*, and* GRA(e, p, s, I) *is equiexpressive with the set of all atoms of* FO*.*

4 Relation operators and fragments of first-order logic

The FO-equivalent algebra $\mathrm{GRA} = \mathrm{GRA}(e, p, s, I, \neg, J, \exists)$ is *only one of many interesting related systems*. Defining alternative algebras equiexpressive with FO is one option, but it is also interesting to consider weaker, stronger and orthogonal systems. We next give a general definition that enables classifying all such algebras in a systematic way. In the definition, AD_A is the set of all AD-relations (of every arity) over a set A. If $T_1, \ldots, T_k$ are AD-relations over A, then $(A, T_1, \ldots, T_k)$ is called an **AD-structure**. A bijection $g : A \to B$ is an isomorphism between AD-structures $(A, T_1, \ldots, T_k)$ and $(B, S_1, \ldots, S_k)$ if $ar(T_i) = ar(S_i)$ for each i and g is an ordinary isomorphism between $(A, rel(T_1), \ldots, rel(T_k))$ and $(B, rel(S_1), \ldots, rel(S_k))$.

▶ **Definition 3.** *A k-ary **relation operator** f is a map that outputs, given an arbitrary set A, a k-ary function $f^A : (\mathrm{AD}_A)^k \to \mathrm{AD}_A$. The operator f is isomorphism invariant in the sense that if the* AD*-structures $(A, T_1, \ldots, T_k)$ and $(B, S_1, \ldots, S_k)$ are isomorphic via $g : A \to B$, then $(A, f^A(T_1, \ldots, T_k))$ and $(B, f^B(S_1, \ldots, S_k))$ are, likewise, isomorphic via g.*

An **arity-regular relation operator** is a relation operator with the property that the arity of the output AD-relation depends only on the sequence of arities of the input AD-relations.

To illustrate the notion of a relation operator, let us consider some concrete examples. Suppose $\mathcal{T}$ and $\mathcal{S}$ are both of arity k. We define $(\mathcal{T} \cup \mathcal{S})^{\mathfrak{A}} = (rel(\mathcal{T}^{\mathfrak{A}}) \cup rel(\mathcal{S}^{\mathfrak{A}}), k)$, $(\mathcal{T} \cap \mathcal{S})^{\mathfrak{A}} = (rel(\mathcal{T}^{\mathfrak{A}}) \cap rel(\mathcal{S}^{\mathfrak{A}}), k)$, and $(\mathcal{T} \setminus \mathcal{S})^{\mathfrak{A}} = (rel(\mathcal{T}^{\mathfrak{A}}) \setminus rel(\mathcal{S}^{\mathfrak{A}}), k)$. And if $\mathcal{T}$ and $\mathcal{S}$ have different arities, then $\cap$ and $\cup$ return $(\varnothing, 0)$ and $\setminus$ returns $\mathcal{T}^{\mathfrak{A}}$. Suppose then that $\mathcal{T}$ and $\mathcal{S}$ have arities k and ℓ, respectively. Calling $m := max\{k, \ell\}$, we let

$$(\mathcal{T} \mathbin{\dot\cap} \mathcal{S})^{\mathfrak{A}} = \big(\{(a_1, \ldots, a_m) \mid (a_{m-k+1}, \ldots, a_m) \in \mathcal{T}^{\mathfrak{A}} \text{ and } (a_{m-\ell+1}, \ldots, a_m) \in \mathcal{S}^{\mathfrak{A}}\},\ m\big),$$

so intuitively, the tuples overlap on some suffix of $(a_1, \ldots, a_m)$ (note that when k or ℓ is zero, then (a_{m+1}, a_m) denotes the 0-tuple ϵ). We call $\dot\cap$ the **suffix intersection**.

In the next section we prove that the guarded fragment GF is sententially equivalent to $\mathrm{GRA}(e, p, s, \setminus, \dot\cap, \exists)$. We note that in [25, 16], the authors define Codd-style relational algebra systems (with inherently infinite signatures) and they then prove their systems to be sententially equiexpressive with GF. The system in [25] uses, e.g., a *semijoin operator*, which is a join operation but employs also a conjunction of identity atoms as part of the input to it. The algebra of [16] employs, e.g., an essentially ternary join operator where the extra input essentially acts as an atomic guard. Both systems have an implicit access to variables via the infinite signatures and extra features, e.g., extra inputs.

The proofs of the characterizations in [25, 16] differ considerably from our corresponding argument, the translations from algebra to logic being inherently *exponential* in [25] and [16]. We carefully develop techniques that allow us to give a *polynomial* translation from $\mathrm{GRA}(e, p, s, \setminus, \dot\cap, \exists)$ to GF, which in turn allows us to prove a 2ExpTime upper bound for the satisfiability problem of $\mathrm{GRA}(e, p, s, \setminus, \dot\cap, \exists)$, the same as that for GF. Since we will also give a polynomial translation from GF to $\mathrm{GRA}(e, p, s, \setminus, \dot\cap, \exists)$, it follows that the satisfiability problem for the algebra is 2ExpTime-complete. Thus $\mathrm{GRA}(e, p, s, \setminus, \dot\cap, \exists)$ is the first algebra for GF for which the same 2ExpTime complexity has been proved.

We next give a characterization for two-variable logic. Technically, the characterization has some similarities with, e.g., the modal-logic-based characterization in [28].

▶ **Theorem 4.** FO^2 *and* $\mathrm{GRA}(e, s, \neg, \dot\cap, \exists)$ *are sententially equiexpressive over vocabularies with at most binary relations.*

Proof. The algebra $\mathrm{GRA}(e, s, \neg, \dot\cap, \exists)$ with at most binary relation symbols clearly contains only terms of arity at most two. Thus it is easy to translate the terms into FO^2.

We then consider the converse translation. We assume that FO^2 is built using $\neg, \wedge$ and $\exists$ and treat other connectives and $\forall$ as abbreviations in the usual way.

Now, let $\varphi \in \mathrm{FO}^2$ be a *sentence* with at most binary relations, and let x and y be the two variables that occur in φ. Note indeed that φ is a sentence, not an open formula. We first convert φ into a sentence that does not contain any subformulas of type $\psi(x) \wedge \chi(y)$ (or of type $\psi(x) \vee \chi(y)$) as follows. Consider any subformula $\exists x\, \eta(x, y)$ where $\eta(x, y)$ is quantifier-free. Put η into disjunctive normal form and distribute $\exists x$ over the disjunctions. Then distribute $\exists x$ also over the conjunctions as follows. Consider a conjunction $\alpha_i(x, y) \wedge \beta_i(y) \wedge \gamma_i$ where each of α_i, β_i, γ_i are conjunctions of literals; the formula γ_i contains the nullary relation symbols and $\alpha_i(x, y)$ contains the literals of type $\pi(x, y)$ and $\pi'(x)$. We distribute $\exists x$ into $\alpha_i(x, y) \wedge \beta_i(y) \wedge \gamma_i$ so that we obtain the formula $\exists x \alpha_i(x, y) \wedge \beta_i(y) \wedge \gamma_i$. Thereby the formula $\exists x\, \eta(x, y)$ gets modified into the formula $\bigvee_{i=1}^{n} (\exists x \alpha_i(x, y) \wedge \beta_i(y) \wedge \gamma_i)$ which is of the right form and does not have x as a free variable. Next we can repeat this process for other existential quantifiers in the formula (by treating the subformulas with one free variable in the way that atoms with one free variable were treated in the above translation step for $\eta(x, y)$). Having started from the *sentence* φ, we ultimately get a sentence equivalent to φ but having no subformulas of the form $\psi(x) \wedge \chi(y)$ or of the form $\psi(x) \vee \chi(y)$.

Next we translate an arbitrary sentence $\varphi \in \mathrm{FO}^2$ that satisfies the above condition to an equivalent term. We let $v \in \{x, y\}$ denote a generic variable.

Atoms of the form $P(v)$ (respectively $v = v$) translate to P (respectively $\exists e$). Relation symbols of arity 0 translate to themselves. $R(x, y)$ translates to R and $R(y, x)$ translates to sR. And $R(v, v)$ translates to $\exists(R \mathbin{\dot\cap} e)$ and the atoms $x = y$ and $y = x$ translate to e.

Now suppose we have translated ψ to $\mathcal{T}$. Then $\neg\psi$ translates to $\neg\mathcal{T}$. If ψ has one free variable v, then $\exists v\psi$ translates to $\exists\mathcal{T}$. If ψ has two free variables, then we either translate $\exists v\psi$ to $\exists\mathcal{T}$ when v is y and to $\exists s\mathcal{T}$ when v is x.

Consider now a formula $\psi \wedge \chi$ and suppose that we have translated ψ to $\mathcal{T}$ and χ to $\mathcal{S}$. If at least one of ψ and χ is a sentence, we translate $\psi \wedge \chi$ to $(\mathcal{T} \mathbin{\dot\cap} \mathcal{S})$. Otherwise, due to the form of the sentence φ to be translated, we have $\mathrm{Free}(\psi) \cap \mathrm{Free}(\chi) \neq \varnothing$. Now $\psi(x, y) \wedge \chi(x, y)$, $\psi(y) \wedge \chi(x, y)$ and $\psi(x, y) \wedge \chi(y)$ are all translated to $\mathcal{T} \mathbin{\dot\cap} \mathcal{S}$, while $\psi(x, y) \wedge \chi(x)$ and $\psi(x) \wedge \chi(x, y)$ are translated to $s(s\mathcal{T} \mathbin{\dot\cap} \mathcal{S})$ and $s(\mathcal{T} \mathbin{\dot\cap} s\mathcal{S})$, respectively. ◀

We note that limiting our algebraic characterizations of GF and FO^2 to *sentential* equiexpressivity is a choice based on the relative elegance of the results. Sentential equiexpressivity suffices for the almost all practical scenarios.

Now, let $\mathrm{GRA}_2(e, s, \neg, \dot\cap, \exists)$ denote the terms of $\mathrm{GRA}(e, s, \neg, \dot\cap, \exists)$ that use at most binary relation symbols; there are no restrictions on term arity, although clearly at most binary terms arise. The proof of Theorem 4 gives a translation of FO^2-sentences to $\mathrm{GRA}_2(e, s, \neg, \dot\cap, \exists)$. However, the translation is not polynomial, so we do not immediately get a NexpTime lower bound for the satisfiability of $\mathrm{GRA}_2(e, s, \neg, \dot\cap, \exists)$. Nevertheless, we can prove the following.

▶ **Theorem 5.** *The satisfiability problem of* $\mathrm{GRA}_2(e, s, \neg, \dot\cap, \exists)$ *is NexpTime-complete.*

We then briefly consider fluted logic (FL) and the unary negation fragment (UNFO). The logic FL is a decidable fragment of FO that has recently received increased attention in the research on first-order fragments, see. e.g., [17, 37]. Now, it is straightforward to show that fluted logic is equiexpressive with $\mathrm{GRA}(\neg, \dot\cap, \exists)$. The logic UNFO is a well-established decidable fragment of FO that enjoys many of the desirable properties that modal logics have [42]. Roughly speaking, its syntax is obtained from that of FO by restricting the use of

negation only to formulas that have at most one free variable. To characterize UNFO, we will need to introduce two additional relation operators. Suppose that $\mathcal{T}$ and $\mathcal{S}$ are terms of arity k AND ℓ respectively. We define

$(\overline{J}(\mathcal{T},\mathcal{S}))^{\mathfrak{A}} = (\{(a_1,\ldots,a_k,b_1,\ldots,b_\ell) \mid (a_1,\ldots,a_k) \in \mathcal{T}^{\mathfrak{A}} \text{ or } (b_1,\ldots,b_\ell) \in \mathcal{S}^{\mathfrak{A}}\}, k+\ell)$. Thus $\overline{J}$ is the *dual* of J. If $k \leq 1$, we define $(\neg_1(\mathcal{T}))^{\mathfrak{A}} = (\neg(\mathcal{T}))^{\mathfrak{A}}$, and otherwise $(\neg_1(\mathcal{T}))^{\mathfrak{A}} = \bot_0$. We call $\neg_1$ the **one-dimensional negation**. It can be shown that UNFO is equiexpressive with $\mathrm{GRA}(e,p,s,I,\neg_1,J,\overline{J},\exists)$.

By comparing the algebraic characterizations, we observe FO^2 and fluted logic are very interestingly and intimately related, and the full system $\mathrm{GRA}(e,s,\neg,\dot{\cap},\exists)$ obviously contains both fluted logic and FO^2. Note also the close relationship of these systems to the algebra $\mathrm{GRA}(e,p,s,\setminus,\dot{\cap},\exists)$ for GF. These connections demonstrate how the algebraic approach can nicely elucidate the relationships between seemingly different kinds of fragments of FO. Indeed, FO^2, FL and GF seem much more closely related than one might first suspect.

Another advantage of our algebraic approach is that it naturally suggests new fragments, as one can select their favourite operators and consider the resulting algebras. Inspired by the present work, in [17] this idea was put into action and various interesting extensions of ordered and fluted logic were studied. Here we point out yet another natural fragment inspired by our algebraic approach, namely the algebra $\mathrm{GRA}(e,s,\setminus,\dot{\cap},\exists)$. This algebra is interesting since, e.g., it contains the guarded FO^2 and guarded FL on the level of sentences.

▶ **Theorem 6.** *The satisfiability problem of* $\mathrm{GRA}(e,s,\setminus,\dot{\cap},\exists)$ *is* ExpTime*-complete.*

The proof of this theorem is quite involved, but it follows a well-known path: we design a polynomial space alternating Turing machine which tries to construct a tree-like model for the input term. The key point here is that since permutations are heavily restricted in $\mathrm{GRA}(e,s,\setminus,\dot{\cap},\exists)$, the "nodes" of the tree-like model have polynomial size. The proof will be presented in the final full version. Note that it is straightforward to read a first-order syntax for this system from the algebra. It essentially mixes the ideas behind FO^2 and the guarded variant of fluted logic.

5 An algebra for the guarded fragment

In this section we consider $\mathrm{GRA}(e,p,s,\setminus,\dot{\cap},\exists)$ and show that it is sententially equiexpressive with GF. Recall that GF is the logic that has all atoms $R(x_1,\ldots,x_k)$, $x=y$ and $x=x$, is closed under $\neg$ and $\wedge$, but existential quantification is restricted to patterns $\exists x_1 \ldots \exists x_k(\alpha \wedge \psi)$ where α is an atomic formula (a guard) having (at least) all the free variables of $\psi \in \mathrm{GF}$.

We start by defining the notion of a **term guard** of a term $\mathcal{T}$. Term guards are a central concept in our proof. The term guard of a term $\mathcal{T}$ of $\mathrm{GRA}(e,p,s,\setminus,\dot{\cap},\exists)$ is a tuple $(\mathcal{S},(i_1,\ldots,i_k))$, where $\mathcal{S} \in \mathrm{GRA}(e)$ and $k = ar(\mathcal{T}) \leq ar(\mathcal{S})$, with the following properties.

1. The tuple $(i_1,\ldots,i_k)$ consists of pairwise distinct integers i_j such that $1 \leq i_j \leq ar(\mathcal{S})$.
2. For every model $\mathfrak{A}$ and every tuple $(a_1,\ldots,a_k) \in \mathcal{T}^{\mathfrak{A}}$, there exists a tuple $(b_1,\ldots,b_{ar(\mathcal{S})}) \in \mathcal{S}^{\mathfrak{A}}$ such that $(a_1,\ldots,a_k) = (b_{i_1},\ldots,b_{i_k})$.

The intuition is that the term guard $(\mathcal{S},(i_1,\ldots,i_k))$ of $\mathcal{T}$ gives an atomic term $\mathcal{S}$ and a list $(i_1,\ldots,i_k)$ of coordinate positions (of tuples of $\mathcal{S}^{\mathfrak{A}}$) that guard the tuples of $\mathcal{T}^{\mathfrak{A}}$. The remaining $ar(\mathcal{S})-k$ coordinate positions of the tuples of $\mathcal{S}^{\mathfrak{A}}$ are intuitively non-guarding. The following lemma will be used below when translating algebraic terms to GF.

▶ **Lemma 7.** *Every term* $\mathcal{T} \in \mathrm{GRA}(e,p,s,\setminus,\dot{\cap},\exists)$ *has a term guard. Furthermore the term guard can be computed from* $\mathcal{T}$ *in polynomial time.*

Proof. We will define inductively a mapping which maps each term $\mathcal{T}$ of the system $\mathrm{GRA}(e, p, s, \setminus, \dot{\cap}, \exists)$ to a term guard for $\mathcal{T}$. We start by defining that e will be mapped to $(e, (1, 2))$ and that every relational symbol R will be mapped to $(R, (1, \ldots, ar(R)))$.

Suppose then that we have mapped a k-ary term $\mathcal{T}$ to the term guard $(\mathcal{S}, (i_1, \ldots, i_k))$. Using the term guard $(\mathcal{S}, (i_1, \ldots, i_k))$ as a starting point, we will construct term guards for the terms $p\mathcal{T}$ and $s\mathcal{T}$. Firstly, term guard for $p\mathcal{T}$ will be $(\mathcal{S}, (i_k, i_1, \ldots, i_{k-1}))$, where we have simply permuted the tuple $(i_1, \ldots, i_k)$ with p. (Note that if $k \leq 1$, then the permuted tuple is the same as the original tuple, as p leaves tuples of length up to 1 untouched.) Similarly, the term guard for $s\mathcal{T}$ will be $(\mathcal{S}, (i_1, \ldots, i_{k-2}, i_k, i_{k-1}))$, where this time we have permuted the tuple $(i_1, \ldots, i_k)$ with s. (Again if $k \leq 1$, the permuted tuple is the original tuple.)

The other cases are similar. Recall the assumption that we have mapped a k-ary term $\mathcal{T}$ to the term guard $(\mathcal{S}, (i_1, \ldots, i_k))$, and suppose further that an ℓ-ary term $\mathcal{P}$ has been mapped to the term guard $(\mathcal{S}', (j_1, \ldots, j_\ell))$. If $k \geq \ell$, then $\mathcal{T} \dot{\cap} \mathcal{P}$ will be mapped to $(\mathcal{S}, (i_1, \ldots, i_k))$, and if $k < \ell$, then $\mathcal{T} \dot{\cap} \mathcal{P}$ will be mapped to $(\mathcal{S}', (j_1, \ldots, j_\ell))$. Independently of how the arities k and ℓ are related, $\mathcal{T} \setminus \mathcal{P}$ will always be mapped to $(\mathcal{S}, (i_1, \ldots, i_k))$. (Recall that if the arities of the terms $\mathcal{Q}$ and $\mathcal{R}$ differ, then by definition $\mathcal{Q} \setminus \mathcal{R}$ is equivalent to $\mathcal{Q}$). If $k \geq 1$, the term $\exists\mathcal{T}$ will be mapped to $(\mathcal{S}, (i_1, \ldots, i_{k-1}))$. If $k = 0$, $\exists\mathcal{T}$ maps to the same term guard as $\mathcal{T}$.

Since this mapping is clearly computable in polynomial time, the claim follows. ◀

▶ **Theorem 8.** $\mathrm{GRA}(e, p, s, \setminus, \dot{\cap}, \exists)$ *and* GF *are sententially equiexpressive.*

Proof. We will first show that for every formula $\exists x_1 \ldots \exists x_k \, \psi$ of GF, there is an equivalent term of $\mathrm{GRA}(e, p, s, \setminus, \dot{\cap}, \exists)$. We begin our (ultimately inductive) argument by first showing this for a formula $\varphi := \exists x_1 \ldots \exists x_k \, \psi$ where ψ is quantifier-free. We assume $\varphi = \exists x_1 \ldots \exists x_k (\alpha(y_1, \ldots, y_n) \wedge \beta(z_1, \ldots, z_m))$ where $\alpha(y_1, \ldots, y_n)$ is an atom and $\{z_1, \ldots, z_m\} \subseteq \{y_1, \ldots, y_n\}$ and $\{x_1, \ldots, x_k\} \subseteq \{y_1, \ldots, y_n\}$.

Now consider a conjunction $\alpha \wedge \rho$ where $\alpha = \alpha(y_1, \ldots, y_n)$ is our guard atom and ρ an arbitrary atom whose set of variables is a subset of $\{y_1, \ldots, y_n\}$. We call such a conjunction an *α-guarded atom*. For each α-guarded atom, we can find an equivalent term as follows. First, by Corollary 2, we can write a term equivalent to any atomic FO-formula using e, p, s, I; note that we can use I in $\mathrm{GRA}(e, p, s, \setminus, \dot{\cap}, \exists)$, since if $ar(\mathcal{T}) > 1$, then $I\mathcal{T}$ is equivalent to $\exists(\mathcal{T} \dot{\cap} e)$, and if $ar(\mathcal{T}) \leq 1$, then $I\mathcal{T}$ is equivalent to $\mathcal{T}$. Therefore we can find terms $\mathcal{T}_\alpha$ and $\mathcal{T}_\rho$ equivalent to α and ρ, respectively. Now, the term $\mathcal{T}_\alpha \dot{\cap} \mathcal{T}_\rho$ is not likely to be equivalent to $\alpha \wedge \rho$, as the variables in $\alpha \wedge \rho$ can be unfavourably ordered instead of matching each other nicely. However – recalling that p and s can be composed to produce arbitrary permutations – we first permute $\mathcal{T}_\alpha$ to match $\mathcal{T}_\rho$ at the last coordinates of tuples, then we combine the terms with $\dot{\cap}$, and finally we permute the obtained term to the final desired form. In this fashion we obtain a term for an arbitrary α-guarded atom.

Now recall the formula $\alpha(y_1, \ldots, y_n) \wedge \beta(z_1, \ldots, z_m)$ from above. For each atom γ in β, let $\mathcal{T}_\gamma^\alpha$ denote the term equivalent to the α-guarded atom $\alpha \wedge \gamma$ formed from γ. The formula β is a Boolean combination composed from atoms by using $\neg$ and $\wedge$. We let $\mathcal{T}_\beta$ denote the term obtained from β by replacing each atom γ by the term $\mathcal{T}_\gamma^\alpha$, each $\wedge$ by $\dot{\cap}$ and each $\neg$ by relative complementation with respect to $\mathcal{T}_\alpha$ (i.e., formulas $\neg\eta$ become replaced by $\mathcal{T}_\alpha \setminus \eta^*$ where η^* is the translation of the formula η). It is easy to show that $\mathcal{T}_\beta$ is equivalent to $\alpha(y_1, \ldots, y_n) \wedge \beta(z_1, \ldots, z_m)$. Thus we can clearly apply p and $\exists$ in a suitable way to the term $\mathcal{T}_\beta$ to get a term equivalent to the formula $\varphi = \exists x_1 \ldots \exists x_k (\alpha(y_1, \ldots, y_n) \wedge \beta(z_1, \ldots, z_m))$.

Thus we managed to translate φ. To get the full translation, we mainly just keep repeating the procedure just described. The only difference is that above the formula $\beta(z_1, \ldots, z_m)$ was a Boolean combination of atoms, while now β will also contain formulas of the form

$\exists x_1 \dots \exists x_r(\delta \wedge \eta)$ in addition to atoms. Proceeding by induction, we get a term equivalent to $\exists x_1 \dots \exists x_r(\delta \wedge \eta)$ by the induction hypothesis, and otherwise we proceed exactly as described above. This concludes the argument for translating formulas to terms.

Let us then translate terms into equivalent formulas of GF. We proceed by induction. As GF is closed under Boolean operators, the only non-trivial case is the translation of $\exists$. The hard part in this case is to ensure that we can translate $\exists$ so that the resulting formula has a suitable guarding pattern (with a suitable guard atom) and thereby belongs to GF.

So suppose that we have translated $\mathcal{T}$ to $\psi(v_1, \dots, v_k)$. By Lemma 7, we can find a term guard $(\mathcal{S}, (i_1, \dots, i_k))$ for $\mathcal{T}$. By the definition of term guards, $\mathcal{S}$ is e or some relation symbol R. We let m denote the arity of $\mathcal{S}$, and we let $\alpha(v_1, \dots, v_m)$ denote $R(v_1, \dots, v_m)$, if $\mathcal{S}$ is a relation symbol, and $v_1 = v_2$ if $\mathcal{S}$ is e. Notice that $\{i_1, \dots, i_k\} \subseteq \{1, \dots, m\}$ by the definition of term guards. Now define $\chi(v_{i_1}, \dots, v_{i_k}) := \exists \overline{v}(\alpha(v_1, \dots, v_m) \wedge \psi(v_{i_1}, \dots, v_{i_k}))$, where $\overline{v}$ lists those variables from $\{v_1, \dots, v_m\}$ that are *not* included in $(v_{i_1}, \dots, v_{i_k})$. Modifying $\chi(v_{i_1}, \dots, v_{i_k})$ to the formula $\chi(v_1, \dots, v_k)$ and recalling the definition of term guards, we now observe that $\exists \mathcal{T}$ is equivalent to $\exists v_k \chi(v_1, \dots, v_k)$ which is a GF-formula. ◀

As GF is 2ExpTime-complete and the above translations polynomial, the following holds.

▶ **Corollary 9.** *The satisfiability problem for* $\mathrm{GRA}(e, p, s, \setminus, \dot{\cap}, \exists)$ *is 2ExpTime-complete.*

6 Decidable fragments of GRA

In this section we identify subsystems of $\mathrm{GRA} = \mathrm{GRA}(e, p, s, I, \neg, J, \exists)$ with a decidable satisfiability problem. We concentrate on systems obtained by limiting to a subset of the operators involved. We show that removing any of the operators $\neg, \exists, J, I$ leads to decidability, and we also pinpoint the exact complexity of each system. As a by-product, we make observations about conjunctive queries (CQs) and show NP-completeness of, e.g., $\mathrm{GRA}(\neg, J, \exists)$ and $\mathrm{GRA}(I, \neg, J)$. We also give a characterization for quantifier-free FO.

Our first result concerns GRA with the complementation operation $\neg$ removed. All negation-free fragments of FO are trivially decidable – every formula being satisfiable – and thus so is $\mathrm{GRA}(e, p, s, I, J, \exists)$. Nevertheless, this system has the following very interesting property concerning conjunctive queries with equality, or CQEs.

▶ **Proposition 10.** $\mathrm{GRA}(e, p, s, I, J, \exists)$ *is equiexpressive with the set of* CQE*s. Also, the system* $\mathrm{GRA}(p, s, I, J, \exists)$ *is equiexpressive with the set of conjunctive queries* (CQ*s*).

Proof. Analyzing the proof that $\mathrm{GRA}(e, p, s, I, \neg, J, \exists)$ and FO are equiexpressive, we see that $\mathrm{GRA}(e, p, s, I, J, \exists)$ can express every formula built from relational atoms and equality atoms with conjunctions and existential quantification. Conversely, an easy induction on term structure establishes that every term of the system $\mathrm{GRA}(e, p, s, I, J, \exists)$ is expressible by a CQE. The claim for $\mathrm{GRA}(p, s, I, J, \exists)$ follows similarly, noting that e is used only to express the atoms $x = x$ and $x = y$ in the proof of Theorem 1. ◀

We then consider GRA without $\exists$.

▶ **Proposition 11.** $\mathrm{GRA}(e, p, s, I, \neg, J)$ *is equiexpressive with quantifier-free FO, and the satisfiability problem for* $\mathrm{GRA}(e, p, s, I, \neg, J)$ *is NP-complete.*

(We can sharpen the lower bound by showing that already $\mathrm{GRA}(I, \neg, J)$ is NP-complete.) We then consider the join-free fragment of GRA. It turns out to be interestingly tame, with a very low complexity:

▶ **Theorem 12.** *Satisfiability of* $\mathrm{GRA}(e, p, s, I, \neg, \exists)$ *can be checked by a finite automaton.*

We then study GRA without I and show it NP-complete. We begin by identifying a new decidable fragment $\mathcal{F}$ of FO. The new logic $\mathcal{F}$ turns out to be an interesting, low-complexity fragment of FO, as we will prove it NP-complete. The fragment $\mathcal{F}$ is defined as the set of formulas φ of FO which satisfy the following condition: if $(\psi \wedge \chi)$ is a subformula of φ, then $\mathrm{Free}(\psi) \cap \mathrm{Free}(\chi) = \varnothing$ (note here that disjunction is not treated as a primitive operator; the primitive operators of $\mathcal{F}$ are $\neg$, $\wedge$, $\exists$).

We will then establish NP-completeness of $\mathrm{GRA}\backslash I$. We will first show that the satisfiability problem of $\mathcal{F}$ is complete for NP. The upper bound is based on a non-deterministic reduction to the satisfiability problem of the set of *relational Herbrand sentences*. We note that checking satisfiability of equality-free relational Herbrand sentences is known to be PTime-complete, see Theorem 8.2.6 in [7]. However, there seems to be *no explicit proof* of the PTime-completeness of case with equality in the literature, so we provide it in [19].

▶ **Theorem 13.** *The satisfiability problem of* $\mathcal{F}$ *is NP-complete.*

Proof. For the lower bound, suppose φ is a formula of propositional logic. We obtain an equisatisfiable formula of $\mathcal{F}$ by replacing each proposition symbol p_i by the sentence $\forall x P_i(x)$.

We thus consider the upper bound. Let $\chi \in \mathcal{F}$ be a formula. First we transform χ into negation normal form, thus obtaining a formula χ'. Now note that in $\mathcal{F}$, the formula $\forall x(\varphi \vee \psi)$ is equivalent to either $(\varphi \vee \psi)$, $(\forall x\varphi \vee \psi)$ or $(\varphi \vee \forall x\psi)$ since $Free(\varphi) \cap Free(\psi) = \varnothing$. Similarly, $\exists x(\varphi \wedge \psi)$ is equivalent to $(\varphi \wedge \psi)$, $(\exists x\varphi \wedge \psi)$ or $(\varphi \wedge \exists x\psi)$. Thus we can push all quantifiers past all connectives in the formula χ' in polynomial time, getting a formula χ''.

Let C be the set of all conjunctions obtained from χ'' as follows: begin from the syntax tree of χ'' and keep eliminating disjunctions $\vee$, always keeping one of the two disjuncts. Now χ'' is satisfiable iff some $\beta \in C$ is satisfiable. Starting from χ'', we nondeterministically guess some $\beta \in C$ (without constructing C). Now, β is a conjunction of formulas $Q_1 x_1 \dots Q_k x_k\, \eta$ where $Q_i \in \{\forall, \exists\}$ for each i and η is a literal. Putting β in prenex normal form, we get a relational Herbrand sentence. In [19] we show that satisfiability of relational Herbrand sentences is PTime-complete. ◀

Using Theorem 13, we then determine the exact complexity of $\mathrm{GRA}\backslash I$.

▶ **Theorem 14.** *The satisfiability problem of* $\mathrm{GRA}(e, p, s, \neg, J, \exists)$ *is NP-complete.*

Proof. We shall first show that $\mathrm{GRA}(e, p, s, \neg, J, \exists)$-terms translate to equisatisfiable formulas of $\mathcal{F}$ in polytime: the upper bound of the current theorem then follows due to Theorem 13. We use induction on the structure of terms $\mathcal{T}$ of $\mathrm{GRA}(e, p, s, \neg, J, \exists)$. For the base case of the induction we note that e is equivalent to $v_1 = v_2$ and R to $R(v_1, \dots, v_k)$. Suppose then that $\mathcal{T}$ is equivalent to $\varphi(v_1, \dots, v_k)$. Then $\neg\mathcal{T}$ is equivalent to $\neg\varphi(v_1, \dots, v_k)$ and $\exists\mathcal{T}$ to $\exists v_k\varphi(v_1, \dots, v_k)$. We translate $s\mathcal{T}$ to the variant of $\varphi(v_1, \dots, v_k)$ that swaps v_{k-1} and v_k and $p\mathcal{T}$ to $\varphi(v_2, \dots, v_k, v_1)$. Finally, suppose that $\mathcal{T}$ translates to $\varphi(v_1, \dots, v_k)$ and $\mathcal{P}$ to $\psi(v_1, \dots, v_\ell)$. Now $J(\mathcal{T}, \mathcal{P})$ is translated to $\varphi(v_1, \dots, v_k) \wedge \psi(v_{k+1}, \dots, v_{k+\ell})$. This concludes the proof of the upper bound.

For the lower bound, we shall prove the sharper result that the satisfiability problem of $\mathrm{GRA}(\neg, J, \exists)$ is NP-hard. We give a reduction from SAT. Let φ be a formula of propositional logic. Let $\{p_1, \dots, p_n\}$ be the set of proposition symbols in φ, and let $\{P_1, \dots, P_n\}$ be a set of unary relation symbols. Let φ^* be the formula obtained from φ by replacing each symbol p_i with $\forall x P_i(x)$. It is easy to see that φ and φ^* are equisatisfiable. Finally, since $\forall x P_i(x)$ is equivalent to $\neg\exists\neg P_i$, the sentence φ^* can be expressed in $\mathrm{GRA}(\neg, J, \exists)$. ◀

7 Undecidable fragments of GRA

In this section we identify undecidable subsystems of GRA. We show that GRA is undecidable without s and also without e. The principal technical result of this section is the following theorem, dealing with GRA without both e and s. Recall that Π^0_1 denotes the set of languages with recursively enumerable complements.

▶ **Theorem 15.** *The satisfiability problem of* $\mathrm{GRA}(p, I, \neg, J, \exists)$ *is* Π^0_1*-hard.*

As the satisfiability problem of FO is Π^0_1-complete, Theorem 15 implies the following corollary.

▶ **Corollary 16.** *Satisfiability of* $\mathrm{GRA}(p, s, I, \neg, J, \exists)$ *and* $\mathrm{GRA}(e, p, I, \neg, J, \exists)$ *are* Π^0_1*-complete.*

A standard method of proving undecidability is via a reduction from the tiling problem of the grid $\mathbb{N} \times \mathbb{N}$, which is well-known to be Π^0_1-complete [7]. While we also follow this approach in our proof of Theorem 15, we have to deal with the fact that in $\mathrm{GRA}(p, I, \neg, J, \exists)$ we can only permute variables in a cyclic manner.

We begin by recalling the **tiling problem** for $\mathbb{N}\times\mathbb{N}$. A **tile** is a function $t : \{R, L, T, B\} \to C$ where C is a countably infinite set of colors. We let t_X denote $t(X)$. Intuitively, t_R, t_L, t_T and t_B are the colors of the right, left, top and bottom edges of a tile. Now, let $\mathbb{T}$ be a finite set of tiles. A **$\mathbb{T}$-tiling** of $\mathbb{N} \times \mathbb{N}$ is a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{T}$ such that for all $i, j \in \mathbb{N}$, we have $t_R = t'_L$ when $f(i, j) = t$ and $f(i + 1, j) = t'$, and similarly, $t_T = t'_B$ when $f(i, j) = t$ and $f(i, j + 1) = t'$. Intuitively, the right color of each tile equals the left color of its right neighbour, and analogously for top and bottom colors. The tiling problem for the grid $\mathbb{N} \times \mathbb{N}$ asks, with the input of a finite set $\mathbb{T}$ of tiles, if there exists a $\mathbb{T}$-tiling of $\mathbb{N} \times \mathbb{N}$. It is well known that this problem is Π^0_1-complete. We will show that satisfiability of $\mathrm{GRA}(p, I, \neg, J, \exists)$ is undecidable by reducing the tiling problem to it.

Define the **standard grid** $\mathfrak{G}_{\mathbb{N}} := (\mathbb{N} \times \mathbb{N}, R, U)$ where $R = \{((i, j), (i + 1, j)) \mid i, j \in \mathbb{N}\}$ and $U = \{((i, j), (i, j+1)) \mid i, j \in \mathbb{N}\}$. If $\mathfrak{G}$ is a structure of vocabulary $\{R, U\}$ with the binary relation symbols R and U, then $\mathfrak{G}$ is **grid-like** if there is a homomorphism $\tau : \mathfrak{G}_{\mathbb{N}} \to \mathfrak{G}$. Consider then the extended vocabulary $\{R, U, L, D\}$ where L and D are binary. Define

$$\varphi_{inverses} := \forall x \forall y (R(x, y) \leftrightarrow L(y, x)) \ \wedge \ \forall x \forall y (U(x, y) \leftrightarrow D(y, x))$$
$$\varphi_{successor} := \forall x (\exists y R(x, y) \wedge \exists y U(x, y))$$
$$\varphi_{cycle} := \ \forall x \forall y \forall z \forall u [(L(y, x) \wedge U(x, z) \wedge R(z, u)) \to D(u, y)].$$

Then define $\Gamma := \varphi_{inverses} \wedge \varphi_{successor} \wedge \varphi_{cycle}$. The intended model of Γ is the standard grid $\mathfrak{G}_{\mathbb{N}}$ extended with two binary relations, L pointing left and D pointing down.

▶ **Lemma 17.** *Let* $\mathfrak{G}$ *be a structure of vocabulary* $\{R, U, L, D\}$*. If* $\mathfrak{G} \models \Gamma$*, then there is a homomorphism from* $\mathfrak{G}_{\mathbb{N}}$ *to* $\mathfrak{G} \restriction \{R, U\}$*, i.e,. to the restriction of* $\mathfrak{G}$ *to the vocabulary* $\{R, U\}$*.*

Proof. As $\mathfrak{G}$ satisfies $\varphi_{inverses}$ and φ_{cycle}, it is easy to see that $\mathfrak{G}$ satisfies the sentence $\varphi_{grid-like} := \forall x \forall y \forall z \forall u [(R(x, y) \wedge U(x, z) \wedge R(z, u)) \to U(y, u)]$. Using this sentence and $\varphi_{successor}$, it is easy to inductively construct a homomorphism from $\mathfrak{G}_{\mathbb{N}}$ to $\mathfrak{G} \restriction \{R, U\}$. ◀

Fix a set of tiles $\mathbb{T}$. We simulate the tiles $t \in \mathbb{T}$ by unary relation symbols P_t. Let $\varphi_{\mathbb{T}}$ be the conjunction of the following sentences (the 2nd one on the first row could be dropped):

$$\forall x \bigvee\nolimits_{t \in \mathbb{T}} P_t(x) \qquad \bigwedge\nolimits_{t \neq t'} \forall x \neg (P_t(x) \wedge P_{t'}(x))$$
$$\bigwedge\nolimits_{t_R \neq t'_L} \forall x \forall y \neg (P_t(x) \wedge R(x, y) \wedge P_{t'}(y)) \qquad \bigwedge\nolimits_{t_T \neq t'_B} \forall x \forall y \neg (P_t(x) \wedge U(x, y) \wedge P_{t'}(y))$$

The following claim shows, using Lemma 17, that $\mathbb{N} \times \mathbb{N}$ is $\mathbb{T}$-tilable iff $\varphi_{\mathbb{T}} \wedge \Gamma$ is satisfiable.

▷ Claim 18. The grid $\mathbb{N} \times \mathbb{N}$ is $\mathbb{T}$-tilable iff $\varphi_{\mathbb{T}} \wedge \Gamma$ is satisfiable.

Proof. Suppose there is a model $\mathfrak{G}$ so that $\mathfrak{G} \models \varphi_{\mathbb{T}} \wedge \Gamma$. Therefore, by Lemma 17, there exists a homomorphism $\tau : \mathfrak{G}_{\mathbb{N}} \to \mathfrak{G} \restriction \{R, U\}$. Define a tiling T of $\mathbb{N} \times \mathbb{N}$ by setting $T((i,j)) = t$ if $\tau((i,j)) \in P_t$. Since $\mathfrak{G} \models \varphi_{\mathbb{T}}$ and τ is homomorphism, the tiling is well-defined and correct.

Now suppose that there is a tiling T of $\mathbb{N} \times \mathbb{N}$ using $\mathbb{T}$. Thus we can expand $\mathfrak{G}_{\mathbb{N}} = (\mathbb{N} \times \mathbb{N}, R, U)$ to $\mathfrak{G}'_{\mathbb{N}} = (\mathbb{N} \times \mathbb{N}, R, U, L, D, (P_t)_{t \in \mathbb{T}})$ in the obvious way. Clearly $\mathfrak{G}'_{\mathbb{N}} \models \varphi_{\mathbb{T}} \wedge \Gamma$. ◁

We can now prove Theorem 15; it suffices to show that $\varphi_{\mathbb{T}}$ and each sentence in Γ is expressible in $\mathrm{GRA}(p, I, \neg, J, \exists)$. Now, the sentence $\varphi_{grid-like}$ in the proof of Lemma 17 reveals the key trick in our argument. The sentence $\varphi_{grid-like}$ would be the natural choice for our argument rather than φ_{cycle}. Indeed, we could replace $\Gamma = \varphi_{inverses} \wedge \varphi_{successor} \wedge \varphi_{cycle}$ in the statement of Lemma 17 by $\varphi_{successor} \wedge \varphi_{grid-like}$, as the proof of the lemma shows. But translating $\varphi_{grid-like}$ to $\mathrm{GRA}(p, I, \neg, J, \exists)$ would become an obstacle due to the arrangement of the variables and the lack of s in the algebra. We solve this issue by using φ_{cycle} instead of $\varphi_{grid-like}$. By extending the vocabulary, we were able to formulate φ_{cycle} so that the variables in it occur in a cyclic order. The steps below will demonstrate that by using this cyclicity, we can express φ_{cycle} in $\mathrm{GRA}(p, I, \neg, J, \exists)$ even though it lacks the swap operator s.

Let us first express $\varphi_{inverses}$. Note that $\varphi_{inverses}$ is equivalent to the conjunction

$$\forall x \forall y (R(x,y) \to L(y,x)) \wedge \forall x \forall y (L(y,x) \to R(x,y)) \\ \wedge \forall x \forall y (U(x,y) \to D(y,x)) \wedge \forall x \forall y (D(y,x) \to U(x,y)).$$

Let us show how to express $\forall x \forall y (R(x,y) \to L(y,x))$ in $\mathrm{GRA}(p, I, \neg, J, \exists)$; the other conjuncts are treated similarly. Consider the formula $R(x,y) \to L(y,x)$. To express this, consider first the formula $\psi := R(x,y) \to L(z,u)$ which can be expressed by the term $\mathcal{T} = \neg J(R, \neg L)$. Now, to make ψ equivalent to $R(x,y) \to L(y,x)$, we could first write $y = z \wedge x = u \wedge \psi$ and then existentially quantify z and u away. On the algebraic side, an essentially corresponding trick is done by transitioning from $\mathcal{T}$ first to $Ip(\mathcal{T})$ and then to $IpIp(\mathcal{T})$ and finally reordering this by p, i.e., going to $pIpIp(\mathcal{T})$. This term is equivalent to $R(x,y) \to L(y,x)$. Therefore the sentence $\forall x \forall y (R(x,y) \to L(y,x))$ is equivalent to $\forall \forall pIpIp\mathcal{T}$ where $\forall = \neg\exists\neg$.

Consider then the formula $\varphi_{cycle} = \forall x \forall y \forall z \forall u [(L(y,x) \wedge U(x,z) \wedge R(z,u)) \to D(u,y)]$. In the quantifier-free part, the variables occur in a cyclic fashion, but with repetitions. We first translate the repetition-free variant $(L(v_1,v_2) \wedge U(v_3,v_4) \wedge R(v_5,v_6)) \to D(v_7,v_8)$ by using $\neg$ and J, letting $\mathcal{T}$ be the resulting term. Now we would need to modify $\mathcal{T}$ so that the repetitions are taken into account. To introduce one repetition, first use p on $\mathcal{T}$ repeatedly to bring the involved coordinates to the right end of tuples, and then use I. Here p suffices (and s is not needed) because φ_{cycle} was designed so that the repeated variable occurrences are cyclically adjacent to each other in the variable ordering. Thus it is now easy to see that we can form a term $\mathcal{T}'$ equivalent to $(L(v_1,v_2) \wedge U(v_2,v_3) \wedge R(v_3,v_4)) \to D(v_4,v_1)$, and $\mathcal{T}'$ can easily be modified to a term for φ_{cycle}.

From subformulas of $\varphi_{\mathbb{T}}$, consider the formula $\neg(P_t(x) \wedge R(x,y) \wedge P_{t'}(y))$. Here $\psi(x,y) := R(x,y) \wedge P_{t'}(y)$ is equivalent to $\mathcal{T} := IJ(R, P_{t'})$ and $P_t(x) \wedge \psi(x,y)$ thus to $pIJ(p\mathcal{T}, P_t)$. It is now easy to see how to translate the rest of $\varphi_{\mathbb{T}}$ and also $\varphi_{successor}$. ◀

8 Conclusions

The principal aim of the article has been to *introduce* an approach for systematically studying logics via algebras based on finite signatures. The technical results obtained demonstrated how the setting works.

Our work can be continued into many directions; the key is to identify relevant collections of *relation operators* and provide classifications for the thereby generated systems. This work can naturally involve systems that capture FO, but also stronger, weaker and orthogonal ones. In addition to decidability, complexity and expressive power, also completeness of equational theories (including the one for GRA) is an interesting research direction.

References

1 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.

2 Franz Baader, Ralf Küsters, and Frank Wolter. Extensions to description logics. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 219–261. Cambridge University Press, 2003.

3 John Bacon. The completeness of a predicate-functor logic. *Journal of Symbolic Logic*, 50:903–921, 1985.

4 Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. *Journal of the ACM*, 62(3):22:1–22:26, 2015.

5 Saguy Benaim, Michael Benedikt, Witold Charatonik, Emanuel Kieronski, Rastislav Lenhardt, Filip Mazowiecki, and James Worrell. Complexity of two-variable logic on finite trees. *ACM Transactions on Computational Logic*, 17(4):32:1–32:38, 2016.

6 Johan Van Benthem. Dynamic bits and pieces. ILLC research report, University of Amsterdam, 1997.

7 Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

8 Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and trees. *ACM Transactions on Computational Logic*, 17(4):31:1–31:27, 2016.

9 Edgar F. Codd. Relational completeness of data base sublanguages. *Research Report / RJ / IBM / San Jose, California*, RJ987, 1972.

10 Erich Grädel. Invited talk: Decision procedures for guarded logics. In Harald Ganzinger, editor, *Automated Deduction – CADE-16, 16th International Conference on Automated Deduction, Proceedings*, volume 1632 of *Lecture Notes in Computer Science*, pages 31–51. Springer, 1999.

11 Erich Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64(4):1719–1742, 1999.

12 Erich Grädel, Phokion Kolaitis, and Moshe Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

13 Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science LICS*, pages 306–317. IEEE, 1997.

14 Lauri Hella and Antti Kuusisto. One-dimensional fragment of first-order logic. In Rajeev Goré, Barteld P. Kooi, and Agi Kurucz, editors, *Invited and contributed papers from the tenth conference on "Advances in Modal Logic" AiML*, pages 274–293. College Publications, 2014.

15 Jelle Hellings, Catherine L. Pilachowski, Dirk Van Gucht, Marc Gyssens, and Yuqing Wu. From relation algebra to semi-join algebra: An approach to graph query optimization. *The Computer Journal*, 64(5):789–811, 2021.

16 Robin Hirsch and Ian Hodkinson. *Relation algebras by games*. North Holland, 2002.

17 Reijo Jaakkola. Ordered Fragments of First-Order Logic. In *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:14, 2021.

18 Reijo Jaakkola and Antti Kuusisto. Algebraic classifications for fragments of first-order logic and beyond. *CoRR*, abs/2005.01184v1, 2020.

19 Reijo Jaakkola and Antti Kuusisto. Algebraic classifications for fragments of first-order logic and beyond. *arXiv Preprint*, arXiv:2005.01184v2, 2021.

20 Emanuel Kieronski and Antti Kuusisto. Complexity and expressivity of uniform one-dimensional fragment with equality. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS, Proceedings, Part I*, volume 8634 of *LNCS*, pages 365–376. Springer, 2014.

21 Emanuel Kieronski, Jakub Michaliszyn, Ian Pratt-Hartmann, and Lidia Tendera. Two-variable first-order logic with equivalence closure. *SIAM Journal on Computing*, 43(3):1012–1063, 2014.

22 Emanuel Kieronski, Ian Pratt-Hartmann, and Lidia Tendera. Equivalence closure in the two-variable guarded fragment. *J. Log. Comput.*, 27(4):999–1021, 2017.

23 Steven T. Kuhn. An axiomatization of predicate functor logic. *Notre Dame Journal of Formal Logic*, 24:233–241, 1983.

24 Antti Kuusisto. On games and computation. *CoRR*, abs/1910.14603, 2019.

25 Dirk Leinders, Maarten Marx, Jerzy Tyszkiewicz, and Jan Van den Bussche. The semijoin algebra and the guarded fragment. *Journal of Logic, Language and Information*, 14(3):331–343, 2005.

26 Per Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32(3):186–195, 1966.

27 Leopold Löwenheim. Über möglichkeiten im relativkalkül. *Matematische Annalen*, 76(4):447–470, 1915.

28 Carsten Lutz, Ulrike Sattler, and Frank Wolter. Modal logic and the two-variable fragment. In Laurent Fribourg, editor, *Computer Science Logic, 15th International Workshop, CSL*, 2001.

29 Amaldev Manuel and Thomas Zeume. Two-variable logic on 2-dimensional structures. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *LIPIcs*, pages 484–499. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.

30 Maarten Marx. Tolerance logic. *Journal of Logic, Language and Information*, 10(3):353–374, 2001.

31 S. Ju. Maslov. The inverse method for establishing deducibility for logical calculi. In V. P. Orevkov, editor, *Logical and logical-mathematical calculus. Part I, Trudy Mat. Inst. Steklov*, volume 98, pages 26–87. Public, 1968.

32 Fabio Mogavero and Giuseppe Perelli. Binding forms in first-order logic. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL*, volume 41 of *LIPIcs*, pages 648–665, 2015.

33 Michael Mortimer. Reasoning about strategies: On the model-checking problem. *Mathematical Logic Quarterly*, 21(1), 1975.

34 Andrdzej Mostowski. On a generalization of quantifiers. *Fundamenta Mathematicae*, 44(1):12–36, 1957.

35 Leszek Pacholski, Wieszlaw Szwast, and Lidia Tendera. Complexity of two-variable logic with counting. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science LICS*, pages 318–327. IEEE, 1997.

36 Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.

37 Ian Pratt-Hartmann, Wieslaw Szwast, and Lidia Tendera. The fluted fragment revisited. *Journal of Symbolic Logic*, 84(3):1020–1048, 2019.

38 Willard Van Quine. Toward a calculus of concepts. *The Journal of Symbolic Logic*, I:2–25, 1936.

39 Willard Van Quine. Variables explained away. In *Proceedings of the American Philosophical Society*, 1960.

40 Willard Van Quine. On the limits of decision. In *Proceedings of the 14th International Congress of Philosophy*, volume III, pages 57–62. University of Vienna, 1969.

41 Willard Van Quine. Algebraic logic and predicate functors. In *Logic and Art*, pages 214–238. Bobbs-Merrill, Indianapolis, Indiana, 1972.

42 Luc Segoufin and Balder ten Cate. Unary negation. *Logical Methods in Computer Science*, 9(3), 2013.

43 Szymon Torunczyk and Thomas Zeume. Register automata with extrema constraints, and an application to two-variable logic. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 873–885. ACM, 2020.

44 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*, pages 1–12. IEEE, 2017.

Counting and Matching

Bart Jacobs ✉
Radboud University Nijmegen, The Netherlands

Dario Stein ✉
Radboud University Nijmegen, The Netherlands

Abstract

Lists, multisets and partitions are fundamental datatypes in mathematics and computing. There are basic transformations from lists to multisets (called "accumulation") and also from lists to partitions (called "matching"). We show how these transformations arise systematically by forgetting/abstracting away certain aspects of information, namely order (transposition) and identity (substitution). Our main result is that suitable restrictions of these transformations are isomorphisms: This reveals fundamental correspondences between elementary datatypes. These restrictions involve "incremental" lists/multisets and "non-crossing" partitions/lists. While the process of forgetting information can be precisely spelled out in the language of category theory, the relevant constructions are very combinatorial in nature. The lists, partitions and multisets in these constructions are counted by Bell numbers and Catalan numbers. One side-product of our main result is a (terminating) rewriting system that turns an arbitrary partition into a non-crossing partition, without improper nestings.

2012 ACM Subject Classification Mathematics of computing → Combinatorics

Keywords and phrases List, Multiset, Partition, Crossing

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.28

Acknowledgements We like to thank Dusko Pavlovic for lively discussions on the topic of this paper – and on much more.

1 Introduction

This paper considers three fundamental datatypes in computing and reasoning, namely lists (or sequences), multisets, and set partitions. Two characteristic properties of lists are: (1) elements in a list are ordered, and (2) elements may occur multiple times. Multisets are datatypes where the first property is dropped, but the second one is kept. Thus, a multiset is like a subset, except that elements may occur multiple times. The order of the occurrences does not matter. Finally, set partitions are collections of non-empty pairwise disjoint subsets of a given set, whose union is the whole set.

We distinguish two fundamental operations on lists, called *accumulation* (abbreviated as *acc*) and *matching* (written as *mat*). Accumulation is a function from lists to multisets that counts occurrences of elements in the list. Matching is a function from lists to partitions that registers equality of elements. Both these functions, accumulation and matching, will be used as two orthogonal operations on lists in the following situation, where the number $K \geq 1$ is a parameter.

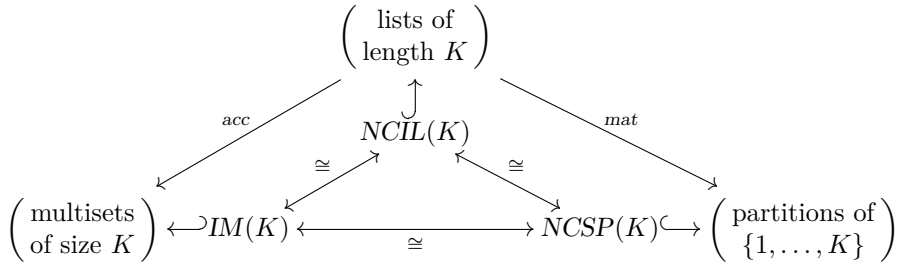
$$\begin{array}{ccc} & \left(\begin{array}{c}\text{lists of}\\ \text{length } K\end{array}\right) & \\ {\scriptstyle acc}\swarrow & & \searrow{\scriptstyle mat} \\ \left(\begin{array}{c}\text{multisets}\\ \text{of size } K\end{array}\right) & & \left(\begin{array}{c}\text{partitions of}\\ \{1,\ldots,K\}\end{array}\right) \end{array} \tag{1}$$

Intuitively, the accumulation of a list forgets the order and only considers the elements in the list with their multiplicity (number of occurrences). In matching we look at which positions equal elements occur and we put these positions in the same block (subset) of the resulting partition of $\{1, \ldots, K\}$.
These two operations *acc* and *mat* are orthogonal in the following sense:

- Accumulation is invariant under *transposition*: permuting (transposing) the elements in a list, by swapping places, does not change their accumulation.
- Matching is invariant under *substitution*: applying a permutation to the elements themselves does not change the outcome of matching.

We recall that transposition and substitution are the two basic operations in (symmetric) cryptography, used to encipher a message.

We now describe in abstract terms the main result of this paper. As in Diagram (1) we fix a parameter $K \geq 1$ and use it in a "double" manner, not only for length/size, but also as the set of elements $\{1, \ldots, K\}$ that may be used in the lists and multisets. In that case we can identify certain subsets of lists, multisets and set partitions that are in bijective correspondence, in a situation:



(2)

The abbreviation *NCIL* stands for "non-crossing incremental list, *IM* for "incremental multiset", and *NCSP* for "non-crossing set partition". Details will be provided below. These isomorphisms in the small triangle in the middle capture fundamental ways to relate the basic structures of lists, multisets and partitions. The construction of these isomorphisms is essentially combinatorial. Interestingly, the number of elements in the isomorphic sets in this sub-triangle is given by Catalan numbers – as observed in [12].

Non-crossing partitions have been introduced in the 1970s in the work of Germain Kreweras [12]. The additional property that we call "incremental" is introduced here. There is a wider story to tell about the usage of these basic datatypes in probability theory, see *e.g.* [5, 4], especially for sufficient statistics [10]. Here, however, we concentrate on the datatypes themselves, and their interconnections.

It is a bit unfortunate that two meanings of the word partition have developed in the literature, namely *set* partitions and *multiset* partitions. We only use these multiset partitions in Section 8, and we focus on set partitions first. When we simply write "partition", we mean "set partition".

The paper is organised as follows. It starts with some background information on multisets and set partitions, which allows us to define accumulation and matching in Sections 2 and 3. Subsequently, Section 4 introduces a special subset of "incremental" lists and shows that these sequences correspond bijectively to partitions. Section 5 recalls the notion of non-crossing partition (from [12]) and defines the corresponding notion on sequences. It captures proper nesting. Our main result, about the subtriangle of isomorphisms in Diagram (2) is in Section 6. A consequence of this result is a mapping from arbitrary partitions to non-crossing

partitions. Section 7 shows how this map can be obtained via a terminating rewriting system. In the final two sections we put our findings in a wider perspective: in Section 8 we show how to extend Diagram 1 to a commuting diamond via multiset partitions. This gives a wider perspective on the datatypes at hand. Finally, in Section 9 we describe how this diamond can be obtained from a general categorical construction, taking a colimit with respect to an exponent Y^X in the two different variables. This is based on Joyal's approach to combinatorics using the theory of species [11], formulated in terms of presheaves on the category of (finite) sets and bijections.

2 Multisets and accumulation

A *multiset*, or a *bag*, is a "subset" in which elements may occur multiple times. We use "ket" notation $|-\rangle$ for multisets and write for instance $3|R\rangle + 4|G\rangle + 5|B\rangle$ for a multiset with three elements R, four elements G, and five elements B. This multiset may represent an urn with three red, four green, and five blue balls. In general, a multiset over a set X is a formal finite sum $\sum_i n_i|x_i\rangle$ with $x_i \in X$ and $n_i \in \mathbb{N}$. Alternatively, such a multiset may be represented as a function $\varphi\colon X \to \mathbb{N}$ whose support $supp(\varphi) = \{x \in X \mid \varphi(x) \neq 0\}$ is finite. We write $\mathcal{M}(X)$ for the set of such multisets over X.

The *size* $\|\varphi\|$ of a multiset $\varphi \in \mathcal{M}(X)$ is its number of elements, including multiplicities, so $\|\varphi\| := \sum_x \varphi(x)$. For a number $K \in \mathbb{N}$ we write $\mathcal{M}[K](X) \subseteq \mathcal{M}(X)$ for the subset of multisets of size K.

As is common, we write $X^K = X \times \cdots \times X$ for the K-fold product of X, containing lists of length K. There is an *accumulation* map $acc\colon X^K \to \mathcal{M}[K](X)$ given by:

$$acc\big(x_1, \ldots, x_K\big) := 1|x_1\rangle + \cdots + 1|x_K\rangle.$$

Then, for instance $acc(b, a, c, b, b, a) = 2|a\rangle + 3|b\rangle + 1|c\rangle$. This accumulation map thus counts the multiplicities of each of the elements in the list. Transposing the elements in a list, by interchanging their positions, does not change the accumulation outcome. Indeed, also $acc(c, a, b, a, b, b) = 2|a\rangle + 3|b\rangle + 1|c\rangle = acc(b, a, c, b, b, a)$.

Accumulation is a fairly standard operation which can be used, for instance, to describe multinomial distributions, see [8, 9]. We include two standard combinatorial results.

▶ **Lemma 1.**

1. *For each multiset $\varphi \in \mathcal{M}(X)$, there are $\binom{\varphi}{}$ many lists that accumulate to φ, where $\binom{\varphi}{}$ is the* multinomial coefficient *of the multiset φ, given by the number:*

$$\binom{\varphi}{} := \frac{\|\varphi\|!}{\prod_x \varphi(x)!}.$$

2. *When the set X has n elements, written as $|X| = n$, then the number of elements in the set $\mathcal{M}[K](X)$ of multisets over X of size K is given by the* multichoose coefficient*:*

$$\left(\!\binom{n}{K}\!\right) := \binom{n+K-1}{K} = \frac{(n+K-1)!}{(n-1)! \cdot K!}.$$

For instance, for the multiset $\varphi = 1|a\rangle + 2|b\rangle + 1|c\rangle \in \mathcal{M}\big(\{a, b, c\}\big)$ of size $\|\varphi\| = 4$ there are $\binom{\varphi}{} = \frac{4!}{1! \cdot 2! \cdot 1!} = 12$ lists in $\{a, b, c\}^4$ that accumulate to φ, namely:

$$\begin{array}{llllll} \langle a,b,b,c\rangle & \langle a,b,c,b\rangle & \langle a,c,b,b\rangle & \langle b,a,b,c\rangle & \langle b,a,c,b\rangle & \langle b,b,a,c\rangle \\ \langle b,b,c,a\rangle & \langle b,c,a,b\rangle & \langle b,c,b,a\rangle & \langle c,a,b,b\rangle & \langle c,b,a,b\rangle & \langle c,b,b,a\rangle. \end{array}$$

3 Set partitions and matching

We shall write $\underline{K} \coloneqq \{1, 2, \dots, K\}$ for the set of the first K positive natural numbers. By a (set) partition of K, we mean a (set) partition of the set $\underline{K}$: It consists of a collection of "blocks" $B_i \subseteq \underline{K}$ of non-empty pairwise disjoint subsets B_i with $\bigcup_i B_i = \underline{K}$. Examples of partitions of K are the single block-partition $\{\underline{K}\}$ and the K-element partition $\{\{1\}, \dots, \{K\}\}$ with singleton blocks. The number of blocks in a partition of K ranges between 1 and K. We shall write $SP(K)$ for the set of set partitions of K. It comes with a size function $|-|\colon SP(K) \to \underline{K}$.

For each set X there is a *matching* function $mat\colon X^K \to SP(K)$. It forms blocks out of the positions in a list with equal elements, as in:

$$mat(b, a, c, b, b, a) \;=\; \Big\{\, \{1, 4, 5\},\ \{2, 6\},\ \{3\} \,\Big\}.$$

In general we define matching as:

$$mat(x_1, \dots, x_K) \;\coloneqq\; \bigcup_{1 \le i \le K} \Big\{ \{j \in \underline{K} \mid x_j = x_i\} \Big\}.$$

This matching operation, like accumulation, is quite standard. An early source is, for instance, [1]. It is not hard to see that matching is stable under each substitution isomorphism $X \xrightarrow{\cong} X$ that is applied elementwise to the input list.

We see that accumulation of lists in X^K is stable under *transposition* – that is, under permutations $K \xrightarrow{\cong} K$ of the positions – whereas matching is stable under *substitution* – that is, under permutations $X \xrightarrow{\cong} X$ of the elements. A systematic categorical perspective is offered in Section 9.

Again we list two basic results, without proof.

▶ **Lemma 2.**

1. *The numbers* $\big|SP(K)\big|$ *of partitions of* K*, for* $K = 1, 2, 3, \dots$*, are given by the Bell numbers:* 1, 2, 5, 15, 52, 203, 877, 4140, ...
2. *Let* X *be a finite set with* $|X| = n$ *and let* $P \in SP(K)$ *be a partition of* K *with* $|P| \le n$*. The number of lists in* X^K *that match to* P *is given by the falling factorial:*

$$(n)_{|P|} \;=\; n(n-1)(n-2)\cdots(n-|P|+1) \;=\; \frac{n!}{(n-|P|)!}.$$ ⌟

For instance, consider the four-element set $X = \{a, b, c, d\}$ and the partition $P \in SP(7)$ with two blocks:

$$P \;=\; \Big\{\, \{1, 3, 7\},\ \{2, 4, 5, 6\} \,\Big\}.$$

Then there are $\frac{4!}{(4-|A|)!} = \frac{4!}{2!} = 12$ lists in X^7 that match to P, namely:

$$\begin{array}{llll}
\langle a, b, a, b, b, b, a\rangle & \langle a, c, a, c, c, c, a\rangle & \langle a, d, a, d, d, d, a\rangle & \langle b, a, b, a, a, a, b\rangle \\
\langle b, c, b, c, c, c, b\rangle & \langle b, d, b, d, d, d, b\rangle & \langle c, a, c, a, a, a, c\rangle & \langle c, b, c, b, b, b, c\rangle \\
\langle c, d, c, d, d, d, c\rangle & \langle d, a, d, a, a, a, d\rangle & \langle d, b, d, b, b, b, d\rangle & \langle d, c, d, c, c, c, d\rangle.
\end{array}$$

4 Incremental lists and multisets

This section introduces what we call "incremental" lists and multisets. As we shall see, the numbers of these lists and multisets are described by the Bell and Catalan numbers, respectively. The main result in this section says that set partitions correspond to incremental lists.

In the previous two sections we have considered partitions, lists and multisets on an arbitrary set X. From now on, we will take the underlying set to be $X = \underline{K} = \{1, \ldots, K\}$. By taking advantage of the order on $\underline{K}$, we can obtain concrete representations of partitions in terms of lists. Of special importance in this context are the minimal elements of each block, which we name *parents*.

▶ **Definition 3.** *Let $P \in SP(K)$ be a set partition of K.*

1. *An element $a \in \underline{K}$ is called a* parent *of the partition P if it is the minimum element of its block, that is, if $a = \min(B)$ where $B \in P$ is the necessarily unique block with $a \in B$. For any element $b \in \underline{K}$, we call the least element of its block the parent of b and denote it by $par_P(b)$.*
2. *For a partition $P \in SP(K)$, its* parent list *$(s_1, \ldots, s_K) \in \underline{K}^K$ is defined as $s_i = par_P(i)$. Its* parent multiset *$\varphi \in \mathcal{M}[K](\underline{K})$ is defined as $\varphi = acc(s_1, \ldots, s_K)$.*

For example, in the partition $P \in SP(\underline{8})$ given by

$$P = \Big\{ \{\mathbf{1}, 3, 5\},\ \{\mathbf{2}, 6\},\ \{\mathbf{4}, 7, 8\} \Big\}$$

we have highlighted the parents $\mathbf{1}, \mathbf{2}, \mathbf{4}$ in bold. The parent list of P is $\vec{s} = \langle \mathbf{1}, \mathbf{2}, \mathbf{1}, \mathbf{4}, \mathbf{1}, \mathbf{2}, \mathbf{4}, \mathbf{4} \rangle$. For instance, there is a 1 in the fifth entry since $par_P(5) = \mathbf{1}$. The associated parent multiset is $\varphi = acc(\vec{s}) = 3|\mathbf{1}\rangle + 2|\mathbf{2}\rangle + 3|\mathbf{4}\rangle$. From the parent multiset, we can read off the parents of the partition, and the sizes of their respective blocks, but not which elements belong to those blocks. Explicitly, the parent multiset φ satisfies:

$$\varphi(b) = \begin{cases} 0 & \text{if } b \text{ is not a parent} \\ |B| & \text{if } b = \min(B) \text{ for } B \in P. \end{cases}$$

It is easy to see that the parent list uniquely characterises the partition, via matching. The question comes up: which lists arise as parent lists? A characterisation of such lists will be formulated below in terms of an "incremental" property. We then get an isomorphism between incremental lists and set partitions, see Theorem 5 below.

We will give a similar characterisation of parent multisets as "incremental" multisets. A partition is generally *not* uniquely characterised by its parent multiset, however we will show in Section 5 that so-called *noncrossing partitions* are.

▶ **Definition 4.**

1. *A list $\vec{s} = (s_1, \ldots, s_K) \in \underline{K}^K$, with $1 \leq s_i \leq K$, is called* incremental *if for each index $1 \leq i \leq K$, we have $s_i \leq i$ and $s_{s_i} = s_i$.*
   We shall write $IL(K) \subseteq \underline{K}^K$ for the subset of incremental lists.
2. *A multiset $\varphi \in \mathcal{M}[K]\big(\underline{K}\big)$ of size K with elements from $\underline{K} = \{1, \ldots, K\}$ is called* incremental *if for all $i \in \underline{K}$, we have*

$$\sum_{j \leq i} \varphi(j) \geq i.$$

We write $IM(K) \subseteq \mathcal{M}[K]\big(\underline{K}\big)$ for the subset of incremental multisets.

The two requirements in Definition 4 (1) express that in an incremental list $\vec{s}$, an entry s_i must be in the range $\{1, \ldots, i\}$ and $s_i = j \leq i$ can only happen if $s_j = j$, that is, if j occurs already in $\vec{s}$ at position j. This follows since $s_j = s_{s_i} = s_i = j$.

Let's make a bit more concrete what this means, for incremental lists.

- When $K = 1$, there is only one list $\langle 1\rangle \in \underline{1}^1$, which is incremental.
- For $K = 2$ there are four lists $\langle 1,1\rangle$, $\langle 1,2\rangle$, $\langle 2,1\rangle$ and $\langle 2,2\rangle$ in $\underline{2}^2$. The last two lists are not incremental because the first requirement $s_i \leq i$ fails for $i = 1$.
- For $K = 3$ we thus have five incremental lists, namely: $\langle 1,1,1\rangle$, $\langle 1,2,1\rangle$, $\langle 1,2,2\rangle$, $\langle 1,1,3\rangle$, $\langle 1,2,3\rangle$.

In a similar way:

- For $K = 1$, there is only $\varphi = 1|1\rangle \in \mathcal{M}[1]\big(\{1\}\big)$. This φ is incremental.
- For $K = 2$ we $\mathcal{M}[2]\big(\{1,2\}\big)$ contains three elements $2|1\rangle$, $1|1\rangle + 1|2\rangle$, $2|2\rangle$. Only the last one, $\varphi = 2|2\rangle$ is not incremental since $\sum_{j\leq 1} \varphi(j) = 0 \not\geq 1$.
- For $K = 3$ there are five incremental multisets, namely $3|1\rangle$, $2|1\rangle + 1|2\rangle$, $2|1\rangle + 1|3\rangle$, $1|1\rangle + 2|2\rangle$ and $1|1\rangle + 1|2\rangle + 1|3\rangle$.

It is not hard to see that for $\varphi \in IM(K)$ one has $\varphi(i) \leq K + 1 - i$, for each $i \in \underline{K}$.

▶ **Theorem 5.** *Fix a number $K \geq 1$. There are bijective correspondences between:*

1. *incremental lists $\vec{s} \in IL(K)$;*
2. *"parent" functions $p\colon \underline{K} \to \underline{K}$ forming an interior operation: $p^2 = p$ and $p \leq id$.*
3. *set partitions $P \in SP(K)$.*

Via these correspondences the match function becomes an isomorphism $\mathrm{mat}\colon IL(K) \xrightarrow{\cong} SP(K)$.

Proof. The equivalence between items (1) and (2) in Theorem 5 is obvious because a list $\vec{s} = \langle s_1, \ldots, s_K\rangle$ corresponds to a function $p\colon \underline{K} \to \underline{K}$ via $p(i) = s_i$. The two requirements $s_i \leq i$ and $s_{s_i} = s_i$ in Definition 4 (1) correspond directly to $p(i) \leq i$ and $p(p(i)) = p(i)$, that is, to $p \leq id$ and $p^2 = p$.

The equivalence between (1) and (3) is obtained by sending a partition to its parent list. ◀

A consequence of the isomorphism $IL(K) \cong SP(K)$ in Theorem 5 is that the number of incremental lists in $IL(K)$ is given by the K-th Bell number, see Lemma 2 (1).

As we will show, the accumulation map takes incremental lists to incremental multisets (Lemma 7). Therefore, we obtain a canonical map from set partitions to incremental multisets.

▶ **Proposition 6.** *The parent multiset function can be expressed by the "minimum size" function ms in:*

$$ms \;\coloneqq\; \Big(SP(K) \xrightarrow[\cong]{mat^{-1}} IL(K) \xrightarrow{\quad acc \quad} IM(K)\Big).$$

As in Definition 3 (2), it is given by the formula:

$$ms(P) \;=\; \sum_{B\in P} |B| \,\Big|\, \min(B)\Big\rangle \;. \qquad \lrcorner$$

The sets $IL(K)$ and $IM(K)$ of incremental lists and multisets can also be defined inductively. This gives a better grip and allows us to express that accumulation restricts to "incremental".

▶ **Lemma 7.**

1. *Define for $K \geq 1$ the sets $S_K \subseteq \underline{K}^K$ as:*

$$\begin{aligned} S_1 &\coloneqq \big\{\langle 1\rangle\big\} \\ S_{K+1} &\coloneqq \big\{\langle s_1, \ldots, s_K, K{+}1\rangle \;\big|\; \vec{s} \in S_K\big\} \;\cup \bigcup_{1\leq i\leq K} \big\{\langle s_1, \ldots, s_K, s_i\rangle \;\big|\; \vec{s} \in S_K\big\}. \end{aligned}$$

Then $S_K = IL(K)$.

2. *We also define sets* $M_K \subseteq \mathcal{M}[K]\big(\underline{K}\big)$ *via:*

$$\begin{aligned} M_1 &\coloneqq \big\{1|1\rangle\big\} \\ M_{K+1} &\coloneqq \big\{\varphi + 1|K{+}1\rangle \;\big|\; \varphi \in M_K\big\} \;\cup \bigcup_{1\leq i\leq K,\, \varphi(i)>0} \big\{\varphi + 1|i\rangle \;\big|\; \varphi \in M_K\big\}. \end{aligned}$$

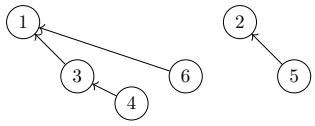
   Then $M_K = IM(K)$.
3. *The accumulation map* $\mathrm{acc}\colon \underline{K}^K \to \mathcal{M}[K]\big(\underline{K}\big)$ *restricts to* $\mathrm{acc}\colon IL(K) \to IM(K)$.

The inductive formulation, especially in item (2), is reminiscent of what is called a *Hoppe urn*, after [7]. One thinks of a multiset $\varphi \in \mathcal{M}[K]\big(\underline{K}\big)$ as an urn with K balls of K-many different colors (in $\underline{K}$). In a "Hoppe" draw an extra ball of the same colour as the drawn ball is returned to the urn but additionally another ball is added with a new, fresh colour, not already occurring in the urn. In item (2) this is represented via the addition of color with number $K+1$ in the sum $\varphi + 1|K{+}1\rangle$. Biologically, this ball of a new colour $K+1$ can be understood as a (genetic) mutation. Indeed, this structures have first been studied in population biology, see *e.g.* [4, 15].

Proof.

1. Clearly, $S_1 = \big\{\langle 1\rangle\big\} = IL(1)$. The inclusions $S_K \subseteq IL(K)$ follow by an easy induction on K. In the other direction, assume $IL(K) \subseteq S_K$; we aim to show $IL(K{+}1) \subseteq S_{K+1}$. So consider $\langle s_1, \ldots, s_K, n\rangle \in IL(K{+}1)$. Then $\vec{s} = \langle s_1, \ldots, s_K\rangle \in IL(K)$ and thus $\vec{s} \in S_K$ by induction hypothesis. The element n must be in $\{1, \ldots, K{+}1\}$. If $n = K{+}1$ we are done. If $n < K{+}1$, then $s_n = n$, so that we are also done.
2. Similarly.
3. By induction on K, using the previous two inductive characterisations. ◀

In the end, we add that the name *parent* is motivated by analogy with the disjoint-set forest data structure (sometimes called union-find data structure) [6]. Any list $\vec{p} \in \underline{K}^K$ with $p_i \leq i$ represents a forest where p_i is the parent of i. For example, the list $\langle 1, 2, 1, 3, 2, 1\rangle$ represents the forest:



This induces a partition of $\underline{K}$ by taking connected components, or successively taking parents. The forest representing a given partition is not unique. We can make it unique by forcing all trees to have depth al most 1; this corresponds to the condition $p_{p_i} = p_i$ for incremental lists.

Another common way to encode set partitions as numeric lists are *restricted growth lists* (RGS) [16]. While similar to incremental lists in many aspects, we argue here that incremental lists have convenient properties especially considered in connection with the accumulation map and with multisets.

5 Non-crossing partitions and lists

The notion of a crossing has been introduced in the literature for partitions, see [12]. We recall that definition and formulate a corresponding notion for lists. In this section we introduce the basics of such crossings, especially that they are counted by Catalan numbers. In the next section we make connections to (incremental) multisets.

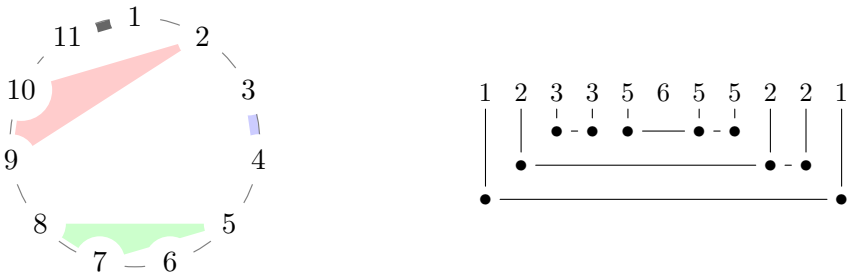
▶ **Definition 8.**

1. *In general, a list* $\vec{s} \in X^K$ *has a* crossing *if there are indices* $n < i < m < j$ *with* $s_n = s_m \neq s_i = s_j$. *The list* $\vec{s}$ *is called* non-crossing *if it has no crossings.*
   We are especially interested in lists which are both incremental and non-crossing. We write $NCIL(K) \subseteq IL(K) \subseteq \underline{K}^K$ *for the subset of such lists.*
2. *A partition* $P \in SP(K)$ *has a* crossing *if there are different blocks* $A, B \in P$ *with numbers*

   $$n < a < m < b \qquad \text{where } n, m \in A \text{ and } a, b \in B.$$

   A set partition is called non-crossing *if it has no such crossings. We shall write* $NCSP(K) \subseteq SP(K)$ *for the subset of non-crossing partitions of* $\underline{K}$.

A list is non-crossing when it involves proper nesting, like in nested blocks $\{\ldots\}$ in programming languages, or in nested brackets $(\ldots)$ in expressions. Non-crossing partitions can be visualized nicely by putting the elements of $\underline{K} = \{1, \ldots, K\}$ on a circle and taking convex hulls of the elements in each block. Non-crossing means these hulls don't overlap. Below, we illustrate these definitions for the non-crossing partition $P = \big\{\{1, 11\}, \{2, 9, 10\}, \{3, 4\}, \{5, 7, 8\}, \{6\}\big\}$, drawn on a circle on the left, without overlapping regions. The associated non-crossing incremental list $\langle 1, 2, 3, 3, 5, 6, 5, 5, 2, 2, 1\rangle$ is drawn on the right, without improper nestings.



(3)

Non-crossing partitions are counted by the Catalan numbers, as observed in [12, Cor. 4.2].

▶ **Lemma 9.** *The number of non-crossing partitions in* $NCSP(K)$ *for* $K = 1, 2, 3, \ldots$, *is given by the* Catalan *numbers:* 1, 2, 5, 14, 42, 132, 429, 1430, 4862, ... ⌟

The next result is immediate from the formulations of "non-crossing" in Definition 8.

▶ **Proposition 10.** *The match isomorphism* $IL(K) \xrightarrow{\cong} SP(K)$ *restricts to an isomorphism between* non-crossing *incremental lists and set partitions in:*

$$\begin{array}{ccc} NCIL(K) & \xrightarrow[\cong]{mat} & NCSP(K) \\ \downarrow & & \downarrow \\ IL(K) & \xrightarrow[\cong]{mat} & SP(K) \end{array}$$

⌟

6 The main sub-triangle result

We go straight to our main result, about the sub-triangle of isomorphisms in Diagram (2).

▶ **Theorem 11.** *For each* $K \geq 1$ *there are isomorphisms between non-crossing incremental lists, incremental multisets, and non-crossing set partitions, in:*

$$\begin{array}{ccc} & NCIL(K) & \\ {\scriptstyle acc}\swarrow_{\cong} & & \searrow^{\cong}{\scriptstyle mat} \\ IM(K) & \xleftarrow[\cong]{ms} & NCSP(K) \end{array}$$

As a consequence, the numbers of all these items are given by the Catalan numbers.

Proof. By Proposition 10 it suffices to prove that accumulation restricts to an isomorphism $acc\colon NCIL(K) \to IM(K)$. We first illustrate surjectivity via an exemplaric construction. Then we prove injectivity.

We describe a map $IM(K) \to NCIL(K)$ via a "stack". Let's take $\varphi = 2|1\rangle + 3|2\rangle + 2|3\rangle + 3|5\rangle + 1|6\rangle \in IM(11)$. We proceed in two steps.

1. All elements $1, 2, 3, 5, 6$ in the support of φ must become parents, so we know their required position in the list of size 11 that we need to build:

$$\ell = \langle 1, 2, 3, -, 5, 6, -, -, -, -, - \rangle. \tag{(*)}$$

2. Next we need to complete to a non-crossing incremental list. We go through the above (partial) list ℓ in $(*)$ from left to right using a stack st, which is empty initially.
   - We first encounter a 1 in the list in $(*)$. We have $\varphi(1) = 2$, which means that one more 1 needs to be placed somewhere in the list. Hence we push one 1 onto the stack, giving $st = \langle 1 \rangle$. This stack serves as an ordered memory that records the elements that we still need to put in the list.
   - The next item in the list $(*)$ is a 2, with $\varphi(2) = 3$. We proceed in the same manner and push two numbers 2 onto the stack, giving $st = \langle 1, 2, 2 \rangle$.
   - There is one more such step when we encounter 3 in $(*)$ with $\varphi(3) = 2$ and turn the stack into $st = \langle 1, 2, 2, 3 \rangle$.
   - The next thing in $(*)$ is a blank $-$. We pop the last element from the stack and put it at this empty spot, giving new list and stack:

$$\ell = \langle 1, 2, 3, 3, 5, 6, -, -, -, -, - \rangle \qquad st = \langle 1, 2, 2 \rangle.$$

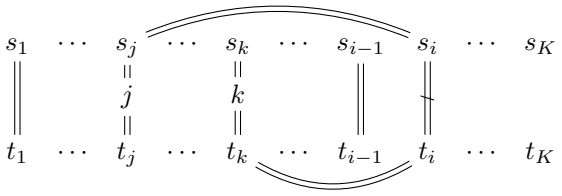
   - The next item in $(*)$ is 5, with $\varphi(5) = 3$, so we push two 5's onto the stack, giving $st = \langle 1, 2, 2, 5, 5 \rangle$.
   - Next we see a 6 in $(*)$ with $\varphi(6) = 1$, so there is no need to put more numbers 6 in the list, and we proceed without any action.
   - After 6 in $(*)$ we encounter only blanks. Thus, one-by-one we pop the items from the current stack $st = \langle 1, 2, 2, 5, 5 \rangle$ and place them in the list. This gives our outcome $\ell \in NCIL(11)$ of the form:

$$\ell = \langle 1, 2, 3, 3, 5, 6, 5, 5, 2, 2, 1 \rangle.$$

By construction, this list is non-crossing, see the diagram on the right in (3). The pushing-and-popping via the stack ensures the proper nesting. This algorithmic description can easily be generalised to an arbitrary incremental multiset.

We turn to injectivity, so let $acc(\vec{s}) = acc(\vec{t})$ for $\vec{s}, \vec{t} \in NCIL(K)$. Our aim is to show $\vec{s} = \vec{t}$. We first note that the assumption $acc(\vec{s}) = acc(\vec{t})$ implies that the sequences $\vec{s}$ and $\vec{t}$ have the same "parent" elements: $s_i = i$ iff $t_i = i$ for each i. Indeed, if $s_i = i$, then $0 < acc(\vec{s})(i) = acc(\vec{t})(i)$. This means that i must occur in $\vec{t}$ and thus $t_i = i$, since $\vec{t}$ is incremental.

Towards a contradiction, let $\vec{s} \neq \vec{t}$, and let i be the least index with $s_i \neq t_i$. Then $s_i \neq i$ and also $t_i \neq i$, by what we just noted. Thus, there is an index $j < i$ with $j = s_j = s_i$ and there is also a $k < i$ with $k = t_k = t_i$. But then $j \neq k$, since $s_i \neq t_i$. Without loss of generality we assume $j < k$. Since i is the least index where s, t differ, we have $t_j = s_j = j = s_i$ and $s_k = t_k = k = t_i$. When we write out the two lists $\vec{s}$ and $\vec{t}$ we get:



The number of occurrences of k in the segment $s_1, \ldots s_i$ is one less than in the segment $t_1, \ldots, t_i$. We do have $acc(\vec{s})(k) = acc(\vec{t})(k)$, which means that the number of occurrences of k in the whole list $\vec{s}$ is the same as the number of occurrences of k in $\vec{t}$. This means that there is an index $\ell > i$ with $s_\ell = s_k = k$. But now we have a contradiction with the non-crossing assumption for s, since we now have:

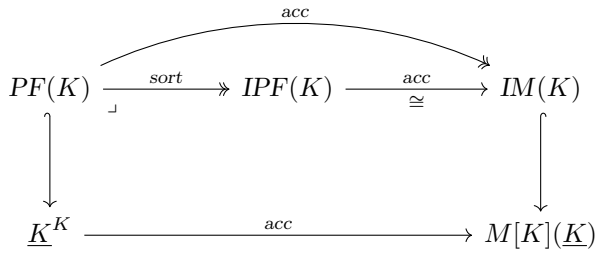
$$j < k < i < \ell \qquad \text{with} \qquad s_j = j = s_i \quad \text{and} \quad s_k = t_k = k = s_\ell. \qquad \blacktriangleleft$$

Relationships to other combinatorial families. The three isomorphic structures in Theorem 11 are counted by Catalan numbers. There are dozens of structures in the "Catalan family" which are counted in this way. Famously, sixty-six are listed in a single exercise in [14, Exc 6.19]. While we believe that incremental multisets are a novel addition, we highlight some connections to other family as follows: If $\varphi = acc(s_1, \ldots, s_K) \in IM(K)$ then

1. the list of multiplicities $\vec{w} = \langle \varphi(1) - 1, \varphi(2) - 1, \ldots, \varphi(K) - 1 \rangle$ satisfies $w_i \geq -1$, has all partial sums nonnegative and $w_1 + \ldots + w_K = 0$; that is Example 6.19.(w) of [14]
2. the increasing rearrangement $\langle s_{(1)}, \ldots, s_{(K)} \rangle = sort(s_1, \ldots, s_k)$ satisfies $1 \leq s_{(1)} \leq \ldots \leq s_{(K)}$ and $s_{(i)} \leq i$; that is Example 6.19.(s) of [14]. Both of these relationships are invertible.

Elaborating the second bullet point further leads to the well-known combinatorial notion of *parking function* (e.g. the chapter of C. Yan in [2]). A parking function is any list $\langle s_1, \ldots, s_k \rangle \in \underline{K}^K$ whose increasing rearrangement $\langle s_{(1)}, \ldots, s_{(K)} \rangle$ satisfies $1 \leq s_{(1)} \leq \ldots \leq s_{(K)}$ and $s_{(i)} \leq i$. A parking function is called *increasing* if it furthermore satisfies $s_i \leq s_j$ for $i \leq j$. Increasing parking functions are another prominent member of the Catalan family [13]. Bullet Point 2 states that they are also in bijection with incremental multisets.

It is easy to conclude that a sequence $\vec{s}$ satisfies $acc(\vec{s}) \in IM(K)$ if and only if $\vec{s}$ is a parking function. The sets $IPF(K) \subseteq PF(K)$ of increasing parking functions and parking functions thus fit into the following diagram, which becomes a pullback:



7 Un-crossing via term rewriting

The constructions in the previous sections give us a mapping from arbitrary partitions to non-crossing partitions, namely:

$$SP(K) \xrightarrow[\cong]{mat^{-1}} IL(K) \xrightarrow{acc} IM(K) \xrightarrow[\cong]{ms^{-1}} NCSP(K).$$

In this section we show how this mapping can also be obtained via rewriting. Concretely, we start with an arbitrary partition $P_0 \in SP(K)$ and successively eliminate single crossings via rewrite steps $P_0 \to P_1 \to P_2 \to \dots$ while in each step preserving the parent multiset $ms(P_i) = ms(P_0)$. We show that this reduction system strongly terminates, that is any choice of reduction sequence leads to a crossing-free normal form. By Theorem 11 this normal form must then be unique.

We begin with a preliminary observation: An *increasing crossing* in a partition $P \in SP(K)$ is a crossing $a < b < c < d$ such that $par_P(a) < par_P(b)$. It is easy to see that if a partition has a crossing, it also has an increasing crossing.

▶ **Definition 12.** *We define the "uncrossing" reduction relation $(\to)$ on $SP(K)$ as follows. If $P \in SP(K)$ and $a < b < c < d$ is an increasing crossing with $a, c \in A$ and $b, d \in B$, we define P' as the partition in which c, d switch block, giving a new partition:*

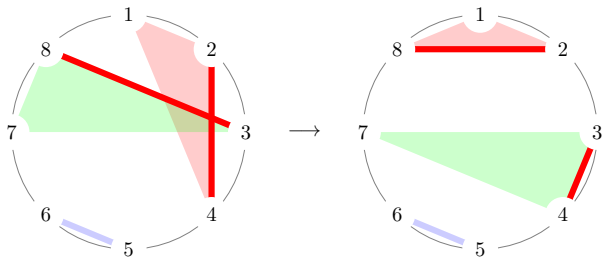
$$P' = P \setminus \{A, B\} \cup \Big\{ A \setminus \{c\} \cup \{d\},\ B \setminus \{d\} \cup \{c\} \Big\}.$$

This defines a single reduction step $P \to P'$.

There is a corresponding rewrite system on incremental lists: an increasing crossing here looks like a length 2-repetition, and we define reductions

$$\langle \dots a, \dots, b \dots, a, \dots, b, \dots \rangle \longrightarrow \langle \dots a, \dots, b \dots, b, \dots, a, \dots \rangle \quad \text{for} \quad a < b. \tag{4}$$

We illustrate a rewrite step on partitions and on the corresponding incremental lists:



It is easy to see that every uncrossing reduction preserves parents as well as block sizes, so it preserves parent multisets: If $P \to P'$ then $ms(P) = ms(P')$.

▶ **Proposition 13.** *The rewriting system of Definition 12 is strongly terminating, that is every list of reductions $P_0 \to P_1 \to \dots$ is finite and terminates with a non-crossing partition P^*.*

Proof. This is elegantly expressed in terms of incremental lists (4): every reduction $\vec{s_i} \to \vec{s_{i+1}}$ is a strict up-step in the lexicographic order on lists, *i.e.* satisfies $\vec{s_i} < \vec{s_{i+1}}$. Because the set $IL(K)$ is finite, there can only be finitely many of such steps. ◀

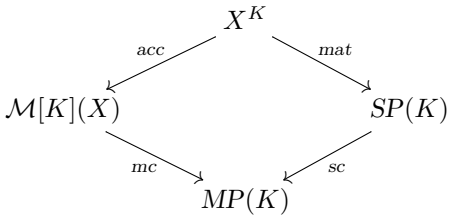
▶ **Corollary 14.** *The reduction system of Definition 12 has the Church-Rosser property (i.e. is confluent).* ⌟

This follows abstractly from strong termination and uniqueness of normal forms. We conjecture that (local) confluence can also be established directly, in the following form: For all P, if $Q_1 \leftarrow P \to Q_2$ then there exists an R and lists of reductions *of length at most 2* with $Q_1 \to^* R \,{}^*\!\leftarrow Q_2$.

As an aside, note that we can apply the reduction rule (4) not just to incremental lists but also to arbitrary lists of natural numbers $\mathbb{N}^*$. This extended rewriting system is still strongly terminating, but no longer confluent. For example, we have reductions $\langle 1,3,2,3,2,1\rangle \;{}^*\!\leftarrow \langle 1,3,2,1,3,2\rangle \to \langle 1,3,2,2,3,1\rangle$ where the left and right lists are both irreducible.

8 A wider picture: adding multiset partitions

Having presented our main results, we step back and put things in a wider perspective. We started in Diagram (1) with lists, multisets and set partitions. The two legs in this diagram can be completed to a diamond of the form:



(5)

The set $MP(K)$ contains the *multiset partitions* with total K. It is defined as:

$$MP(K) \coloneqq \big\{\sigma \in \mathcal{M}\big(\underline{K}\big) \;\big|\; \textstyle\sum_i \sigma(i)\cdot i = K\big\}.$$

Multiset partitions represent unlabelled partitions where the underlying elements of X are not distinguishable anymore. Such multiset partitions are commonly considered in number theory. The sizes $\big|MP(K)\big|$, for $K \geq 1$ are given by the partition function $p(K)$ with values $1,2,3,5,7,11,15,22,30,42,56,\ldots$. For example $p(4) = 5$ counts the 5 number-theoretic partitions of the number 4,

$$1+1+1+1=4 \qquad 1+1+2=4 \qquad 1+3=4 \qquad 2+2=4 \qquad 4=4.$$

Alternatively, in multiset notation, we get the elements of $MP(4)$, namely:

$$4|1\rangle \qquad 2|1\rangle + 1|2\rangle \qquad 1|1\rangle + 1|3\rangle \qquad 2|2\rangle \qquad 1|4\rangle.$$

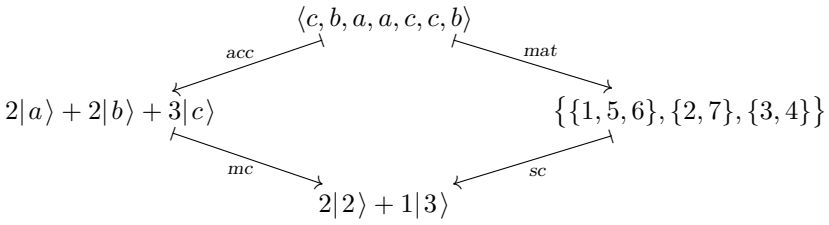
The function $mc\colon \mathcal{M}[K](X) \to MP(K)$ is defined in [10] and is called *multiplicity count*. It counts the multiplicities in a multiset $\varphi \in \mathcal{M}[K](X)$ via:

$$mc(\varphi) \coloneqq \sum_{x \in supp(\varphi)} 1\big|\,\varphi(x)\,\big\rangle.$$

The fourth function $sc\colon SP(K) \to MP(K)$ in (5), from set partitions to multiset partitions, will be called *size count*. It keeps track of the sizes of blocks in a set partition:

$$sc(P) \coloneqq \sum_{B \in P} 1\big|\,|\,B\,|\,\big\rangle.$$

The following instantiation illustrates how the operations in Diagram (5) work.



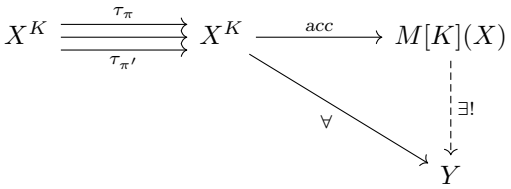
It is not hard to show in general that Diagram (5) commutes.

Set and multiset partitions are studied in mathematical biology [5, 4], to capture mutations, and more recently also in clustering in machine learning, to handle possible extension of the numbers of clusters, see *e.g.* [3]. Here we study the underlying datatypes and their relations (5).

Finally, we remark that the diamond (5) is a *weak pullback*. That is for every pair (φ, P) of a multiset and a partition which induce the same multiset partition $mc(\varphi) = sc(P)$, we can find some list $\vec{x}$ with $\varphi = acc(\vec{x})$ and $P = mat(\vec{x})$. For example, the multiset $\varphi = 3|a\rangle + 2|b\rangle + 2|c\rangle$ and the partition $P = \{\{1,3,4\},\{2,6\},\{5,7\}\}$ induce the same multiset partition $1|3\rangle + 2|2\rangle \in MP(7)$. They are themselves induced by the list $\vec{x} = \langle a, b, a, a, c, b, c\rangle$. This list $\vec{x}$ is not unique however; in our example, $\vec{x} = \langle a, c, a, a, b, c, b\rangle$ also works.

9 An even wider categorical picture

In this section, we give a more high-level, structural view on the diamond (5), returning to general finite sets X, Y. Category theory is an abstract language of structure and datatypes. The idea of forgetting information can be formalized using *coequalizers*, an abstract form of quotient. In [9] it is shown that the accumulation map is the coequaliser of all transposition maps:



Any bijection $\pi\colon K \xrightarrow{\cong} K$ induces a reordering (transposition) map $\tau_\pi\colon X^K \to X^K$. Accumulation is invariant under all those transpositions, and furthermore universal with that property: any map $f\colon X^K \to Y$ which is invariant under all transpositions factors uniquely through acc. This can be understood as a proof principle about multisets.

We wish to understand partitions $SP(X)$ in the same way. An immediate obstacle is that unlike multisets, set partitions do not push forward under arbitrary functions $X \to Y$ in an obvious way. They do however push forward under *bijections* $X \to Y$. This naturally leads us to consider *combinatorial species*, which are functors $Bij \to Set$ where Bij is the category of finite sets X, Y and bijections between them. Since their invention [11], species have been a staple in combining combinatorial and categorical reasoning. Using their framework, we can precisely state the process of forgetting information by quotienting out the action of permutations.

▶ **Definition 15** (Anonymisation). *Let $\mathbb{C}$ be any category, and write $\mathbb{N}$ for the discrete category whose objects are the natural numbers. If $\mathcal{F}\colon Bij \times \mathbb{C} \to Set$ is a functor, we define a functor $\mathcal{F}_1\colon \mathbb{N} \times \mathbb{C} \to Set$ as the coequalizer*

$$\mathcal{F}(\underline{K}, C) \underset{\mathcal{F}(\pi', id_C)}{\overset{\mathcal{F}(\pi, id_C)}{\rightrightarrows}} \mathcal{F}(\underline{K}, C) \longrightarrow \mathcal{F}_1(K, C) \tag{6}$$

More concretely, the set $\mathcal{F}(\underline{K}, C)$ admits an action of the group of bijections $Aut(\underline{K})$ and we let $\mathcal{F}_1(K, C) = \mathcal{F}(\underline{K}, C)/Aut(\underline{K})$. This canonical construction is already present in the first section of [11]. We will also write $\mathcal{F}_1(K, C) = \int^K \mathcal{F}(X, C)\mathrm{d}X$ in analogy with coend calculus or the category of elements.

▶ **Proposition 16.** *Let $\mathcal{H}\colon Bij \times Bij \to Set$ be a functor in two variables. Then we can anonymise variables in different orders and obtain the same result*

$$\int^n \int^m \mathcal{H}(X,Y)\,\mathrm{d}X\,\mathrm{d}Y \cong \int^m \int^n \mathcal{H}(X,Y)\,\mathrm{d}Y\,\mathrm{d}X.$$ ⌟

For the formally minded, the cardinality functor $|-|\colon Bij \to \mathbb{N}$ induces a functor $\Delta\colon [\mathbb{N}, Set^{\mathbb{C}}] \to [Bij, Set^{\mathbb{C}}]$ between functor categories by precomposition; it has a left adjoint which is given by "anonymisation". The unit of this adjunction is a natural transformation $\mathcal{F}(X,C) \to \mathcal{F}_1(|X|,C)$. Thus any functor $\mathcal{H}\colon Bij \times Bij \to Set$ gives rise to a commuting diamond of natural transformations

$$\begin{array}{ccccc} & & \mathcal{H}(X,Y) & & \\ & \swarrow & & \searrow & \\ \int^{|X|} \mathcal{H}(X,Y)\mathrm{d}X & & & & \int^{|Y|} \mathcal{H}(X,Y)\mathrm{d}X \\ & \searrow & & \swarrow & \\ & & \int^{|Y|}\int^{|X|} \mathcal{H}(X,Y)\mathrm{d}X\mathrm{d}Y & & \end{array} \tag{7}$$

We can now describe the diamond (5) by invoking Proposition 16 on the function space construction Y^X.

▶ **Proposition 17.** *Let $\mathcal{H}\colon Bij \times Bij \to Set$ be given by $\mathcal{H}(X,Y) = Y^X$ with functorial action $\mathcal{H}(\alpha,\beta)(f) = \beta \circ f \circ \alpha^{-1}$. Then*

$$\int^K Y^X \mathrm{d}X \cong \mathcal{M}[K](Y)$$

is multisets with exactly K elements, and

$$\int^N Y^X \mathrm{d}Y \cong SP[N](X)$$

is set partitions with at most N blocks, i.e. $SP[N](X) = \{P \in SP(X) \mid |P| \leq N\}$. ⌟

Furthermore, if we define $MP[n,k]$ as multiset partitions of k with at most n blocks, we obtain the following improved categorical diamond.

$$\begin{array}{ccccc} & & Y^X & & \\ & \overset{acc}{\swarrow} & & \overset{mat}{\searrow} & \\ \mathcal{M}\big[|X|\big](Y) & & & & SP\big[|Y|\big](X) \\ & \underset{mc}{\searrow} & & \underset{sc}{\swarrow} & \\ & & MP\big[|Y|,|X|\big] & & \end{array} \tag{8}$$

All maps are coequalizers and natural in X, Y. This makes it clear that multisets and partitions arise from a fully symmetric situation, where we forget information along two independent axis: order (transposition $X \cong X$) and identity (substitution $Y \cong Y$).

10 Conclusion

In this paper we studied two orthogonal operations on lists from a fundamental perspective, namely: (1) accumulation of lists to multisets, which is stable under transposition, and (2) matching of lists to set partitions, which is stable under substitution. In subsequent work we wish to include the various distributions on these datatypes in the same perspective.

References

1 D. Aldous. Exchangeability and related topics. In P. Hennequin, editor, *École d'Été de Probabilités de Saint-Flour XIII – 1983*, number 1117 in Lect. Notes Math., pages 1–198. Springer, Berlin, 1985. doi:10.1007/BFb0099421.

2 M. Bona. *Handbook of Enumerative Combinatorics*. Discrete Mathematics and Its Applications. CRC Press, 2015.

3 T. Broderick, J. Pitman, and M. Jordan. Feature allocations, probability functions, and paintboxes. *Bayesian Analysis*, 8:801–836, 2013. doi:10.1214/13-BA823.

4 H. Crane. The ubiquitous Ewens sampling formula. *Statistical Science*, 31(1):1–19, 2016. doi:10.1214/15-STS529.

5 W. Ewens. The sampling theory of selectively neutral alleles. *Theoret. Population Biology*, 3:87–112, 1972. doi:10.1016/0040-5809(72)90035-4.

6 Z. Galil and G. Italiano. Data structures and algorithms for disjoint set union problems. *ACM Comput. Surv.*, 23:319–344, September 1991. doi:10.1145/116873.116878.

7 F. Hoppe. Pólya-like urns and the Ewens' sampling formula. *Journ. Math. Biology*, 20:91–94, 1984. doi:10.1007/BF00275863.

8 B. Jacobs. From multisets over distributions to distributions over multisets. In *Logic in Computer Science*. IEEE, Computer Science Press, 2021. doi:10.1109/lics52264.2021.9470678.

9 B. Jacobs. Multinomial and hypergeometric distributions in Markov categories. In A. Sokolova, editor, *Math. Found. of Programming Semantics*, number 351 in Elect. Proc. in Theor. Comp. Sci., pages 98–115, 2021. doi:10.4204/EPTCS.351.7.

10 B. Jacobs. Sufficient statistics and split idempotents in discrete probability theory. In *Math. Found. of Programming Semantics*, 2022.

11 A. Joyal. Une théorie combinatoire des séries formelles. *Advances in Mathematics*, 42(1):1–82, 1981. doi:10.1016/0001-8708(81)90052-9.

12 G. Kreweras. Sur les partitions non croisées d'un cycle. *Discrete Math.*, 1(4):333–350, 1972.

13 R. Stanley. Parking functions and noncrossing partitions. *Elect. Jour. of Combinatorics*, 2(4), 1997. doi:10.37236/1335.

14 R. Stanley and S. Fomin. *Enumerative Combinatorics*, volume 2 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1999. doi:10.1017/CBO9780511609589.

15 S. Tavaré. The magical Ewens sampling formula. *Bull. London Math. Soc.*, 53:1563–1582, 2021. doi:10.1112/blms.12537.

16 E. Weisstein. Restricted growth string. MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/RestrictedGrowthString.html, July 2022.

Evaluation Trade-Offs for Acyclic Conjunctive Queries

Ahmet Kara
Universität Zürich, Switzerland

Milos Nikolic
University of Edinburgh, UK

Dan Olteanu
Universität Zürich, Switzerland

Haozhe Zhang
Universität Zürich, Switzerland

Abstract

We consider the evaluation of acyclic conjunctive queries, where the evaluation time is decomposed into preprocessing time and enumeration delay. In a seminal paper at CSL'07, Bagan, Durand, and Grandjean showed that acyclic queries can be evaluated with linear preprocessing time and linear enumeration delay. If the query is free-connex, the enumeration delay becomes constant. Further prior work showed that constant enumeration delay can be achieved for arbitrary acyclic conjunctive queries at the expense of a preprocessing time that is characterised by the fractional hypertree width.

We introduce an approach that exposes a trade-off between preprocessing time and enumeration delay for acyclic conjunctive queries. The aforementioned prior works represent extremes in this trade-off space. Yet our approach also allows for the enumeration delay and the preprocessing time between these extremes, in particular the delay may lie between constant and linear time.

Our approach decomposes the given query into subqueries and achieves for each subquery a trade-off that depends on a parameter controlling the times for preprocessing and enumeration. The complexity of the query is given by the Pareto optimal points of a bi-objective optimisation program whose inputs are possible query decompositions and parameter values.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory); Information systems → Database views

Keywords and phrases acyclic queries, query evaluation, enumeration delay

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.29

Funding This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 682588.

1 Introduction

The problem of query evaluation is central to databases, e.g., [19, 16, 12, 15]. Over the past five decades, extensive work has been conducted on finding algorithms for conjunctive query evaluation with increasingly lower computational complexity. This complexity is governed by notions of width, such as the treewidth [17], hypertree width [10, 13, 16], fractional edge cover number [2], and submodular width [14, 12]. Yet, this coarse analysis puts on the same par intrinsically hard queries and easy ones with the same asymptotic output size.

A finer analysis decomposes the complexity into preprocessing time and enumeration delay. The preprocessing time is the time to build a data structure that represents succinctly the query result. The enumeration delay is the maximum of three times: the time to output the first tuple in the query result, the time between outputting any two consecutive result tuples, and the time between the last result tuple and the end of the enumeration process [8].

▶ **Example 1.** The query $P(A, B) = R(A), S(B)$ computes the Cartesian product of the unary relations R and S. The size of the query result is quadratic in the size of the relations, so any join algorithm needs at least quadratic time to compute this result. Yet the tuples in the query result can be enumerated with constant delay directly from the input relations.

The query $J(A, B) = R(A, C), S(B, C)$ computes the natural join of the binary relations R and S and projects away the join column B. It is conjectured that it is not possible to evaluate this query with constant enumeration delay after linear preprocessing time [3]. The query can be evaluated, however, with linear delay after no (or at most linear) preprocessing time [3] or with constant delay after quadratic preprocessing time [16].

Even though both queries P and J can have results of quadratic size, they differ in the amount of preprocessing required to guarantee constant-delay enumeration: Whereas P does not need preprocessing, J requires to compute the result in the preprocessing step. ⌟

There is a solid body of work on the trade-off between preprocessing time and enumeration delay for conjunctive queries. Any conjunctive query can be evaluated with $\mathcal{O}(N^{\mathsf{w}})$ preprocessing time and constant enumeration delay [16], where N is the size of the input database and w is the width of the query; w generalises the fractional hypertree width [13] from Boolean to conjunctive queries with arbitrary free variables. Any acyclic conjunctive query admits linear preprocessing time and linear enumeration delay [3]. If the acyclic query is free-connex, the enumeration delay becomes constant; otherwise, the query cannot be evaluated with linear preprocessing time and constant enumeration delay, assuming the Boolean Matrix Multiplication conjecture [3]. A subclass of acyclic queries, called hierarchical, can be evaluated with $\mathcal{O}(N^{1+(\mathsf{w}-1)\epsilon})$ preprocessing time and $\mathcal{O}(N^{1-\epsilon})$ enumeration delay for any $\epsilon \in [0, 1]$ [11]. In this trade-off space, the preprocessing time ranges from $O(N)$ to $\mathcal{O}(N^{\mathsf{w}})$, while the enumeration delay ranges from $O(N)$ to constant. Conjunctive queries of bounded free-connex submodular width admit constant (in the database size, albeit non-polynomial in the query size) delay after a fixed-parameter tractable preprocessing step [5].

In this paper, we introduce an approach that defines a preprocessing-enumeration trade-off for *any* acyclic conjunctive query. Prior works are points in this trade-off space. Our approach can achieve lower query evaluation time than prior works in case only a fraction of the result is needed. For this, we pick one point in the trade-off space that defines the preprocessing time and the enumeration delay so that the overall computation time is the minimum across the entire space.

Our approach works as follows. Consider an acyclic query Q with free variables $\mathcal{F}$ and a partial order ω of the variables of Q, such that the free variables come before the bound variables in ω [16]. We decompose Q into queries that are *induced* by the free variables: For each free variable $X \in \mathcal{F}$, we construct an induced query Q_X, whose body is the join of all relations in Q subject to further simplifications by semi-join reductions and projections. The free variables of Q_X are X and the variables that come before X in ω and on which X depends (Section 2 defines dependent variables and Section 3 defines induced queries).

To enumerate the tuples in the result of Q, we use a chain of calls to the enumeration procedures for the induced queries in the order of the free variables $X_1, \ldots, X_n$ in ω. We iterate over the result of Q_{X_1} to enumerate the distinct values of X_1. We then iterate over the result of Q_{X_2} to enumerate the distinct values of X_2 *given* a value for X_1. In general, we iterate over the result of Q_{X_i} to enumerate the distinct values of X_i given values for X_1 to X_{i-1}, as fixed by the iterators for Q_{X_1} to $Q_{X_{i-1}}$. Once a tuple of values for all free variables is obtained, we backtrack.

For each induced query Q_X, we design a preprocessing-enumeration trade-off controlled by a parameter $\epsilon_X \in [0, 1]$ specific to Q_X. A low value for ϵ_X means low, down to linear preprocessing time at the expense of a high, up to linear delay. A high value for ϵ_X means

high preprocessing time to achieve a low, down to constant delay. To achieve a trade-off depending on ϵ_X, we partition each relation on the join attributes such that the light parts have join values whose frequencies are below a threshold defined by ϵ_X while the heavy parts have join values with high frequencies. In the preprocessing phase, we then compute the query over the light parts of the relations. In the enumeration phase, we enumerate the tuples from the join of the parts where at least one part is heavy. The computation times for the two phases are given by $\mathcal{O}(N^{p(Q_X,\epsilon_X)})$ for preprocessing and $\mathcal{O}(N^{e(Q_X,\epsilon_X)})$ for enumeration delay, where $p(Q_X,\epsilon_X)$ and $e(Q_X,\epsilon_X)$ are called the preprocessing cost and enumeration cost, respectively. We can use the parameters for the induced queries to define a trade-off for the original query Q. To find a desired trade-off, we need to consider the set of possible variable orders for Q, as they may yield different sets of induced queries, as well as possible parameter values for the induced queries. Given a variable order ω for an acyclic query Q and a set $\boldsymbol{\epsilon} = \{\epsilon_X \mid X \in \mathcal{F}\}$ of parameter values for the induced queries, we define the preprocessing and enumeration costs of Q as the maximum preprocessing cost and enumeration cost, respectively, of the induced queries: $p(\omega,\boldsymbol{\epsilon}) = \max_{X\in\mathcal{F}}\ p(Q_X,\epsilon_X)$ and $e(\omega,\boldsymbol{\epsilon}) = \max_{X\in\mathcal{F}}\ e(Q_X,\epsilon_X)$.

Given two pairs (p_1,e_1) and (p_2,e_2), $(p_1,e_1) < (p_2,e_2)$ holds if (1) $p_1 \le p_2$, (2) $e_1 \le e_2$, and (3) at least one of the two inequalities is strict. A pair of preprocessing cost p and enumeration cost e is *Pareto optimal* if there are no trade-off parameters $\boldsymbol{\epsilon}$ and variable order ω such that $(p(\omega,\boldsymbol{\epsilon}), e(\omega,\boldsymbol{\epsilon})) < (p,e)$.

The Pareto optimal pairs of objective values of the following bi-objective *trade-off program* are possible preprocessing-enumeration costs for an acyclic query [1] Q with free variables $\mathcal{F}$:

$$\begin{aligned} \text{minimise} \quad & \left(\max_{X\in\mathcal{F}}\ p(Q_X,\epsilon_X)\ ,\ \max_{X\in\mathcal{F}}\ e(Q_X,\epsilon_X) \right) \\ \text{subject to} \quad & \omega \text{ is a variable order for } Q \text{ and} \\ & Q_X \text{ is induced by } X \text{ wrt. } \omega,\ \forall X \in \mathcal{F} \text{ and} \\ & \epsilon_X \in [0,1],\ \forall X \in \mathcal{F} \end{aligned}$$

The minimisation of the program objective is based on the inequality ($<$) defined above. We denote the set of Pareto optimal pairs of objective values of this program by $\mu(Q)$. The complexity of our evaluation algorithm is given by any of these Pareto optimal values.

▶ **Theorem 2.** *Any acyclic conjunctive query Q can be evaluated over a database of size N with $\mathcal{O}(N^p)$ preprocessing time and $\mathcal{O}(N^e)$ enumeration delay, where $(p,e) \in \mu(Q)$.*

The exponents p and e are explained in detail in Sections 4.1 and 5.2. Our approach recovers prior works as corollaries, as they represent possible Pareto optimal pairs in $\mu(Q)$. It recovers the case of free-connex queries [3]:

▶ **Corollary 3.** *Any free-connex acyclic conjunctive query can be evaluated over a database of size N with $\mathcal{O}(N)$ preprocessing time and $\mathcal{O}(1)$ enumeration delay.*

This is equivalent to stating that the pair $(1,0)$ of preprocessing and enumeration costs is in the set $\mu(Q)$ of Pareto optimal pairs of objective values of the trade-off program.

Our approach also recovers the more general result for arbitrary acyclic queries [3]:

[1] In this work we focus on acyclic queries with at least one free variable; in case of no free variables, there are no induced queries and the optimisation program is not well-defined. If an acyclic query has no free variables, it can be evaluated in linear time and the enumeration delay is trivially constant [3].

▶ **Corollary 4.** *Any acyclic conjunctive query can be evaluated over a database of size N with $\mathcal{O}(N)$ preprocessing time and $\mathcal{O}(N)$ enumeration delay.*

This means that $(1, e) \in \mu(Q)$ for $e \leq 1$. Next, our approach recovers the result on constant enumeration delay from prior work on factorised representations for query results, when restricted to acyclic queries [16]:

▶ **Corollary 5.** *Any acyclic conjunctive query with fractional hypertree width w can be evaluated over a database of size N with $\mathcal{O}(N^{\mathsf{w}})$ preprocessing time and $\mathcal{O}(1)$ enumeration delay.*

This corollary is implied by the fact that $(\mathsf{w}, 0) \in \mu(Q)$. Finally, our approach recovers the trade-off for hierarchical queries [11]:

▶ **Corollary 6.** *Any hierarchical query with fractional hypertree width w can be evaluated over a database of size N with $\mathcal{O}(N^{1+(\mathsf{w}-1)\epsilon})$ preprocessing time and $\mathcal{O}(N^{1-\epsilon})$ enumeration delay for any $\epsilon \in [0, 1]$.*

We obtain the above result by showing that $(p, e) \in \mu(Q)$ with $p \leq 1 + (\mathsf{w} - 1)\epsilon$ and $e \leq 1 - \epsilon$, for any $\epsilon \in [0, 1]$. In contrast to prior work [11], our approach uses different, more general evaluation strategies to account for the generality of acyclic queries. This generality comes at a price: Our trade-off does not have a closed-form expression for the computational complexity of preprocessing and enumeration as for hierarchical queries.

In this paper we use one approach to evaluate all induced queries. Our framework is however permissive and allows plugging in different evaluation strategies for different induced queries, e.g., evaluation strategies tailored specifically at path and star queries [7].

The structure of the paper is as follows. Section 2 introduces basic notions and tools. Section 3 explains our query decomposition technique. Sections 4 and 5 detail the preprocessing and enumeration phases. Section 6 compares our approach against two mainstream approaches by means of examples. Section 7 concludes with future work. Proofs of formal statements are deferred to Appendix A.

2 Preliminaries

Data Model

A *schema* $\mathcal{X} = (X_1, \ldots, X_n)$ is a tuple of distinct variables. Each variable X_i has a discrete domain $\mathsf{Dom}(X_i)$. We treat schemas and sets of variables interchangeably, assuming a fixed ordering of variables. A tuple $\mathbf{x}$ over schema $\mathcal{X}$ is an element from $\mathsf{Dom}(\mathcal{X}) = \mathsf{Dom}(X_1) \times \cdots \times \mathsf{Dom}(X_n)$.

A *relation* R over schema $\mathcal{X}$ is a set of tuples over the same schema. The size of R is given by the number of tuples in R. A database is a set of relations and has size given by the sum of the sizes of its relations.

Given a tuple $\mathbf{x}$ over schema $\mathcal{X}$ and $\mathcal{S} \subseteq \mathcal{X}$, $\mathbf{x}[\mathcal{S}]$ is the restriction of $\mathbf{x}$ onto $\mathcal{S}$. For a relation R over $\mathcal{X}$, a schema $\mathcal{S} \subseteq \mathcal{X}$, and tuple $\mathbf{s} \in \mathsf{Dom}(\mathcal{S})$: $\sigma_{\mathcal{S}=\mathbf{s}} R = \{\, \mathbf{x} \mid \mathbf{x} \in R \wedge \mathbf{x}[\mathcal{S}] = \mathbf{s} \,\}$ is the set of tuples in R that agree with $\mathbf{s}$ on the variables in $\mathcal{S}$; $\pi_{\mathcal{S}} R = \{\, \mathbf{x}[\mathcal{S}] \mid \mathbf{x} \in R \,\}$ is the set of restrictions of the tuples in R to the variables in $\mathcal{S}$.

Conjunctive Queries

A *conjunctive query* (CQ) is of the form

$$Q(\mathcal{F}) = R_1(\mathcal{X}_1), \ldots, R_n(\mathcal{X}_n).$$

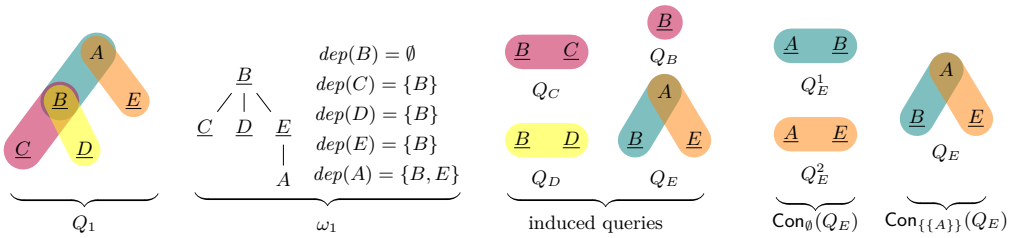


Figure 1 Left to right: Hypergraph of the query Q_1 from Example 7; variable order ω_1 for Q_1; the queries induced by the free variables in ω_1; the $\emptyset$- and $\{\{A\}\}$-connected components of Q_E.

We denote by: $(R_i)_{i\in[n]}$ the relation symbols; $(R_i(\mathcal{X}_i))_{i\in[n]}$ the atoms; $vars(Q) = \bigcup_{i\in[n]} \mathcal{X}_i$ the set of variables; $free(Q) = \mathcal{F} \subseteq vars(Q)$ the set of free variables; and $atoms(Q) = \{R_i(\mathcal{X}_i) \mid i \in [n]\}$ the set of atoms in Q. The variables in $vars(Q) \setminus \mathcal{F}$ are called *bound*. Given a set $\mathcal{R}$ of atoms, we denote $vars(\mathcal{R}) = \bigcup_{R(\mathcal{X})\in\mathcal{R}} \mathcal{X}$.

The hypergraph of Q is a multi-hypergraph $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of variables in Q and $\mathcal{E}$ contains a hyperedge over $\mathcal{X}$ for each atom $R(\mathcal{X})$ in Q. In the graphical representation of hypergraphs, we underline the free variables.

A set $\mathcal{V} \subseteq vars(Q)$ is called a *join v-set* if there are two distinct atoms $R_i(\mathcal{X}_i)$ and $R_j(\mathcal{X}_j)$ in Q such that $\mathcal{X}_i \cap \mathcal{X}_j = \mathcal{V}$. We denote the set of join v-sets in Q by $\mathsf{JVSets}(Q)$.

A CQ is *acyclic* (ACQ in short) if it admits a *join tree* where each node is an atom and if any two nodes have variables in common, then all nodes along the path between them also have these variables [19]. The query is *free-connex* acyclic if it is acyclic and stays acyclic after adding a fresh atom whose schema consists of the free variables of the query [6]. A query is *hierarchical* if for any two of its variables, either their sets of atoms are disjoint or one is contained in the other [18].

Variable Orders

Given a CQ Q, two variables *depend* on each other if they occur in the same atom of the query. A *variable order* ω for Q is a pair (T, dep) [16] where:

- T is a rooted forest with one node per variable in Q. The variables of each atom in Q lie along the same root-to-leaf path in T. No bound variable is an ancestor of a free variable.
- The function dep maps each variable X to the subset of its ancestor variables in T on which the variables in the subtree rooted at X depend.

This type of variable orders were first introduced as d-tree extensions [16] and later called free-top variable orders [11]. We denote the set of variable orders of Q by $\mathsf{VO}(Q)$. Using the language of hypertree decompositions, we call the set $\{X\} \cup dep(X)$ the *bag* of ω at X. In the graphical representation of variable orders, we underline the free variables.

▶ **Example 7.** Consider the query $Q_1(B, C, D, E) = R(A, B), S(B, C), T(B, D), U(A, E)$. Figure 1 shows its hypergraph (left) and next to it a variable order ω_1 for the query. Consider now the variant $Q_2(C, D, E) = R(A, B), S(B, C), T(B, D), U(A, E)$ of Q_1 where the variable B is bound. Its hypergraph is given in Figure 2 (left). The variable order ω_1 from Figure 1 is not valid for Q_2, since the bound variable B sits on top of the free variables C, D, and E. The variable order ω_2 in Figure 2 is a valid variable order for Q_2. ⌟

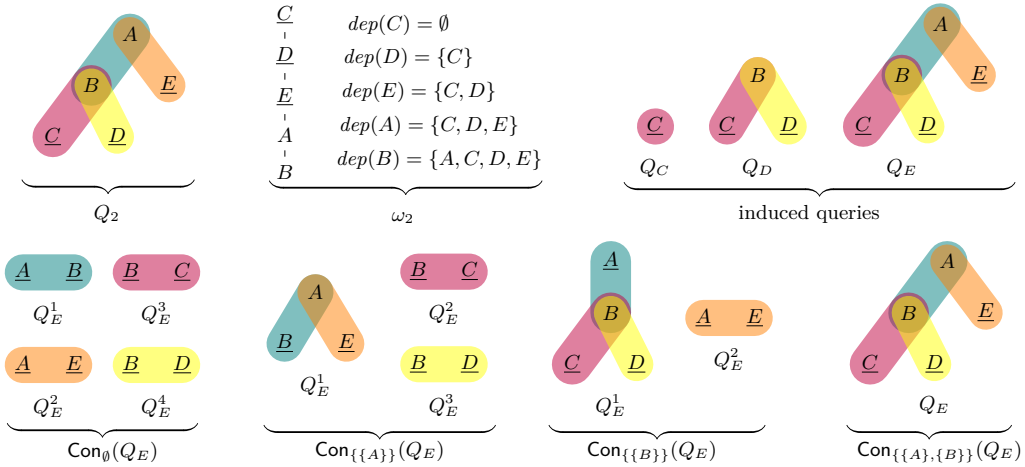


Figure 2 First row, left to right: Hypergraph of the query Q_2 from Example 7; a variable order ω_2 for Q_2; the queries induced by the free variables in ω_2. Second row, left to right: The $\emptyset$-, $\{\{A\}\}$-, $\{\{B\}\}$-, and $\{\{A\},\{B\}\}$-connected components of Q_E.

Width Measures

Given a CQ Q and a set $\mathcal{V}$ of variables from Q, a *fractional edge cover* of $\mathcal{V}$ is a solution $\boldsymbol{\lambda} = (\lambda_{R(\mathcal{X})})_{R(\mathcal{X}) \in atoms(Q)}$ to the following linear program [2]:

$$\begin{aligned} \text{minimise} \quad & \sum_{R(\mathcal{X}) \in\, atoms(Q)} \lambda_{R(\mathcal{X})} \\ \text{subject to} \quad & \sum_{R(\mathcal{X}):\, X \in \mathcal{X}} \lambda_{R(\mathcal{X})} \geq 1 && \text{for all } X \in \mathcal{F} \text{ and} \\ & \lambda_{R(\mathcal{X})} \in [0,1] && \text{for all } R(\mathcal{X}) \in atoms(Q) \end{aligned}$$

The optimal objective value of the above program is called the *fractional edge cover number* of $\mathcal{V}$ and denoted as $\rho^*_Q(\mathcal{V})$. We sometimes represent Q by the set $\mathcal{E} = \{\mathcal{X} \mid \exists R(\mathcal{X}) \in atoms(Q)\}$ of the schemas of its atoms and write $\rho^*_{\mathcal{E}}(\mathcal{V})$. If Q and $\mathcal{E}$ are clear from the context, we skip them in the expressions.

We define the *width* $\mathsf{w}(Q)$ of Q as in prior work [11]:

$$\mathsf{w}(Q) = \min_{\omega \in \mathsf{VO}(Q)} \mathsf{w}(\omega), \text{ where}$$
$$\mathsf{w}(\omega) = \max_{X \in vars(Q)} \rho^*_Q(\{X\} \cup dep(X))$$

The measure w is equivalent to the fractional hypertree width when extended from Boolean to non-Boolean queries [16].

▶ **Example 8.** Consider the variable order ω_1 for the query Q_1 in Figure 1. The bag at variable A consists of the variables A, B, and E. It holds $\rho^*(\{A, B, E\}) = 2$, since we need two atoms to cover the variables in the bag. Similarly, for the bag $\{B, E\}$ at E, we have $\rho^*(\{B, E\}) = 2$. Each of the other bags $\mathcal{B}$ of the variable order can be covered by a single atom, therefore it holds $\rho^*(\mathcal{B}) = 1$. Hence, $\mathsf{w}(\omega_1) = 2$, which implies $\mathsf{w}(Q_1) \leq 2$. By checking other possible variable orders for Q_1, one can see that $\mathsf{w}(Q_1)$ equals 2.

Now, consider the variable order ω_2 for the query Q_2 in Figure 2. The bag at E consists of C, D, and E. We have $\rho^*(\{C, D, E\}) = 3$. It holds $\mathsf{w}(Q_2) = \mathsf{w}(\omega_2) = 3$. ⌟

Partitioning

We partition relations based on the frequencies of their values. For a database $\mathcal{D}$, relation $R \in \mathcal{D}$ over schema $\mathcal{X}$, schema $\mathcal{S} \subset \mathcal{X}$, and threshold θ, the pair $(R^{\mathcal{S}\to H}, R^{\mathcal{S}\to L})$ is a *partition* of R on $\mathcal{S}$ with threshold θ if:

$$\text{(light part)} \quad R^{\mathcal{S}\to L} = \{\mathbf{x} \in R \mid \forall K \in \mathcal{D}, |\sigma_{\mathcal{S}=\mathbf{x}[\mathcal{S}]} K| < \theta\}$$
$$\text{(heavy part)} \quad R^{\mathcal{S}\to H} = R \setminus R^{\mathcal{S}\to L}$$

The relations $R^{\mathcal{S}\to H}$ and $R^{\mathcal{S}\to L}$ are called *heavy* and respectively *light* on the partition key $\mathcal{S}$. For $|\mathcal{D}| = N$ and a partition $(R^{\mathcal{S}\to H}, R^{\mathcal{S}\to L})$ of R on $\mathcal{S}$ with threshold $\theta = N^\epsilon$ for $\epsilon \in [0,1]$, we have: (1) $\forall \mathbf{t} \in \pi_{\mathcal{S}} R^{\mathcal{S}\to L} : |\sigma_{\mathcal{S}=\mathbf{t}} R^{\mathcal{S}\to L}| < \theta = N^\epsilon$; and (2) $|\pi_{\mathcal{S}} R^{\mathcal{S}\to H}| \leq \frac{N}{\theta} = N^{1-\epsilon}$. The first bound follows from the light part condition. The second bound holds because each tuple in $\pi_{\mathcal{S}} R^{\mathcal{S}\to H}$ is paired with at least θ distinct tuples in at least one relation in the database. The database can contain at most $\frac{N}{\theta}$ such tuples over $\mathcal{S}$; otherwise, the database size exceeds N.

Given schemas $\mathcal{S}_1, \ldots, \mathcal{S}_n \subset \mathcal{X}$, an HL-signature *sig* for R has the form $\{\mathcal{S}_1 \to s_1, \ldots, \mathcal{S}_n \to s_n\}$, where $s_i \in \{H, L\}$ for $i \in [n]$. The relation part R^{sig} is defined as $\bigcap_{i\in[n]} R^{\mathcal{S}_i \to s_i}$.

Computational Model

We consider the RAM model of computation. Each relation R over schema $\mathcal{X}$ is implemented by a data structure that stores the tuples in R using $O(|R|)$ space. This data structure can: (1) look up tuples in constant time; (2) enumerate all tuples in R with constant delay; and (3) report $|R|$ in constant time. For a schema $\mathcal{S} \subset \mathcal{X}$, we use an index data structure that for any $\mathbf{s} \in \mathsf{Dom}(\mathcal{S})$ can: (4) enumerate all tuples in $\sigma_{\mathcal{S}=\mathbf{s}} R$ with constant delay; (5) check $\mathbf{s} \in \pi_{\mathcal{S}} R$ in constant time; and (6) return $|\sigma_{\mathcal{S}=\mathbf{s}} R|$ in constant time. Our complexity results are based on data complexity where the input query is assumed to be fixed.

3 Query Decomposition and Connected Components

Our approach decomposes queries following variable orders. Each query in such a decomposition is then split further into subqueries that are connected via specific join v-sets. In the following we introduce both concepts.

3.1 Decomposing A Query into Induced Queries

Given a CQ $Q(\mathcal{F})$, a variable order ω for Q, and $X \in \mathcal{F}$, we define the query $Q_X(\mathcal{F}_X)$ induced by X wrt. ω as follows. The free variables of Q_X are $\mathcal{F}_X = \{X\} \cup dep_\omega(X)$. The body of Q_X is $\mathsf{bGYO}(Q, \mathcal{F}_X)$, where the function bGYO applies a variant of the GYO algorithm [20], which we call *bound-GYO* and explain next.

Given a CQ Q and a variable set $\mathcal{F}_X$, bound-GYO replaces atoms in Q by new atoms for fresh relations computed from the input relations. The algorithm repeats the following two rules as long as possible: (1) If a variable $Y \notin \mathcal{F}_X$ only occurs in one atom $R(\mathcal{X})$, replace $R(\mathcal{X})$ by the atom $R'(\mathcal{X} \setminus \{Y\})$, where R' is obtained by projecting away Y from R; (2) if there are two atoms $R(\mathcal{X})$ and $S(\mathcal{Y})$ with $\mathcal{X} \subseteq \mathcal{Y}$, remove the first atom and replace the second atom with a new atom $S'(\mathcal{Y})$ for a new relation S' obtained from S by filtering out the tuples that are not in R (S' is the result of the semi-join reduction of S with R). All new relations in $\mathsf{bGYO}(Q, \mathcal{F}_X)$ can be computed in time linear in the size of the input database.

▶ **Example 9.** Consider the query Q_1 from Example 7. Figure 1 shows the hypergraphs of Q_1 and of the queries Q_B, Q_C, Q_D, and Q_E induced by the free variables of Q_1 wrt. ω_1. We explain the construction of Q_E. We first eliminate the variables C and D by computing $S' = \pi_B S$ and $T' = \pi_B T$. Then, we absorb S' and T' by computing $R' = (R \ltimes S') \ltimes T'$. We obtain $Q_E(B, E) = R'(A, B), U(A, E)$.

Now, consider the query Q_2 from Example 7. Figure 2 shows the hypergraph of Q_2 and of the queries induced by the free variables in ω_2. The query Q_E induced by E is Q_2. The bound-GYO algorithm cannot apply any rule in this case, since all variables in $vars(Q_2) \setminus (\{E\} \cup dep_{\omega_2}(E)) = \{A, B\}$ are join variables and no atom has a schema that subsumes the schema of another atom. ⌟

3.2 Connected Components

Consider a CQ $Q(\mathcal{F})$ and a set $\mathcal{J} \subseteq \mathsf{JVSets}(Q)$ of join v-sets. A $\mathcal{J}$-path between two atoms $R(\mathcal{X})$ and $R'(\mathcal{X}')$ is a sequence $R_1(\mathcal{X}_1), \ldots, R_n(\mathcal{X}_n)$ of atoms such that $R_1(\mathcal{X}_1) = R(\mathcal{X})$, $R_n(\mathcal{X}_n) = R'(\mathcal{X}')$, and $\mathcal{X}_i \cap \mathcal{X}_{i+1} \in \mathcal{J}$ for $i \in [n-1]$. We say that a set $\mathcal{R} \subseteq atoms(Q)$ is *$\mathcal{J}$-connected* if there is a $\mathcal{J}$-path between any two atoms in $\mathcal{R}$. The set $\mathcal{R}$ is *maximally* $\mathcal{J}$-connected if it is $\mathcal{J}$-connected and any proper superset of $\mathcal{R}$ is not. Let $\mathcal{R}_1, \ldots, \mathcal{R}_k$ be the maximally $\mathcal{J}$-connected atom sets of Q. For each $i \in [k]$, consider the query $Q_i(\mathcal{F}_i)$ whose body consists of the atoms in $\mathcal{R}_i$ and whose free variable set $\mathcal{F}_i$ consists of $vars(\mathcal{R}_i) \cap (\mathcal{F} \cup \bigcup_{i \neq j} vars(\mathcal{R}_j))$. We call $Q_i(\mathcal{F}_i)$ a *$\mathcal{J}$-connected component* of Q and denote the set of all $\mathcal{J}$-connected components of Q by $\mathsf{Con}_{\mathcal{J}}(Q)$.

▶ **Example 10.** Consider the query $Q_E(B, E) = R'(A, B), U(A, E)$ depicted in Figure 1. We have $\mathsf{JVSets}(Q_E) = \{\{A\}\}$. Figure 1 shows the $\emptyset$- and the $\{\{A\}\}$-connected components of the query Q_E. The $\emptyset$-connected components of Q_E are the queries $Q_E^1(A, B) = R'(A, B)$ and $Q_E^2(A, E) = U(A, E)$. The only $\{\{A\}\}$-connected component of Q_E is Q_E itself.

Now, consider the query $Q_E(C, D, E) = R(A, B), S(B, C), T(B, D), U(A, E)$ depicted in Figure 2. In this case, we have $\mathsf{JVSets}(Q_E) = \{\{A\}, \{B\}\}$. Figure 2 shows the $\mathcal{J}$-connected components of Q_E for any $\mathcal{J} \subseteq \{\{A\}, \{B\}\}$. The query has four $\emptyset$-connected components. Each of them contains in its body exactly one of the atoms in Q_E. The $\{\{B\}\}$-connected components are $Q_E^1(A, C, D) = R(A, B)$, $S(B, C)$, $T(B, D)$ and $Q_E^2(A, E) = U(A, E)$. The only $\{\{A\}, \{B\}\}$-connected component is Q_E. ⌟

4 Preprocessing

Given an ACQ $Q(\mathcal{F})$, a variable order ω for Q, and a parameter $\epsilon_X \in [0, 1]$ for each free variable X, we construct in the preprocessing stage a compact data structure that represents the result of Q. The data structure consists of the results of *skew-aware queries*, which are obtained from Q by taking the frequencies of values in the input relations into account. We next explain the construction of skew-aware queries.

Given a query Q_X induced by $X \in \mathcal{F}$ wrt. ω and $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, consider the $\mathcal{J}$-connected components $\{Q_1, \ldots, Q_k\}$ of Q_X. For $i \in [k]$, the skew-aware query $S_{\mathcal{J}}(Q_i)$ is obtained from Q_i by replacing each atom $R(\mathcal{X})$ in Q_i with $R^{sig}(\mathcal{X})$, where $sig = \{\mathcal{V} \to L \mid \mathcal{V} \in \mathcal{J} \text{ and } \mathcal{V} \subseteq \mathcal{X}\} \cup \{\mathcal{V} \to H \mid \mathcal{V} \in \mathsf{JVSets}(Q_X) \setminus \mathcal{J} \text{ and } \mathcal{V} \subseteq \mathcal{X}\}$. That is, for any $\mathcal{V} \in \mathsf{JVSets}(Q_X)$ that is contained in schema $\mathcal{X}$ of R, it holds: The relation R^{sig} is light on $\mathcal{V}$ if $\mathcal{V}$ is in $\mathcal{J}$ and heavy otherwise. The skew-aware query $S_{\mathcal{J}}(Q_i)$ has the same free variables as Q_i. Observe that each variable Y that is free in at least two distinct skew-aware queries $S_{\mathcal{J}}(Q_i)$ and $S_{\mathcal{J}}(Q_j)$ must be part of a heavy join v-set. Otherwise, assume that all join

τ(variable order ω, free variables $\mathcal{F}$) : skew-aware queries

1 **let** $s_X = \emptyset, \forall X \in \mathcal{F}$
2 **foreach** $X \in \mathcal{F}$
3 **let** Q_X be the query induced by X wrt. ω
4 **foreach** join v-set $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$
5 **let** $\{Q_1, \ldots, Q_k\}$ be the $\mathcal{J}$-connected components of Q_X
6 $s_X = s_X \cup \{\mathcal{J} \mapsto \{S_\mathcal{J}(Q_1), \ldots, S_\mathcal{J}(Q_k)\}\}$
7 **return** $\{s_X\}_{X \in \mathcal{F}}$

Figure 3 Construction of skew-aware queries from a variable order ω and a variable set $\mathcal{F}$. The skew-aware query $S_\mathcal{J}(Q_i)$ results from Q_i by replacing each relation in Q_i by its part that is light on the join v-sets in $\mathcal{J}$ and heavy on all other join v-sets of the query Q_X induced by X wrt. ω.

v-sets including Y are light. By construction, this means that all such join v-sets are in $\mathcal{J}$. This implies that there is a $\mathcal{J}$-path between the atoms in Q_i and Q_j that include Y. This means that Q_i and Q_j are not maximally $\mathcal{J}$-connected, which contradicts our assumption that Q_i and Q_j are $\mathcal{J}$-connected components of Q_X.

▶ **Example 11.** The only $\{\{A\}\}$-connected component of the query $Q_E(B,E) = R'(A,B), U(A,E)$ in Figure 1 is Q_E itself. The corresponding skew-aware query is $Q_E(B,E) = R'^{A\to L}(A,B), U^{A\to L}(A,E)$.

The $\emptyset$-connected components of $Q_E(C,D,E) = R(A,B), S(B,C), T(B,D), U(A,E)$ in Figure 2 are $Q^1_E(A,B) = R(A,B)$, $Q^2_E(A,E) = U(A,E)$, $Q^3_E(B,C) = S(B,C)$, and $Q^4_E(B,D) = T(B,D)$. We obtain the skew-aware queries $Q^1_E(A,B) = R^{A\to H, B\to H}(A,B)$, $Q^2_E(A,E) = U^{A\to H}(A,E)$, $Q^3_E(B,C) = S^{B\to H}(B,C)$, and $Q^4_E(B,D) = T^{B\to H}(B,D)$. The $\{\{B\}\}$-connected components of Q_E are $Q^1_E(A,C,D) = R(A,B), S(B,C), T(B,D)$ and $Q^2_E(A,E) = U(A,E)$. We obtain the skew-aware queries $Q^1_E(A,C,D) = R^{A\to H, B\to L}(A,B)$, $S^{B\to L}(B,C)$, $T^{B\to L}(B,D)$ and $Q^2_E(A,E) = U^{A\to H}(A,E)$. ⌟

The procedure $\tau(\omega, \mathcal{F})$ shown in Figure 3 constructs all skew-aware queries of the preprocessing stage. The procedure returns a set $\{s_X\}_{X\in\mathcal{F}}$ of maps, where s_X maps each set of join v-sets of the query induced by X wrt. ω to a set of skew-aware queries. We explain the construction in more detail. For each variable $X \in \mathcal{F}$, the procedure first constructs the query Q_X induced by X wrt. to ω (Line 3). For each $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, it creates the $\mathcal{J}$-connected components $\mathsf{Con}_\mathcal{J}(Q) = \{Q_1, \ldots, Q_k\}$ of Q_X (Line 5). Then, it adds $\mathcal{J} \mapsto \{S_\mathcal{J}(Q_1), \ldots, S_\mathcal{J}(Q_k)\}$ to the map s_X, where each $S_\mathcal{J}(Q_i)$ is the skew-aware query obtained from Q_i (Line 6). The procedure returns the set of maps s_X with $X \in \mathcal{F}$ (Line 7).

Consider an ACQ $Q(\mathcal{F})$, a variable order ω for Q, $\epsilon_X \in [0,1]$ for $X \in \mathcal{F}$, and a database of size N. In the preprocessing stage, we first eliminate all dangling tuples in the database using Yannakakis' algorithm [19]. For each free variable $X \in \mathcal{F}$, we then do the following. First, we compute the fresh relations in the induced query Q_X. Then, we partition the relations in Q_X on each join v-set in $\mathsf{JVSets}(Q_X)$ using the threshold N^{ϵ_X}. Finally, we compute the results of all skew-aware queries in $\tau(\omega, \mathcal{F})$.

We next show that the skew-aware queries in $\tau(\omega, \mathcal{F})$ represent the input query $Q(\mathcal{F})$.

▶ **Definition 12** (Skew-aware Composite Queries). *Given an ACQ $Q(\mathcal{F})$ and $\mathcal{J} \subseteq \mathsf{JVSets}(Q)$, consider the $\mathcal{J}$-connected components $\mathsf{Con}_\mathcal{J}(Q) = \{Q_1, \ldots, Q_k\}$ of Q and the skew-aware queries $S_\mathcal{J}(Q_1)$, …, $S_\mathcal{J}(Q_k)$. The query $Q_\mathcal{J}(\mathcal{F}) = S_\mathcal{J}(Q_1), \ldots, S_\mathcal{J}(Q_k)$ is called a skew-aware composite query obtained from Q.*

Given an induced query $Q_X(\mathcal{F}_X)$, let $Q_X^{\cup}(\mathcal{F}_X)$ be the union of all skew-aware composite queries obtained from Q_X and $Q_{\bowtie}(\mathcal{F}) = (Q_X^{\cup}(\mathcal{F}_X))_{X\in\mathcal{F}}$ the join of these unions. On any database, the result of $Q(\mathcal{F})$ is equal to the result of $Q_{\bowtie}(\mathcal{F})$:

▶ **Proposition 13.** *For any ACQ $Q(\mathcal{F})$, it holds $Q(\mathcal{F}) \equiv Q_{\bowtie}(\mathcal{F})$.*

4.1 Preprocessing Time

Consider the output $\{s_X\}_{X\in\mathcal{F}}$ of the procedure $\tau(\omega, \mathcal{F})$ in Figure 3. Assume that relations in the induced queries of ω are partitioned using the trade-off parameters $\{\epsilon_X\}_{X\in\mathcal{F}}$. The preprocessing time is dominated by the time to compute the skew-aware queries in $s_X(\mathcal{J})$ for any $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$. Consider for some $X \in \mathcal{F}$ and $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, a skew-aware query $\hat{Q}$ in $s_X(\mathcal{J})$. The atoms in the query are connected via light join keys. There are several ways to compute such a query. One strategy is to materialise the result of $\hat{Q}$ using factorised computation over all relations in the query [16]. Another strategy is to use a join plan where we join in one relation at a time using light join keys. We can also mix these strategies, i.e., we can use factorised computation over a subset of the relations and then join in, one by one, the remaining relations.

We formalise join strategies for a skew-aware query $\hat{Q}$ by the notion of a *cover*. A cover is a pair $(\mathcal{R}_1, \mathcal{R}_2)$ with $\mathcal{R}_1, \mathcal{R}_2 \subseteq \mathit{atoms}(\hat{Q})$ such that $\mathcal{R}_1$ is non-empty and $\mathcal{R}_1 \cup \mathcal{R}_2 = \mathit{atoms}(\hat{Q})$. We denote by $\mathsf{Cov}(\hat{Q})$ the set of all covers for $\hat{Q}$. A cover $(\mathcal{R}_1, \mathcal{R}_2)$ represents the strategy where we first use factorised computation to join the relations in $\mathcal{R}_1$ and then join in the relations in $\mathcal{R}_2$ using light join keys. Assuming that the database size is N and the relations are partitioned using the threshold N^{ϵ_X}, the time required by this strategy is $\mathcal{O}(N^{c+|\mathcal{R}_2|\epsilon_X})$, where $c = \max\{\mathsf{w}(\hat{Q}_1), \rho^*_{\hat{Q}_1}(\mathcal{V})\}$, $\mathcal{V} = \mathit{vars}(\mathcal{R}_1) \cap (\mathit{free}(\hat{Q}) \cup \mathit{vars}(\mathcal{R}_2))$, and $\hat{Q}_1$ is the query with free variables $\mathcal{V}$ whose body consists of the atoms in $\mathcal{R}_1$. The following cost measure p, called *preprocessing cost*, minimises the exponent of the computation time over all possible covers of the skew-aware query $\hat{Q}$:

$$p(\hat{Q}, \epsilon_X) = \min_{(\mathcal{R}_1,\mathcal{R}_2)\in\mathsf{Cov}(\hat{Q})} (c + |\mathcal{R}_2|\epsilon_X) \tag{1}$$

where c is defined as above. The exponent of the overall preprocessing time is bounded by the maximal cost of any skew-aware query for the variable order ω:

$$p(\omega, \{\epsilon_X\}_{X\in\mathcal{F}}) = \max_{X\in\mathcal{F}} p(Q_X, \epsilon_X) \tag{2}$$

$$p(Q_X, \epsilon_X) = \max_{\mathcal{J}\subseteq\mathsf{JVSets}(Q)} \max_{\hat{Q}\in\mathsf{Con}_{\mathcal{J}}(Q)} p(\hat{Q}, \epsilon_X) \tag{3}$$

We state the preprocessing time of our approach using the preprocessing cost.

▶ **Proposition 14.** *Given an ACQ $Q(\mathcal{F})$, a variable order ω for Q, $\epsilon_X \in [0,1]$ for $X \in \mathcal{F}$, and a database of size N, the skew-aware queries in $\tau(\omega, \mathcal{F})$ can be computed in time $\mathcal{O}(N^p)$, where $p = p(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$.*

▶ **Example 15.** Consider the query $Q_E(B, E) = R'(A, B), U(A, E)$ from Figure 1. For any $\epsilon_E \in [0, 1]$, a cover with minimal cost is $(\{R(A, B)\}, \{U(A, E)\})$, which implies $p(Q_E, \epsilon_E) = 1 + \epsilon_E$. The preprocessing cost for the query Q_1 from Example 7 is dominated by this cost. This leads to $\mathcal{O}(N^{1+\epsilon_E})$ overall preprocessing time.

Consider now the query $Q_E(C, D, E) = R(A, B), S(B, C), T(B, D), U(A, E)$ from Figure 2. Depending on ϵ_E, we can choose one of the two covers $(\{R(A, B)\}, \{S(B, C), T(B, D), U(A, E)\})$ and $(\{R(A, B), T(B, D), U(A, E)\}, \{S(B, C)\})$. This leads to $p(Q_E, \epsilon_E) = \min\{1 + 3\epsilon_E, 2 + \epsilon_E\}$. This cost dominates the preprocessing cost for the query Q_2 in Example 7. This implies $\mathcal{O}(N^{\min\{1+3\epsilon_E, 2+\epsilon_E\}})$ overall preprocessing time. ⌟

5 Enumeration

Consider an ACQ $Q(\mathcal{F})$ and a variable order ω for Q. In the preprocessing stage, we construct for each $X \in \mathcal{F}$, an induced query Q_X with free variables $\{X\} \cup dep(X)$. In the enumeration phase, we consider $dep(X)$ as the *input* variables and X as the *output* variable of Q_X. The main building block of our enumeration procedure is a mechanism that supports the enumeration of the X-values in the result of Q_X, given that the values of the input variables $dep(X)$ are fixed. Assume for now that we have such an enumeration mechanism for each induced query. We can enumerate the result of Q as follows. We traverse the variables in ω in pre-order. The input variables of an induced query Q_X are a subset of the variables above X in ω, so when we arrive at variable X, the input variables of Q_X are already fixed by the output values of the induced queries for the variables above X. We use the enumeration mechanism of Q_X to obtain the X-values paired with the fixed input values. Each tuple in the result of Q is a concatenation of the output values of the induced queries. The overall enumeration delay for Q is the sum of the enumeration delays for all induced queries.

▶ **Example 16.** Consider the query Q_1 and its variable order ω_1 in Figure 1. The query Q_1 has four induced queries: $Q_B(B)$, $Q_C(B, C)$, $Q_D(B, D)$, and $Q_E(B, E)$. The query Q_B has no input variables and the output variable B, whereas Q_C, Q_D, and Q_E have the input variable B and the output variables C, D, and respectively E. Following the variable order ω_1, we enumerate the B-values from Q_B. For each such B-value b, we enumerate the C-, D-, and E-values that are paired with b from Q_C, Q_D, and respectively Q_E. The concatenation of output values, one from each induced query, forms a tuple in the result of Q_1. The enumeration delay of Q_1 is the sum of the enumeration delays of the four induced queries. ⌟

5.1 Enumeration for an Induced Query

We now show the enumeration mechanism for an induced query Q_X. The preprocessing stage decomposes Q_X into a set of skew-aware composite queries such that the result of Q_X is the union of the results of these composite queries. We first show how to enumerate the result of one composite query and then the union of the results of all composite queries built for Q_X.

Consider a skew-aware composite query $Q_{\mathcal{J}}(\mathcal{F}) = Q_1(\mathcal{X}_1), \ldots, Q_k(\mathcal{X}_k)$ obtained from the $\mathcal{J}$-connected components of an induced query $Q_X(\mathcal{F})$ for some $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$. The query $Q_{\mathcal{J}}$ has the same free variables $\mathcal{F}$ as Q_X. We consider the output variable O and input variables $\mathcal{I}$ of Q_X as the output and respectively input variables of $Q_{\mathcal{J}}$. Given a tuple $\mathbf{i}$ over $\mathcal{I}$, we next show how to enumerate the O-values that are paired with $\mathbf{i}$ in the result of $Q_{\mathcal{J}}$.

Case 1. We first discuss the case that the output variable O is a join variable in $Q_{\mathcal{J}}$. For each skew-aware query $Q'(\mathcal{X}') \in \{Q_1(\mathcal{X}_1), \ldots, Q_k(\mathcal{X}_k)\}$, we filter out all tuples in the result of Q' that do not match the input tuple $\mathbf{i}$. The time to do this filtering is bounded by the size of the result of Q' after fixing the input variables by $\mathbf{i}$. Then, we evaluate the skew-aware composite query $Q_{\mathcal{J}}$ over the filtered skew-aware queries.

By construction, each non-join variable is free, i.e., either an input or an output variable (non-join bound variables are removed by bound-GYO as shown in Section 3). Since $Q_{\mathcal{J}}$ has only one output variable O, which is a join variable in Case 1, all non-join variables of $Q_{\mathcal{J}}$ are input variables. These variables are fixed by the input tuple $\mathbf{i}$, so the output variable O is the only free variable that is not fixed. This makes $Q_{\mathcal{J}}$ a free-connex query. Hence, for an input tuple $\mathbf{i}$, we can compute the first result tuple of $Q_{\mathcal{J}}$ in time linear in the output size of the filtered skew-aware queries and enumerate the remaining tuples with constant delay [3].

Given an input tuple, the enumeration delay for $Q_{\mathcal{J}}$ is thus linear in the size of the result of the skew-aware queries. Since the values of all non-join variables are fixed, this size is bounded by the size of the projections of the skew-aware queries onto their join variables:

$$\max_{\hat{Q} \in atoms(Q_{\mathcal{J}})} |\pi_{\hat{\mathcal{F}}} \hat{Q}|, \quad \text{where } \hat{\mathcal{F}} = free(\hat{Q}) \cap \bigcup_{\mathcal{V} \in \mathsf{JVSets}(Q_X)} \mathcal{V}. \tag{4}$$

The set $\hat{\mathcal{F}}$ consists of the free variables of $\hat{Q}$ that are part of the join v-sets of Q_X.

▶ **Example 17.** Consider the skew-aware composite query $Q(A, C, D, E) = Q_1(A, B)$, $Q_2(A, E)$, $Q_3(B, C)$, $Q_4(B, D)$, which joins the materialised skew-aware queries Q_1, Q_2, Q_3, and Q_4 and has the output variable A and the input variables C, D, and E. For a given tuple (c, d, e) over (C, D, E), we show how to enumerate the A-values that are paired with (c, d, e) in the result of Q. The output variable A is a join variable. For each skew-aware query, we filter out all tuples that do not match (c, d, e). For Q_3 and Q_4, we keep only the tuples that contain c and respectively d, denoted as $Q_3(B, c)$ and $Q_4(B, d)$. For Q_2, we obtain $Q_2(A, e)$. The query Q_1 has no input variables, so no need for filtering. After the filtering, the query becomes $Q(A, c, d, e) = Q_1(A, B), Q_2(A, e), Q_3(B, c), Q_4(B, d)$. It is free-connex since A is the only unfixed free variable. After the preprocessing stage that takes time bounded by the maximum of the sizes $|\pi_{A,B}(Q_1(A, B))|$, $|\pi_A(Q_2(A, E))|$, $|\pi_B(Q_3(B, C))|$, and $|\pi_B(Q_4(B, D))|$, we can enumerate the A-values with constant delay. ◿

Case 2. We now discuss the case where the output variable O of the composite query $Q_{\mathcal{J}}$ is not a join variable. Let $Q_O(\mathcal{X}_O) \in \{Q_1(\mathcal{X}_1), \ldots, Q_k(\mathcal{X}_k)\}$ be the skew-aware query with $O \in \mathcal{X}_O$ and $\mathcal{F}_O \subseteq \mathcal{X}_O$ be the join variables in the schema of Q_O. We define a sub-query $Q_{\mathcal{F}_O}$ of $Q_{\mathcal{J}}$ by removing Q_O from the body of $Q_{\mathcal{J}}$ and setting the schema of $Q_{\mathcal{F}_O}$ to be $(\mathcal{F} \setminus \mathcal{X}_O) \cup \mathcal{F}_O$, that is, removing the free variables of Q_O from the schema of $Q_{\mathcal{J}}$ and adding the join variables $\mathcal{F}_O$ in Q_O. We set the new variables from $\mathcal{F}_O$ to be the output variables and other free variables, $(\mathcal{F} \setminus \mathcal{X}_O) \setminus \mathcal{F}_O$, to be the input variables of $Q_{\mathcal{F}_O}$.

These input variables of $Q_{\mathcal{F}_O}$ are a subset of the input variables $\mathcal{I}$ of $Q_{\mathcal{J}}$ since all variables in $\mathcal{F} \setminus \mathcal{X}_O$ are input variables of $Q_{\mathcal{J}}$ (the only output variable O of $Q_{\mathcal{J}}$ is in $\mathcal{X}_O$ and removed). Hence, an input tuple $\mathbf{i}$ over $\mathcal{I}$ fixes the input variables of $Q_{\mathcal{F}_O}$. When these input variables are fixed, $Q_{\mathcal{F}_O}$ becomes free-connex. The reason is that by adding an atom with variables $\mathcal{X}_O$ to $Q_{\mathcal{F}_O}$, the query remains acyclic. All output variables of $Q_{\mathcal{F}_O}$ are join variables, so we can compute the $\mathcal{F}_O$-tuples in the result of $Q_{\mathcal{F}_O}$ as in the previous case. The computation time is as defined in Equation (4). Then, we can enumerate the distinct output of $Q_{\mathcal{J}}$ using the skew-aware query Q_O, as explained next.

For the given input tuple $\mathbf{i}$ over $\mathcal{I}$ and for each output $\mathcal{F}_O$-tuple $\mathbf{t}$ enumerated from $Q_{\mathcal{F}_O}$ for $\mathbf{i}$, all variables in Q_O except O are fixed by $\mathbf{i}$ and $\mathbf{t}$, therefore, the skew-aware query Q_O supports constant-time lookups and constant-delay enumeration of the matching O-values in the result of Q_O. We decompose Q_O into a union of queries instantiated for the distinct $\mathcal{F}_O$-tuples and the input tuple $\mathbf{i}$. From each query, instantiated for an $\mathcal{F}_O$-tuple $\mathbf{t}$ and the input tuple $\mathbf{i}$, we can enumerate with constant delay the O-values that are paired with $\mathbf{t}$ and $\mathbf{i}$ in the result of Q_O. The O-values from these instantiated queries might not be disjoint, so we cannot enumerate the O-values from each query one after the other. To enumerate the distinct O-values, we employ the Union algorithm [9]. The enumeration delay is linear in the sum of the enumeration delays of the instantiated queries. Since each instantiated query supports constant-delay enumeration, the enumeration delay over all instantiated queries is linear in the number of distinct $\mathcal{F}_O$-tuples, which is bounded by the size defined by Expression (4). Overall, the enumeration delay of the result of $Q_{\mathcal{J}}$ in this case is bounded by the size defined by Expression (4).

▶ **Example 18.** Consider the induced query Q_E and its $\emptyset$-connected components in Figure 2. The skew-aware composite query is of the form: $Q_E^{\emptyset}(C, D, E) = Q_1(A, B), Q_2(A, E), Q_3(B, C), Q_4(B, D)$, where $Q_1, \ldots, Q_4$ are the skew-aware queries that are heavy on the join v-sets $\{A\}$ and $\{B\}$. The query has the input variables C and D and the output variable E. The output variable E is not a join variable. We build a sub-query Q_A from $Q_E^{\emptyset}$ by removing the query Q_2, which contains the output variable E, and setting the join variable A to be the output variable of Q_A. The sub-query is then $Q_A(A, C, D) = Q_1(A, B), Q_3(B, C), Q_4(B, D)$. Given an input tuple (c, d) over (C, D), we retain only the tuples from the skew-aware queries Q_1, Q_3, and Q_4 that match (c, d) to obtain the query $Q_A(A, c, d) = Q_1(A, B), Q_3(B, c), Q_4(B, d)$. The query $Q_A(A, c, d)$ is free-connex since A is the only unfixed free variable. Preparing for the enumeration of the A-value from Q_A takes time linear in the size of the filtered skew-aware queries in Q_A, $\mathcal{O}(\max(|\pi_{A,B}(Q_1(A, B))|, |\pi_B(Q_3(B, C))|, |\pi_B(Q_4(B, D))|))$. We can then enumerate the A-values in the result of $Q_A(A, c, d)$ with constant delay [3]. For each such A-value a, we can enumerate the E-values paired with a in $Q_2(A, E)$ with constant delay. The output values of $Q_E^{\emptyset}$ are over the schema (A, E) and might contain duplicates of E-values. We apply the Union algorithm [9] to enumerate only the distinct E-values with the delay linear in the number of distinct A-values, which is bounded by $\mathcal{O}(\pi_A(Q_1(A, B)))$. Since $|\pi_A(Q_1(A, B))| \leq |\pi_{A,B}(Q_1(A, B)|$, the delay for the enumeration of the output values of $Q_E^{\emptyset}$ is bounded by $\mathcal{O}(\max(|\pi_{A,B}(Q_1(A, B))|, |\pi_B(Q_3(B, C))|, |\pi_B(Q_4(B, D))|))$. ⌟

So far, we discussed the enumeration of the result of one skew-aware composite query. To enumerate the result of an induced query, we need to enumerate the union of the results of all skew-aware composite queries obtained from the induced query. Since these results might not be disjoint, we again apply the Union algorithm [9]. The enumeration delay is the maximum of the enumeration delays of all skew-aware composite queries.

Overall, the enumeration delay for a skew-aware composite query is bounded by the size of the projections of its skew-aware queries onto their join variables, as defined in Expression (4). The enumeration delay for an induced query is the maximum of the enumeration delays of its skew-aware composite queries, and the enumeration delay for the input ACQ is the maximum of the enumeration delays of its induced queries. Hence, the enumeration delay of an ACQ $Q(\mathcal{F})$ is given by:

$$\max_{X \in \mathcal{F}} \max_{\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)} \max_{\hat{Q} \in \mathsf{Con}_{\mathcal{J}}(Q_X)} |\pi_{\hat{\mathcal{F}}} S_{\mathcal{J}}(\hat{Q})|, \tag{5}$$

where $S_{\mathcal{J}}(\hat{Q})$ is the skew-aware query obtained from $\hat{Q}$ and $\hat{\mathcal{F}} = \mathit{free}(\hat{Q}) \cap \bigcup_{\mathcal{V} \in \mathsf{JVSets}(Q_X)} \mathcal{V}$.

5.2 Enumeration Delay

Let $\{s_X\}_{X \in \mathcal{F}}$ be the output of the procedure $\tau(\omega, \mathcal{F})$ in Figure 3 for some variable order ω and a set $\mathcal{F}$ of free variables. Assume that the relations are partitioned using parameter values $\{\epsilon_X\}_{X \in \mathcal{F}}$. Consider a skew-aware query $Q \in s_X(\mathcal{J})$ for some $X \in \mathcal{F}$ and $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$. Expression (5) implies that the overall delay is bounded by the maximal size of the projection of the result of such a query Q onto $\hat{\mathcal{F}} = \mathit{free}(Q) \cap \bigcup_{\mathsf{JVSet} \in \mathsf{JVSets}(Q_X)} \mathsf{JVSet}$, i.e., the free variables of Q that are part of join v-sets of Q_X. We give three bounds on the exponent e of the size complexity: 1) One bound on e is given by $p(Q, \epsilon_X)$, where p is the preprocessing cost defined in Equation (1). 2) Another bound is $\rho^*_{Q_X}(\hat{\mathcal{F}})$, i.e., the fractional edge cover number of $\hat{\mathcal{F}}$. 3) The relations in Q are heavy on the join v-sets of Q_X that are included in $\hat{\mathcal{F}}$. This means that for each relation R in Q and $\mathsf{JVSet} \in \mathsf{JVSets}(Q_X)$, we have $|\pi_{\mathsf{JVSet}} R| \leq N^{1-\epsilon_X}$.

Hence, we can bound e by $\rho^*_{\mathsf{JVSets}(Q_X)}(\hat{\mathcal{F}})(1-\epsilon_X)$, i.e., the minimal number of join v-sets covering all variables in $\hat{\mathcal{F}}$ multiplied by $1-\epsilon_X$. We define the enumeration cost e of Q_X, Q, and ϵ_X as the minimum of these three bounds:

$$e(Q_X, Q, \epsilon_X) = \min\{\, p(Q, \epsilon_X),\ \rho^*_{Q_X}(\hat{\mathcal{F}}),\ \rho^*_{\mathsf{JVSets}(Q_X)}(\hat{\mathcal{F}})(1-\epsilon_X)\,\}, \tag{6}$$

The overall enumeration cost is given by the maximum $e(Q_X, Q, \epsilon_X)$ over all induced and skew-aware queries:

$$e(\omega, \{\epsilon_X\}_{X\in\mathcal{F}}) = \max_{X\in\mathcal{F}} e(Q_X, \epsilon_X) \tag{7}$$

$$e(Q_X, \epsilon_X) = \max_{\mathcal{J}\subseteq\mathsf{JVSets}(Q_X)} \max_{\hat{Q}\in\mathsf{Con}_{\mathcal{J}}(Q_X)} e(Q_X, \hat{Q}, \epsilon_X) \tag{8}$$

We state the delay of our approach for a given variable order and trade-off parameters.

▶ **Proposition 19.** *Given an ACQ $Q(\mathcal{F})$, a variable order ω for Q, $\epsilon_X \in [0,1]$ for $X \in \mathcal{F}$, and a database of size N, the result of Q can be enumerated from the queries in $\tau(\omega, \mathcal{F})$ with $\mathcal{O}(N^e)$ delay, where $e = e(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$.*

▶ **Example 20.** Consider the induced query Q_E from Figure 2 and the skew-aware composite query $Q^{\emptyset}_E(C,D,E) = Q_1(A,B), Q_2(B,C), Q_3(B,D), Q_4(A,E)$ obtained from its $\emptyset$-connected components. The query $Q^{\emptyset}_E(C,D,E)$ has the join variables $\hat{\mathcal{F}} = \{A,B\}$. Let $\epsilon_E \in [0,1]$.

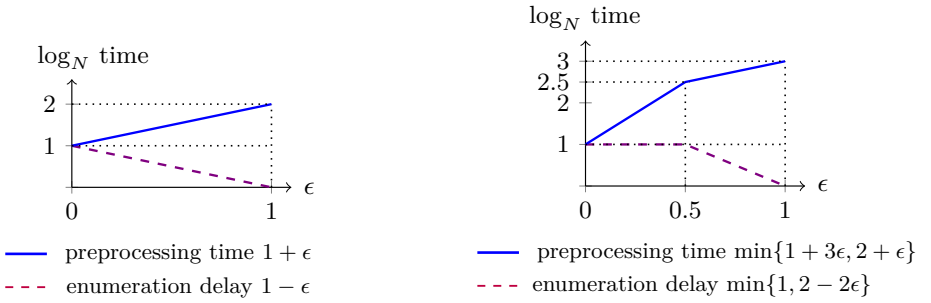
We compute the enumeration cost $e(Q_E, Q_1, \epsilon_E)$ of Q_1 as defined in Equation (6). Since the body of Q_1 consists of a single atom that contains A and B, we have $p(Q_1, \epsilon_E) = 1$ and $\rho^*_{Q_E}(A,B) = 1$. Since A and B can be covered by two join v-sets of Q_E, we get $\rho^*_{\mathsf{JVSets}(Q_E)}(A,B)(1-\epsilon_E) = 2(1-\epsilon_E) = 2-2\epsilon_E$. Overall, we obtain $e(Q_E, Q_1, \epsilon_E) = \min\{1, 2-2\epsilon\}$. The query Q_2 consists of one atom. The only join variable in its schema is B. Hence, the enumeration cost of Q_2 is $e(Q_E, Q_2, \epsilon_E) = 1 - 1\epsilon_E$. Similarly, the enumeration cost for Q_3 is $e(Q_E, Q_3, \epsilon_E) = 1 - 1\epsilon_E$. The skew-aware query Q_4 has no join variables. Its enumeration cost is therefore constant.

The maximum of the enumeration costs for the four skew-aware queries considered above is $\max\{\min\{1, 2-2\epsilon_E\}, 1-\epsilon_E, 0\} = \min\{1, 2-2\epsilon_E\}$. This cost dominates the enumeration costs of all induced queries of the query Q_2 in Example 7. This implies $\mathcal{O}(N^{\min\{1,2-2\epsilon\}})$ overall enumeration delay. ⌟

6 Benefits of Our Approach

In this section we exemplify the benefits of our approach versus two mainstream approaches dubbed lazy and eager. The lazy approach invests no time in the preprocessing phase at the expense of linear enumeration delay [3]. The eager approach constructs a factorised representation of the query result in time $\mathcal{O}(N^{\mathsf{w}})$, where w is the width of the query, after which it needs constant enumeration delay [16]. In case we only need to enumerate a fraction of the query result, our approach can be asymptotically faster than both competing approaches. Furthermore, if the fraction is known in advance, we can derive the trade-off parameters for the lowest overall complexity to compute this fraction of the query result.

▶ **Example 21.** Consider the query $Q_1(B,C,D,E) = R(A,B), S(B,C), T(B,D), U(A,E)$ from Example 7 whose hypergraph is depicted in Figure 1 (left). The query has width $\mathsf{w} = 2$ (see Example 8). Assume that the input relations have size N. Our approach achieves $\mathcal{O}(N^{1+\epsilon})$ preprocessing time (Example 15) and $\mathcal{O}(N^{1-\epsilon})$ enumeration delay for any $\epsilon \in [0,1]$, as depicted in Figure 4 (top left). These complexities are obtained by using the



γ	0	$\frac{1}{2}$	1	$1\frac{1}{2}$	2	$2\frac{1}{2}$	3
lazy	1	$1\frac{1}{2}$	2	$2\frac{1}{2}$	3	$3\frac{1}{2}$	4
eager	2	2	2	2	2	$2\frac{1}{2}$	3
ours	1	$1\frac{1}{4}$	$1\frac{1}{2}$	$1\frac{3}{4}$	2	$2\frac{1}{2}$	3
ϵ	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	1	1	1

γ	0	$\frac{1}{2}$	1	$1\frac{1}{2}$	2	$2\frac{1}{2}$	3
lazy	1	$1\frac{1}{2}$	2	$2\frac{1}{2}$	3	$3\frac{1}{2}$	4
eager	3	3	3	3	3	3	3
ours	1	$1\frac{1}{2}$	2	$2\frac{1}{2}$	$2\frac{2}{3}$	$2\frac{5}{6}$	3
ϵ	0	0	0	0	$\frac{2}{3}$	$\frac{5}{6}$	1

Figure 4 Top: Evaluation trade-offs for the query Q_1 from Example 21 (left) and the query Q_2 from Example 22 (right). Bottom: The overall evaluation times achieved by the lazy, eager, and our approaches in case we want to enumerate N^γ output tuples of Q_1 (left) and of Q_2 (right). The last row gives the ϵ values at which we achieve the complexities of our approach.

same trade-off parameter value ϵ for all induced queries. The lazy approach achieves linear enumeration delay after constant preprocessing time, while the eager approach achieves constant enumeration delay after $\mathcal{O}(N^2)$ preprocessing time. Our approach recovers the lazy approach[2] at $\epsilon = 0$ and the eager approach at $\epsilon = 1$. For any $\epsilon \in (0, 1)$, our approach allows for new trade-offs between preprocessing and enumeration.

The query result has at most N^3 tuples. Assume that we want to compute N^γ tuples from the query result for $0 \leq \gamma \leq 3$. The second to fourth rows of the bottom left table in Figure 4 give the exponents of the overall evaluation times achieved by the lazy, eager, and our approaches for different values of γ. The last row gives the ϵ values at which we achieve the complexities of our approach. For instance, for $\gamma = \frac{1}{2}$, the lazy approach needs $\mathcal{O}(N \cdot N^{\frac{1}{2}}) = \mathcal{O}(N^{1\frac{1}{2}})$, the eager approach needs $\mathcal{O}(N^2 + N^{\frac{1}{2}}) = \mathcal{O}(N^2)$, and our approach needs $\mathcal{O}(N^{\frac{1}{4}} + N^{\frac{3}{4}}N^{\frac{1}{2}}) = \mathcal{O}(N^{1\frac{1}{4}})$ time. In case γ is equal to $\frac{1}{2}$, 1, or $1\frac{1}{2}$, the computation time of our approach is strictly better than of the lazy and eager approaches. For the other cases shown in the table, our approach recovers the best of the other two approaches. ⌟

▶ **Example 22.** Consider the variant Q_2 of the query Q_1 from Example 21 where the variable B is bound. Its hypergraph is shown in Figure 2 (left). This query has width 3 (Example 8). Assume that the input relations have size N. We explained in Examples 15 and 20 that our approach achieves $\mathcal{O}(N^{\min\{1+3\epsilon, 2+\epsilon\}})$ preprocessing time and $\mathcal{O}(N^{\min\{1,2-2\epsilon\}})$ enumeration delay for any $\epsilon \in [0, 1]$, cf. Figure 4 (top right). Just like in case of Q_1, we obtain these complexities by using the same trade-off parameter value for all induced queries. We recover prior work using $\epsilon = 0$ (lazy) and $\epsilon = 1$ (eager). For $\epsilon \in (\frac{1}{2}, 1)$, we obtain new trade-offs.

[2] Our approach requires at least linear preprocessing time. In case the delay is linear, the preprocessing time is also linear and we can shift it to the enumeration of the first tuple to match the constant preprocessing time of the lazy approach.

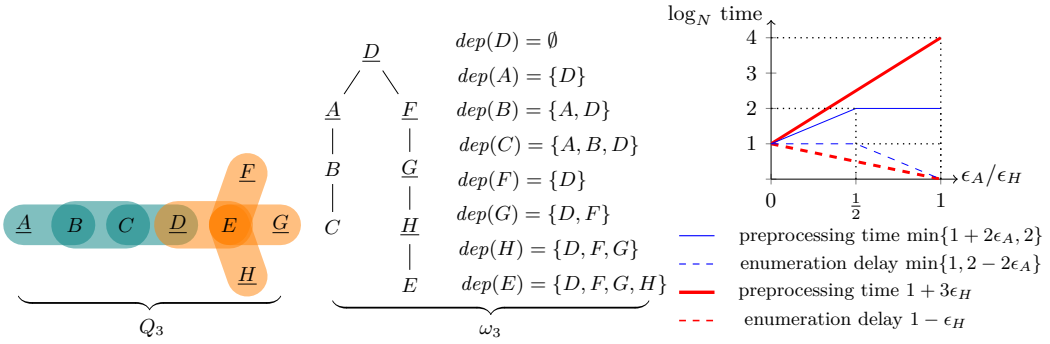


Figure 5 Left to right: Hypergraph of the query Q_3 in Example 23; variable order ω_3 for Q_3; trade-offs for the induced queries Q_A (thin blue lines) and Q_H (thick red lines).

There can be at most N^3 tuples in the result of Q_2. The bottom right table in Figure 4 gives the exponents of the overall computation times for the lazy, eager and our approaches for computing N^γ tuples in the query result. In case γ equals 2 or $2\frac{1}{2}$, the overall computation time of our approach is strictly lower than of the other approaches. In all other cases considered in the table, our approach recovers the best of the other two approaches. ⌟

The next example illustrates two aspects of our approach. First, even though our approach may not achieve novel trade-offs for some induced queries for a given acyclic query Q, it may still achieve new trade-offs for the entire query Q. Second, in case we are given a budget for one of the preprocessing and enumeration costs, we can pick the trade-off parameters so as to optimise for the other cost.

▶ **Example 23.** Consider the query $Q_3(A, D, F, G, H) = R(A, B), S(B, C), T(C, D), U(D, E), V(E, F), W(E, G), X(E, H)$ visualised in Figure 5 (left) and its variable order ω_3 in Figure 5 (middle). The query has width 4. We focus on the queries induced by the variables A and H, since their complexities dominate the overall complexity of Q_3. The query induced by A is the path query $Q_A(A, D) = R(A, B), S(B, C), T'(C, D)$ and has width 2. The query induced by H is the star query $Q_H(D, F, G, H) = U'(D, E), V(E, F), W(E, G), X(E, H)$ and has width 4. The relations T' and U' are computed by bound-GYO from T and U through semi-join reductions with U and T, respectively. For both induced queries, the lazy approach achieves linear enumeration delay after constant preprocessing time. The eager approach achieves constant enumeration delay after $\mathcal{O}(N^2)$ preprocessing time for Q_A and after $\mathcal{O}(N^4)$ preprocessing time for Q_H. For any $\epsilon_H \in [0, 1]$, our approach evaluates Q_H with $\mathcal{O}(N^{1+3\epsilon_H})$ preprocessing time and $\mathcal{O}(N^{1-\epsilon_H})$ enumeration delay, as visualised with thick red lines in Figure 5 (right). For any $\epsilon_A \in [0, 1]$, it takes $\mathcal{O}(N^{\min\{1+2\epsilon_A, 2\}})$ preprocessing time and $\mathcal{O}(N^{\min\{1, 2-2\epsilon_A\}})$ enumeration delay for Q_A, as depicted with thin blue lines in Figure 5 (top right). For both queries, it recovers the lazy approach at $\epsilon_A = \epsilon_H = 0$ and the eager approach at $\epsilon_A = \epsilon_H = 1$. For any $\epsilon_H \in (0, 1)$, it allows for new trade-offs for Q_H beyond the aforementioned ones of the existing approaches. For instance, for $\epsilon_H = \frac{1}{2}$, it achieves $\mathcal{O}(N^{2\frac{1}{2}})$ preprocessing time, which is less than the preprocessing time of the eager approach and $\mathcal{O}(N^{\frac{1}{2}})$ enumeration delay, which is less than the delay of the lazy approach.

Our approach does not achieve new trade-offs for the induced query Q_A, but it still achieves new trade-offs for the whole query Q_3. The lazy and eager approaches achieve for Q_3 the same trade-offs as for Q_H. Our approach achieves $\mathcal{O}(N^p)$ preprocessing time and $\mathcal{O}(N^e)$ delay, where $p = \max\{\min\{1 + 2\epsilon_A, 2\}, 1 + 3\epsilon_H\}$ and $e = \max\{\min\{1, 2 - 2\epsilon_A\}, 1 - \epsilon_H\}$. This means that for $\epsilon_A = 1$ and $\epsilon_H \in (\frac{1}{2}, 1)$, we obtain new trade-offs for Q_3.

Now, assume that we have a preprocessing cost budget of $2\frac{1}{2}$, i.e., we can afford $\mathcal{O}(N^{2\frac{1}{2}})$ preprocessing time. By setting $\epsilon_H = \frac{1}{2}$, we achieve the lowest enumeration cost of $\frac{1}{2}$ for Q_H. The preprocessing cost of Q_A does not reach $2\frac{1}{2}$ for any ϵ_A. So, one possibility is to set $\epsilon_A = 1$ to obtain an enumeration cost of 0 for Q_A and an overall enumeration cost of $\frac{1}{2}$. If we have an enumeration cost budget of $\frac{1}{4}$, we can set $\epsilon_H = \frac{3}{4}$ to obtain the lowest preprocessing cost of $3\frac{1}{4}$ for Q_H. Since the preprocessing cost of Q_A never reaches $3\frac{1}{4}$, we set $\epsilon_A = 1$ to achieve 0 enumeration cost for Q_A and an overall enumeration cost of $\frac{1}{4}$. ⌟

7 Conclusion

In this paper we introduce an evaluation approach for acyclic conjunctive queries that trades off between the preprocessing time and the enumeration delay. This trade-off space includes points representing prior works and also points corresponding to novel evaluation strategies where the enumeration delay lies between linear and constant. Our approach can be extended to arbitrary conjunctive queries: in the preprocessing phase, we either fully materialise induced queries with cycles or do nothing. For such cyclic induced queries, the trade-off space of our approach thus only consists of two points where the delay is constant or as high as needed to compute the cycles in the query. In future work, we would like to generalise our approach to queries with aggregates over arbitrary semirings [1].

References

1 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: Questions Asked Frequently. In *PODS*, pages 13–28, 2016.

2 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013.

3 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *CSL*, pages 208–222, 2007.

4 Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020.

5 Christoph Berkholz and Nicole Schweikardt. Constant delay enumeration with fpt-preprocessing for conjunctive queries of bounded submodular width. In *MFCS*, pages 58:1–58:15, 2019.

6 Johann Brault-Baron. *De la pertinence de l'énumération: complexité en logiques propositionnelle et du premier ordre*. PhD thesis, Université de Caen, 2013.

7 Shaleen Deep, Xiao Hu, and Paraschos Koutris. Enumeration algorithms for conjunctive queries with projection. In *ICDT*, pages 14:1–14:17, 2021.

8 Arnaud Durand and Etienne Grandjean. First-order Queries on Structures of Bounded Degree are Computable with Constant Delay. *ACM Trans. Comput. Logic*, 8(4):21, 2007.

9 Arnaud Durand and Yann Strozecki. Enumeration complexity of logical query problems with second-order variables. In *CSL*, pages 189–202, 2011.

10 Georg Gottlob, Matthias Lanzinger, Reinhard Pichler, and Igor Razgon. Complexity analysis of generalized and fractional hypertree decompositions. *J. ACM*, 68(5):38:1–38:50, 2021.

11 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in Static and Dynamic Evaluation of Hierarchical Queries. In *PODS*, pages 375–392, 2020.

12 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *PODS*, pages 429–444, 2017.

13 Dániel Marx. Approximating fractional hypertree width. *ACM Trans. Alg.*, 6(2):29:1–29:17, 2010.

14 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42:1–42:51, 2013.

15 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3):16:1–16:40, 2018.

16 Dan Olteanu and Jakub Závodný. Size Bounds for Factorised Representations of Query Results. *ACM TODS*, 40(1):2:1–2:44, 2015.

17 Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

18 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

19 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.

20 Clement T. Yu and Meral Z. Ozsoyoglu. An algorithm for tree-query membership of a distributed query. In *COMPSAC*, pages 306–312, 1979.

A Appendix: Proofs

A.1 Proof of Theorem 2

Consider an ACQ $Q(\mathcal{F})$ and a database of size N. Assume that the pair (p, e) is included in the set $\mu(Q)$ of Pareto optimal pairs of objective values of the trade-off program in Section 1. This means that there is a variable order ω for Q and a set $\{\epsilon_X\}_{X\in\mathcal{F}}$ of trade-off parameters such that $p = p(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (2)) and $e = e(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (7)). By Propositions 14 and 19, the query Q can be evaluated with $\mathcal{O}(N^p)$ preprocessing time and $\mathcal{O}(N^e)$ delay.

A.2 Proof of Corollary 3

Consider a free-connex ACQ $Q(\mathcal{F})$. We show that $(1, 0)$ is included in the set $\mu(Q)$ of Pareto optimal pairs of objective values of our trade-off program in Section 1.

Since Q is free-connex acyclic, it admits a variable order ω of width 1 [4]. Consider an arbitrary free variable X in ω. Since the width of ω is 1, the variables $\{X\} \cup dep_\omega(X)$ are covered by a single atom of Q. This means that the body of the induced query Q_X consists of a single atom whose variables are free, i.e., the induced query is of the form $Q_X(\mathcal{X}) = R(\mathcal{X})$. This implies that $\mathsf{JVSets}(Q_X) = \emptyset$. The only $\emptyset$-connected component of Q_X is Q_X itself. This means $p(Q_X, \epsilon_X) = 1$ (Equation (1)) for any $\epsilon_X \in [0, 1]$ and $e(Q_X, Q_X, \epsilon_X) = 0$ (Equation (6)) for $\epsilon_X = 1$. Since X is an arbitrary free variable in ω, we derive that the preprocessing cost $p(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (2)) is 1 and the enumeration cost $e(\omega, \{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (7)) is 0 for parameter values $\{\epsilon_X = 1\}_{X\in\mathcal{F}}$. No variable order can admit lower preprocessing or enumeration cost. This implies $(1, 0) \in \mu(Q)$.

A.3 Proof of Corollary 4

Consider an arbitrary ACQ $Q(\mathcal{F})$. We show that $(1, e)$ with $e \leq 1$ is included in the set $\mu(Q)$ of Pareto optimal pairs of objective values of the trade-off program in Section 1.

Given an arbitrary variable order ω for Q, consider the induced query Q_X for any free variable X in ω. For any $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, let $\hat{Q}$ be an arbitrary $\mathcal{J}$-connected component of Q_X. Consider the cover $(\{R(\mathcal{X})\}, \mathcal{R})$ for $\hat{Q}$, where $R(\mathcal{X})$ is an arbitrary atom in $\hat{Q}$ and $\mathcal{R} = atoms(\hat{Q}) \setminus \{R(\mathcal{X})\}$. This means that the preprocessing cost $p(\hat{Q}, \epsilon_X)$ (Equation (1)) is upper-bounded by $1 + |\mathcal{R}|\epsilon_X$ for any $\epsilon_X \in [0, 1]$. This implies $p(\hat{Q}, 0) = 1$. By definition, $e(Q_X, \hat{Q}, 0)$ (Equation (6)) is upper-bounded by $p(\hat{Q}, 0) = 1$. Since ω and X were chosen arbitrarily, we derive that the preprocessing cost $p(\omega, \{\epsilon_X = 0\}_{X\in\mathcal{F}})$ (Equation (2)) is 1 and the enumeration cost $e(\omega, \{\epsilon_X = 0\}_{X\in\mathcal{F}})$ (Equation (7)) is at most 1. Overall, we conclude that $(1, e)$ with $e \leq 1$ is included in $\mu(Q)$.

A.4 Proof of Corollary 5

Consider an ACQ $Q(\mathcal{F})$ of width w. We show that $(\mathsf{w}, 0)$ is included in the set $\mu(Q)$ of Pareto optimal pairs of objective values of our trade-off program in Section 1.

Consider a variable order ω for Q of width w. Let $Q_X(\mathcal{F}_X)$ be the query induced by a variable $X \in \mathcal{F}$. By construction, $\rho^*_{Q_X}(\mathcal{F}_X) \leq \mathsf{w}$. For any $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, let $\hat{Q}(\hat{\mathcal{F}})$ be an arbitrary $\mathcal{J}$-connected component of Q_X. Consider the cover $(atoms(\hat{Q}), \emptyset)$ for $\hat{Q}$. Note that Q_1 as used in Equation (1) is equal to $\hat{Q}$. It follows from the structure of Q_X and $\hat{Q}$ that $\mathsf{w}(\hat{Q}) \leq \mathsf{w}$ and $\rho^*_{\hat{Q}}(\hat{\mathcal{F}}) \leq \rho^*_{Q_X}(\mathcal{F}_X) \leq \mathsf{w}$. This implies that is $p(\hat{Q}, \epsilon_X) \leq \mathsf{w}$ (Equation (1)) for any $\epsilon_X \in [0, 1]$. By definition, $e(Q_X, \hat{Q}, \epsilon_X)$ (Equation (6)) is upper-bounded by $k(1 - \epsilon_X)$, for some $k \geq 0$. By setting $\epsilon_X = 1$, we obtain $e(Q_X, \hat{Q}, 1) = 0$. Since ω and X were chosen arbitrarily, we conclude that the preprocessing cost is $p(\omega, \{\epsilon_X = 1\}_{X \in \mathcal{F}}) = \mathsf{w}$ (Equation (2)) and the enumeration cost is $e(\omega, \{\epsilon_X = 1\}_{X \in \mathcal{F}}) = 0$ (Equation (7)). This implies that $(\mathsf{w}, 0)$ is included in $\mu(Q)$.

A.5 Proof of Corollary 6

Consider a hierarchical query $Q(\mathcal{F})$ of width w. We show that for any $\epsilon \in [0, 1]$, (p, e) with $p \leq 1 + (\mathsf{w} - 1)\epsilon$ and $e \leq 1 - \epsilon$ is included in the set $\mu(Q)$ of Pareto optimal pairs of objective values of the trade-off program in Section 1.

Let ω be a variable order for Q of width w and $\epsilon \in [0, 1]$. Consider the query $Q_X(\mathcal{F}_X)$ induced by a variable $X \in \mathcal{F}$ such that $\rho^*_Q(\mathcal{F}_X) = \mathsf{w}$, where $\mathcal{F}_X = \{X\} \cup dep_\omega(X)$. Hierarchical queries stay hierarchical after removing variables or atoms. Hence, $Q_X(\mathcal{F}_X)$ is hierarchical. It follows from the construction of Q_X and the shape of hierarchical queries that the number of atoms in Q_X is w. For any $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, consider an arbitrary $\mathcal{J}$-connected component $\hat{Q}$ of Q_X. The query $\hat{Q}$ is hierarchical and the number of atoms in $\hat{Q}$ is at most w. Consider the cover $C = (\{R(\mathcal{X})\}, \mathcal{R})$ for $\hat{Q}$, where $R(\mathcal{X})$ is an arbitrary atom in $\hat{Q}$ and $\mathcal{R} = atoms(\hat{Q}) \setminus \{R(\mathcal{X})\}$. Since the width of a query with a single atom and the fractional edge cover of a variable set included in a single atom are at most one, the cost of C is at most $1 + |\mathcal{R}|\epsilon = 1 + (\mathsf{w} - 1)\epsilon$. It follows, $p(\hat{Q}, \epsilon) \leq 1 + (\mathsf{w} - 1)\epsilon$ (Equation (1)). Let $\hat{\mathcal{F}}$ (as defined in Equation (6)) be the free variables of $\hat{Q}$ that are included in the join v-sets of Q_X. Since Q_X is hierarchical, there must be a single join v-set in Q_X that subsumes $\hat{\mathcal{F}}$. Hence, $e(Q_X, \hat{Q}, \epsilon)$ is upper-bounded by $\rho^*_{\mathsf{JVSets}(Q_X)}(\hat{\mathcal{F}})(1 - \epsilon) \leq 1 - \epsilon$. We conclude that $p(\omega, \{\epsilon_X = \epsilon\}_{X \in \mathcal{F}}) \leq 1 + (\mathsf{w} - 1)\epsilon$ and $e(\omega, \{\epsilon_X = \epsilon\}_{X \in \mathcal{F}}) \leq 1 - \epsilon$. This implies that (p, e) with $p \leq 1 + (\mathsf{w} - 1)\epsilon$ and $e \leq 1 - \epsilon$ is included in $\mu(Q)$.

A.6 Proof of Proposition 13

Consider an ACQ $Q(\mathcal{F})$ and a variable order ω for Q. Let $Q_X(\mathcal{F}_X)$ be the query induced by $X \in \mathcal{F}$. In the preprocessing stage, we partition the relations of Q_X into disjoint parts. Each skew-aware composite query obtained from Q_X computes the join of a combination of these parts. Hence, the union $Q^\cup_X(\mathcal{F}_X)$ of all skew-aware composite queries is the result of the induced query $Q_X(\mathcal{F}_X)$. The query $Q_\bowtie(\mathcal{F})$ is defined by the join of the induced queries. Each induced query $Q_X(\mathcal{F}_X)$ computes the projection of the result of Q onto $\mathcal{F}_X$. The union of the free variables of the induced queries covers exactly the free variables $\mathcal{F}$ of Q. Hence, on any database, the result of $Q_\bowtie(\mathcal{F})$ is equal to the result of $Q(\mathcal{F})$.

A.7 Proof of Proposition 14

Consider an acyclic query $Q(\mathcal{F})$, a variable order ω for Q, trade-off parameters $\{\epsilon_X\}_{X\in\mathcal{F}}$, and a database of size N. Let $\{s_X\}_{X\in\mathcal{F}}$ be the output of $\tau(\omega,\mathcal{F})$ (Figure 3). In the preprocessing stage, we first eliminate the dangling tuples in the database. Since Q is acyclic, this can be done in $\mathcal{O}(N)$ time [19]. For any free variable X, we compute the fresh relations in the induced query Q_X by computing projections and semi-joins. These operations can be executed in $\mathcal{O}(N)$ time. Partitioning the relations can also be done in linear time.

The preprocessing time is dominated by the time to compute the skew-aware queries in $\{s_X\}_{X\in\mathcal{F}}$. Given $X \in \mathcal{F}$ and $\mathcal{J} \subseteq \mathsf{JVSets}(Q_X)$, consider a skew-aware query $\hat{Q}$ in $s_X(\mathcal{J})$. Let $(\mathcal{R}_1,\mathcal{R}_2)$ be a cover of $\hat{Q}$ such that $c+|\mathcal{R}_2|\epsilon_X$ is minimal, where c is defined as in the preprocessing cost $p(\hat{Q},\epsilon_X)$ (Equation (1)). Using factorised computation [16], we join the relations in $\mathcal{R}_1$ in $\mathcal{O}(N^c)$ time. Let relation S be the result of this join. Then, we use a left-deep join plan to join S, one by one, with the relations in $\mathcal{R}_2$. In each step, we take a relation from $\mathcal{R}_2$ that shares a light join v-set with the variables covered so far. Given an intermediate result S', we join S' with such a relation R from $\mathcal{R}_2$ as follows: We iterate over the tuples in S' and for each such tuple, we iterate over the matching tuples in R. Since R is light on the join v-set, this can be done in $\mathcal{O}(|S|\cdot N^{\epsilon_X})$ time. Since $\mathcal{R}_1$ and $\mathcal{R}_2$ cover all relations in $\hat{Q}$ and these relations are connected via light join v-sets, this gives overall $\mathcal{O}(N^{c+|\mathcal{R}_2|\epsilon_X})$ computation time for $\hat{Q}$. By definition of the preprocessing cost $p(\omega,\{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (2)), the overall computation time for the induced queries Q_X is $\mathcal{O}(N^p)$.

A.8 Proof of Proposition 19

Consider an acyclic query $Q(\mathcal{F})$, a variable order ω for Q, trade-off parameters $\{\epsilon_X\}_{X\in\mathcal{F}}$, and a database of size N. Let $\{s_X\}_{X\in\mathcal{F}}$ be the output of $\tau(\omega,\mathcal{F})$. As explained in Section 5 (Equation (5)), the enumeration delay of our approach is bounded by:

$$\max_{X\in\mathcal{F}} \max_{\mathcal{J}\subseteq\mathsf{JVSets}(Q_X)} \max_{\hat{Q}\in\mathsf{Con}_{\mathcal{J}}(Q_X)} |\pi_{\hat{\mathcal{F}}} S_{\mathcal{J}}(\hat{Q})|,$$

where $S_{\mathcal{J}}(\hat{Q})$ is the skew-aware query obtained from $\hat{Q}$ and $\hat{\mathcal{F}} = \mathit{free}(\hat{Q}) \cap \bigcup_{\mathcal{V}\in\mathsf{JVSets}(Q_X)} \mathcal{V}$.

We give bounds on the exponent e in the size complexity $\mathcal{O}(N^e)$ of $\pi_{\hat{\mathcal{F}}} S_{\mathcal{J}}(\hat{Q})$. One bound is the fractional edge cover number $\rho^*_{Q_X}(\hat{\mathcal{F}})$ [2]. Another bound is given by the preprocessing cost $p(\hat{Q},\epsilon_X)$ (Equation (1)) for $\hat{Q}$. The join v-sets of Q_X that are included in $\hat{\mathcal{F}}$ are heavy, which means that the number of distinct tuples over each of these join v-sets must be bounded $N^{1-\epsilon_X}$. This implies that e is bounded $k - k\epsilon_X$ where k is the minimal number of join v-sets covering all variables in $\hat{\mathcal{F}}$. The number k can be expressed by $\rho^*_{\mathsf{JVSets}(Q_X)}(\hat{\mathcal{F}})$. From the definition of the enumeration cost $e(\omega,\{\epsilon_X\}_{X\in\mathcal{F}})$ (Equation (7)), we conclude that e is bounded by $e = e(\omega,\{\epsilon_X\}_{X\in\mathcal{F}})$.

Gödel's Theorem Without Tears

Essential Incompleteness in Synthetic Computability

Dominik Kirst
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany

Benjamin Peters
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

Gödel published his groundbreaking first incompleteness theorem in 1931, stating that a large class of formal logics admits independent sentences which are neither provable nor refutable. This result, in conjunction with his second incompleteness theorem, established the impossibility of concluding Hilbert's program, which pursued a possible path towards a single formal system unifying all of mathematics. Using a technical trick to refine Gödel's original proof, the incompleteness result was strengthened further by Rosser in 1936 regarding the conditions imposed on the formal systems.

Computability theory, which also originated in the 1930s, was quickly applied to formal logics by Turing, Kleene, and others to yield incompleteness results similar in strength to Gödel's original theorem, but weaker than Rosser's refinement. Only much later, Kleene found an improved but far less well-known proof based on computational notions, yielding a result as strong as Rosser's.

In this expository paper, we work in constructive type theory to reformulate Kleene's incompleteness results abstractly in the setting of synthetic computability theory and assuming a form of Church's thesis, an axiom internalising the fact that all functions definable in such a setting are computable. Our novel, greatly condensed reformulation showcases the simplicity of the computational argument while staying formally entirely precise, a combination hard to achieve in typical textbook presentations. As an application, we instantiate the abstract result to first-order logic in order to derive essential incompleteness and, along the way, essential undecidability of Robinson arithmetic.

This paper is accompanied by a Coq mechanisation covering all our results and based on existing libraries of undecidability proofs and first-order logic, complementing the extensive work on mechanised incompleteness using the Gödel-Rosser approach. In contrast to the related mechanisations, our development follows Kleene's ideas and utilises Church's thesis for additional simplicity.

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

Keywords and phrases incompleteness, undecidability, synthetic computability theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.30

Supplementary Material To seamlessly integrate the mechanisation with the written text, each formal statement in the PDF version of this paper is hyperlinked with HTML documentation of the Coq files (signalled via a small Coq symbol).
InteractiveResource (Website): `https://www.ps.uni-saarland.de/extras/incompleteness`
Software (Source Code): `https://github.com/uds-psl/coq-synthetic-incompleteness`

1 Introduction

Shortly after Gödel published his celebrated completeness theorem of first-order logic [15, 17] in 1930, he discovered the surprising phenomenon of incompleteness [16] of sufficiently strong axiom systems. While completeness states that all valid formulas are provable, incompleteness (sometimes called negation-incompleteness for disambiguation) refers to the existence of independent sentences that are neither provable nor refutable from a given set of axioms. Considered from the programmatic perspective of metamathematics, completeness encouragingly entails that the formal method of syntactic, finitary deduction is an adequate

means to explore mathematical validities. In contrast, incompleteness establishes a principal limitation to axiomatic reasoning and therefore triggered a long tradition of interpretations (and sometimes misinterpretations [14]) in mathematics, philosophy, and even pop culture,[1] especially regarding the consequential observation that no such sufficiently strong axiom system can verify its own consistency (referred to as Gödel's second incompleteness theorem).

Concretely, Gödel showed that for all formal systems expressing enough properties of the natural numbers while being sound (i.e. all derivable arithmetical sentences are true for the standard model over $\mathbb{N}$) or at least ω-consistent (i.e. if $\varphi(\overline{n})$ is provable for all numerals $\overline{n}$, then $\exists x.\, \neg\varphi(x)$ is not provable) one can explicitly construct an independent sentence. For his elaborate construction, a lot of machinery regarding the arithmetisation of syntax and deduction systems as well as their interplay with substitution had to be developed, for instance Gödel numbering, the β-function, and the diagonal lemma. All this complexity obscures the underlying simple liar paradox of the constructed self-referential sentence, which is the reason why even full textbooks (e.g. Smith's monographs [44, 45]) are devoted to a formal exposition. Rosser later improved on the result by lifting the requirement of ω-consistency to plain consistency using a technically compact trick, entailing essential incompleteness meaning that independent sentences can be constructed in all consistent extensions of an incomplete system, but he still followed the same rather sophisticated strategy [42].

Only with the development of formal notions of computability and the resulting discovery of undecidability in 1936 by Church [4] and Turing [53], a much simpler proof strategy relying on a direct encoding of the halting problem was conceived, as directly remarked in Turing's paper. The underlying observation (already anticipated by Post, cf. [40]) is that complete axiom systems are decidable,[2] and thus systems able to express the halting problem and therefore inheriting its undecidability must be incomplete. To establish that a given system correctly expresses the halting problem, however, one typically relies on soundness to extract termination information from a formal derivation and, additionally, the proof does not readily yield a concrete independent sentence. Thus the nowadays well-known proof of incompleteness via undecidability, though elementary enough to be taught in basic courses on computability theory, yields a result even weaker than Gödel's original statement ahead of Rosser's refinement.

Far less well-known is the line of work pursued by Kleene [26, 27, 28, 29, 30], ultimately accomplishing a form of incompleteness as strong as Rosser's while still transparently showcasing the computational core of the argument.[3] Kleene's improved strategy is based on a switch from the encoded halting problem to encoding a pair of recursively inseparable sets via a stronger representability property, which is in turn established by a technique akin to Rosser's trick in [42]. By this switch the requirement of soundness instead of consistency can be avoided, since no termination information needs to be extracted from derivations but only existing derivations and refutations need to be preserved. Moreover, on more careful inspection already of the previous argument employing the halting problem, an explicit independent sentence can be extracted, similarly for the improved version. The only drawback of the computational variant of Gödel's first incompleteness theorem is that it no longer prepares the machinery for the second incompleteness theorem, but for the mere construction of independent sentences Kleene's argument seems superior and deserves wider popularity.

---

[1] See Douglas Hofstadter's classic "Gödel, Escher, Bach" [21] or far-reaching Youtube channels like Derek Muller's Veritasium (`https://www.youtube.com/watch?v=HeQX2HjkcNo`).

[2] Where we crucially consider the collection of axioms as enumerable, since the set of sentences satisfied in the standard model over $\mathbb{N}$ is a simple example of a complete but undecidable theory.

[3] For instance, a recent posting on the FOM mailing list (`https://cs.nyu.edu/pipermail/fom/2021-September/022872.html`) testified surprise "to discover such a proof laid out" in equally astonished blog posts and StackExchange threads.

Working in the constructive type theory CIC [5, 36], we translate Kleene's incompleteness proofs to the framework of synthetic computability of Richman and Bauer [41, 1], replacing the formal model of computation needed for the notions of enumerability and decidability by the implicit computation inherent to any intuitionistic meta-theory like CIC. Taking this perspective, Kleene's proofs can be further enhanced as no (often left informal) manipulation of Turing machines, μ-recursive functions, or untyped λ-terms is necessary to single out the computable functions $\mathbb{N} \to \mathbb{N}$. Instead, the necessary constructions can be (then directly formally) done with respect to all functions $\mathbb{N} \to \mathbb{N}$, as they are guaranteed to be computable by definability in our intuitionistic meta-theory. To enable the usual diagonalisation referring to universal machines for negative results, we assume variants of Church's thesis [31, 41, 9, 8], internalising the computability of all definable functions and inducing synthetic definitions of an undecidable halting problem and recursively inseparable sets.

With such a synthetic reformulation of Kleene's ideas, we contribute a strikingly simple yet fully formal proof of the strong Gödel-Rosser incompleteness theorem, isolating the computational essence at the core of the phenomenon. To this end, we first work with a fully abstract notion of formal systems to pin down their necessary properties and showcase the strategy free of any contingent overhead, an approach also followed by Beklemishev [2], Smullyan [46], Popescu and Traytel [38, 39], as well as Kirst and Hermes [23]. Subsequently, we instantiate the abstract development to the concrete case of first-order arithmetic, culminating in a proof of essential incompleteness of Robinson arithmetic Q, a finitely axiomatised fragment of Peano arithmetic PA. First, this conclusion is drawn, still maintaining the argument's simplicity, by assuming Church's thesis directly for Q as already employed by Hermes and Kirst [20]. Afterwards we replace this assumption by Church's thesis for μ-recursive functions and an application of the, naturally highly non-elementary, DPRM theorem [6, 33] to bring every μ-recognisable predicate into Diophantine and thus Q-expressible form.

On top of the mathematical contribution to formalise the computational incompleteness proofs in synthetic computability theory, especially the abstract proofs are straightforward to implement in the Coq proof assistant [49], suggesting that the chosen approach is well-suited for the notoriously hard mechanisation of incompleteness [43, 35, 19, 37, 39]. This approach was already exploited in [23], where only the weakest incompleteness result is derived from the undecidability of Q and PA. Following up on [23], the code for the abstract Gödel-Rosser theorem implemented as part of this paper spans merely about 200 lines, while the instantiation to Q adds roughly 2500 lines on top of the employed Coq libraries for first-order logic [24] and undecidability proofs [13]. The latter contains Larchey-Wendling and Forster's extensive mechanisation of the DPRM theorem [32], which could be replaced by a much weaker arithmetisation of a machine model to allow for a realistic comparison to the previous stand-alone mechanisations. Nevertheless, we deem it a valuable contribution to complement the extensive line of work regarding mechanisations of Gödel's original proof strategy with the first equally general mechanisation of the computational argument.

Outline. In Section 2 we summarise preliminary definitions and facts about constructive type theory, synthetic computability, and first-order logic. Then in the core technical part, we give synthetic and abstract proofs of the weak computational incompleteness theorem (Section 3) and Kleene's improvement (Section 4), the latter assuming a general form of Church's thesis. In Section 5, the abstract results are instantiated to Robinson arithmetic Q, assuming Church's thesis for Q to maintain a simple proof outline. Afterwards, this assumption is derived from a more conventional axiom referring to μ-recursive functions, now carrying out the core argument why Q can represent computation (Section 6). We close with some general remarks and further comments on related and future work in Section 7.

2 Preliminaries

In order to make this paper self-contained and accessible to a broader audience, we briefly outline the synthetic approach to computability theory and the representation of first-order logic in constructive type theory as used in prior work [10, 11, 25, 23].

2.1 Constructive Type Theory

We work in the framework of a constructive type theory such as CIC implemented in Coq, providing a predicative hierarchy of *type universes* above a single impredicative universe $\mathbb{P}$ of *propositions*. On type level, we have the unit type $\mathbb{1}$ with unique element $* : \mathbb{1}$, the void type $\mathbb{0}$, function spaces $X \to Y$, products $X \times Y$, sums $X + Y$, dependent products $\forall(x : X).\, F\, x$, and dependent sums $\Sigma(x : X).\, F\, x$. On propositional level, these types are denoted by logical notation ($\top$, $\bot$, $\to$, $\wedge$, $\vee$, $\forall$, and $\exists$). So-called *large elimination* from $\mathbb{P}$ into computational types is restricted, in particular case distinction on proofs of $\vee$ and $\exists$ to form computational values is disallowed. On the other hand, this restriction is permeable enough to allow large elimination of the equality predicate $=\colon \forall X.\, X \to X \to \mathbb{P}$ specified by the constructor $\forall(x : X).\, x = x$, as well as function definitions by well-founded recursion.

We further employ the basic inductive types of *Booleans* ($\mathbb{B} := \mathsf{tt} \mid \mathsf{ff}$), *Peano natural numbers* ($n : \mathbb{N} := 0 \mid n + 1$), as well as the *option type* ($\mathbb{O}(X) := \ulcorner x \urcorner \mid \emptyset$), and add further inductive types by need. Note that there is a canonical embedding of $\mathbb{B}$ into $\mathbb{N}$, encoding tt as 1 and ff as 0, which we sometimes use to interpret functions $X \to \mathbb{B}$ as functions $X \to \mathbb{N}$.

2.2 Synthetic Computability Theory

The base of the synthetic approach to computability theory of Richman and Bauer [41, 1] is the fact that all functions definable in an intuitionistic foundation are computable. This fact applies to many variants of constructive type theory and we let the assumed variant sketched in the previous section be one of those. Of course, we are confident that in particular the predicative calculus of cumulative inductive constructions (pCuIC) [51], the variant of CIC currently implemented in Coq, satisfies this condition although there is no formal proof yet.

As a basis we can introduce decidability, semi-decidability, and enumerability of decision problems synthetically, i.e. without reference to a formal model of computation (cf. [10]):

Definition 1. *Let $P : X \to \mathbb{P}$ be a predicate over a type X.*

- *P is* decidable *if there exists $d : X \to \mathbb{B}$ with $P\, x$ iff $d\, x = \mathsf{tt}$,*
- *P is* enumerable *if there exists $e : \mathbb{N} \to \mathbb{O}(X)$ with $P\, x$ iff $\exists n.\, e\, n = \ulcorner x \urcorner$,*
- *P is* semi-decidable *if there exists $s : X \to \mathbb{N} \to \mathbb{B}$ with $P\, x$ iff $\exists n.\, s\, x\, n = \mathsf{tt}$.*

On data types like $\mathbb{N}$, semi-decidability and enumerability coincide:

Fact 2. *Every predicate $P : \mathbb{N} \to \mathbb{P}$ is semi-decidable iff it is enumerable.*

Due to this fact, we will interchange both notions fluidly where appropriate to trigger different intuitions. In general, we prefer the view of semi-decidability as it harmonises with the much-used concept of partial functions.

Definition 3. *$f : X \to \mathbb{N} \to \mathbb{O}(Y)$ is a* partial function *if it is deterministic, i.e.:*

$$\forall x n n' y y'.\, f\, x\, n = \ulcorner y \urcorner \to f\, x\, n' = \ulcorner y' \urcorner \to y = y'$$

We write $f : X \rightharpoonup Y$ to denote that f is a partial function from X to Y. We write $f\, x \downarrow y$ if there is n with $f\, x\, n = \ulcorner y \urcorner$, $f\, x \downarrow$ if there is y with $f\, x \downarrow y$, and $f\, x \uparrow$ if $f\, x\, n = \emptyset$ for all n. The notation $f\, x \downarrow$ is meant to suggest termination while $f\, x \uparrow$ denotes divergence.

From every partial function $f : X \rightharpoonup Y$ that is total, i.e. satisfies $f\,x \downarrow$ for all x, one can extract a function $X \to Y$. Conversely, every function $X \to Y$ induces a total partial function $X \rightharpoonup Y$. We therefore freely change between both perspectives.

Finally, we introduce a notion of reductions capable of transporting decidability, foreshadowing the way how computational properties of formal systems can be expressed.

Definition 4. *Given predicates $P : X \to \mathbb{P}$ and $Q : Y \to \mathbb{P}$, we call a function $f : X \to Y$ a* (many-one) reduction *if $P\,x$ iff $Q\,(f\,x)$ for all x. We write $P \preceq Q$ if such a function exists.*

Fact 5. *If $P \preceq Q$ and Q is decidable, then so is P.*

Note that all these synthetic notions are only meaningful as long as no classical axioms jeopardising the computational interpretation of the function space $X \to Y$ are assumed. Instead, we will consider several axioms internalising the computational interpretation later.

2.3 First-Order Logic

The abstract incompleteness theorems discussed in Sections 3 and 4 make no reference to a concrete formalism, but the instantiation subject to Sections 5 and 6 will be based on first-order logic. We therefore summarise the representation of first-order terms and formulas with inductive types $\mathbb{T}$ and $\mathbb{F}$, respectively, as underlying [24]:

$$t, t' : \mathbb{T} \;::=\; x \mid O \mid S\,t \mid t \oplus t' \mid t \otimes t' \qquad (x : \mathbb{N})$$
$$\varphi, \psi : \mathbb{F} \;::=\; t \equiv t' \mid \dot{\bot} \mid \varphi \dot{\to} \psi \mid \varphi \dot{\wedge} \psi \mid \varphi \dot{\vee} \psi \mid \dot{\forall} x.\, \varphi \mid \dot{\exists} x.\, \varphi \qquad (x : \mathbb{N})$$

Given a number $n : \mathbb{N}$, we write $\overline{n}$ for the numeral $S^n\,O$. Given formulas φ and ψ, we let $\dot{\neg}\varphi$ denote $\varphi \dot{\to} \dot{\bot}$ and $\varphi \dot{\leftrightarrow} \psi$ denote $(\varphi \dot{\to} \psi) \dot{\wedge} (\psi \dot{\to} \varphi)$. We write $\varphi(x)$ to indicate that x is the only variable occurring free (i.e. not bound by a quantifier) in φ and $\varphi(t)$ to denote the usual capture-avoiding substitution of x with t, similarly for formulas with more free variables.

Axiom systems are represented as enumerable predicates $\mathcal{A} : \mathbb{F} \to \mathbb{P}$. We will consider the standard axiomatisation of Peano arithmetic PA, consisting of the defining equations for $\oplus$ and $\otimes$, injectivity of S, disjointness of S and O, as well as the induction scheme. The weaker system of Robinson arithmetic Q is obtained by replacing the induction scheme with a formula expressing case distinction.

Deduction systems are represented as inductive predicates of type $(\mathbb{F} \to \mathbb{P}) \to \mathbb{F} \to \mathbb{P}$ relating a context with a formula. Concretely, we use classical ($\vdash_c$) and intuitionistic ($\vdash_i$) natural deduction but since all presented results are agnostic to the particular flavour we simply write $\mathcal{A} \vdash \varphi$ standing for both. Since we assume axiomatisations $\mathcal{A}$ to be enumerable, their deductive closure denoted by $\mathcal{A}^{\vdash}$ can be shown enumerable, too.

A general representation of (Tarski) semantics is based on types $\mathcal{M}$ providing the structure to interpret the function symbols of the term language, giving rise to the recursive entailment relation $\mathcal{M} \vDash \varphi$ embedding formulas into propositions of the meta-logic. In this paper, we are exclusively concerned with the standard model $\mathcal{N}$ with $\mathbb{N}$ as domain and the natural interpretations of the function symbols. In this model, $\mathcal{N} \vDash \varphi$ evaluates to ordinary arithmetical statements and in particular $\mathcal{N} \vDash \mathsf{PA}$ can be shown. Note that for $\mathcal{N} \vDash \mathsf{PA}$ to hold constructively, it is crucial that we do not by default include classical axioms in PA [54].

In previous work [23], reductions from the solvability of Diophantine equations (H_{10}) as formalised by Larchey-Wendling and Forster [32] to arithmetical systems were verified. We recollect this fact to include the resulting weak form of incompleteness (already observed in [23]) as motivating approximation in the uniformised framework of this paper. Again note that without additional axioms a predicate like H_{10} cannot be shown undecidable in the synthetic sense but, given its actual undecidability, serves as a suitable computational taboo.

Fact 6. *There is a reduction witnessing $\mathsf{H}_{10} \preceq \mathsf{Q}^{\vdash}$ and $\mathsf{H}_{10} \preceq \mathsf{PA}^{\vdash}$.*

3 Synthetic and Abstract Approach to Incompleteness

In this and the next section, we develop incompleteness results of various strengths in a purely abstract setting. Our exposition follows the computational approach described by Kleene [29, 30], which we translate to the setting of synthetic computability to achieve a highly condensed but still fully formal presentation. We begin with the underlying notion of a formal system, involving only modest assumptions about sentences, negation, and provability.

Definition 7 (Formal System). *A triple $\mathcal{S} = (\mathbb{S}, \dot\neg, \vdash)$ is called a formal system if:*

- *$\mathbb{S}$ is a type, considered the* sentences *of $\mathcal{S}$,*
- *$\dot\neg : \mathbb{S} \to \mathbb{S}$ is a function on sentences, considered the* negation operation*,*
- *$\vdash\colon \mathbb{S} \to \mathbb{P}$ is a semi-decidable predicate on sentences, considered the* provable sentences*.*
- *Consistency holds in the form that for all $\varphi : \mathbb{S}$ not both $\vdash \varphi$ and $\vdash \dot\neg\varphi$.*

A formal system $\mathcal{S}' = (\mathbb{S}, \dot\neg, \vdash')$ is called an extension of $\mathcal{S}$ if $\vdash \varphi$ implies $\vdash' \varphi$ for all φ. Moreover, $\mathcal{S}$ is called decidable if the provability predicate $\vdash$ is decidable.

This general definition captures first-order axiomatisations as will be made precise in Section 5, but also applies to many other formalisms including constructive type theories like CIC or classical systems like HOL.

(Negation-)completeness can be easily expressed as a property of such formal systems, contrasting an informative notion of incompleteness relying on independent sentences.

Definition 8 (Completeness). *We call $\mathcal{S}$ complete if for all φ either $\vdash \varphi$ or $\vdash \dot\neg\varphi$. In contrast, $\mathcal{S}$ admits an independent sentence if there is φ with neither $\vdash \varphi$ nor $\vdash \dot\neg\varphi$.*

To obtain a first weak form of incompleteness, it suffices to observe that complete formal systems are decidable, therefore deciding every decision problem they can encode. This observation is an immediate consequence of Post's theorem [1, 10], however, we prefer to give an alternative proof employing a partial decider that will be reused later.

Lemma 9 (Partial Decider). *One can construct a partial function $d_{\mathcal{S}} : \mathbb{S} \rightharpoonup \mathbb{B}$ with:*

$$\forall\varphi.\ (\vdash \varphi \leftrightarrow d_{\mathcal{S}}\,\varphi \downarrow \mathsf{tt}) \ \wedge\ (\vdash \dot\neg\varphi \leftrightarrow d_{\mathcal{S}}\,\varphi \downarrow \mathsf{ff})$$

Note that by this specification $d_{\mathcal{S}}$ exactly diverges on the independent sentences of $\mathcal{S}$.

Proof. By the definition of formal systems, we have semi-deciders f_1 for $\lambda\varphi.\vdash\varphi$ and f_2 for $\lambda\varphi.\vdash\dot\neg\varphi$, where the latter is obtained from the former by testing if a given negation $\dot\neg\varphi$ is derivable, i.e. by $f_2\,\varphi := f_1\,(\dot\neg\varphi)$. Then we construct $d_{\mathcal{S}} : \mathbb{S} \rightharpoonup \mathbb{B}$ to be the (partial) function that on input φ simultaneously runs $f_1\,\varphi$ and $f_2\,\varphi$, returns tt if the former terminates and ff if the latter terminates, and diverges otherwise:

$$d_{\mathcal{S}}\,\varphi\,n \ :=\ \mathsf{if}\ f_1\,\varphi\,n\ \mathsf{then}\ \ulcorner\mathsf{tt}\urcorner\ \mathsf{else}\ \mathsf{if}\ f_2\,\varphi\,n\ \mathsf{then}\ \ulcorner\mathsf{ff}\urcorner\ \mathsf{else}\ \emptyset$$

Consistency is used as the crucial property to show that this function is deterministic. ◀

Now the connection of completeness and decidability can be established transparently:

Fact 10 (Decidability). *If $\mathcal{S}$ is complete, then it is decidable.*

Proof. By completeness the partial decider $d_{\mathcal{S}}$ is total, inducing a decider $\mathbb{S} \to \mathbb{B}$. ◀

To derive said weak form of incompleteness, it remains to clarify what it means for a formal system to encode a decision problem. An intuitive characterisation, called weak representability, exhibits the structure of many-one reductions.

Definition 11 (Weak Representability). *$\mathcal{S}$ weakly represents $P : X \to \mathbb{P}$ if $P \preceq \mathcal{S}$, i.e. if there is a function $r : X \to \mathbb{S}$ such that $P\,x \leftrightarrow \vdash r\,x$. If only $\vdash r\,x$ implies $P\,x$, then we call $\mathcal{S}$ sound for P and r (or simply sound for P if we leave r implicit).*

We can now derive incompleteness in the sense that systems weakly representing an undecidable problem cannot be complete. As the property of weak representability is preserved along sound extensions, we instantiate this result later to derive weak incompleteness of PA and other axiomatisations sound for $\mathcal{N}$.

Theorem 12 (Weak Incompleteness). *If $\mathcal{S}$ weakly represents $P : X \to \mathbb{P}$, then for any extension $\mathcal{S}'$ of $\mathcal{S}$ sound for P it holds that if $\mathcal{S}'$ is complete, then P is decidable. Therefore, if P is known to be undecidable, then $\mathcal{S}'$ must be incomplete.*

Proof. Note that any sound extension $\mathcal{S}'$ of $\mathcal{S}$ still weakly represents P. Since completeness induces decidability of $\vdash$ (Fact 10), we obtain decidability of P from Fact 5. ◀

4 Improving the Computational Incompleteness Result

Although Theorem 12 correctly identifies the computational essence of incompleteness, namely the connection to undecidability, it still falls short of the stronger Gödel-Rosser theorem:

1. The reliance on weak representability excludes consistent but unsound extensions and hence, for instance, essential incompleteness of Q cannot be achieved.
2. There is no concrete example of an independent sentence constructed since the global completeness assumption is needed to totalise the partial decider $d_{\mathcal{S}}$.
3. The result is presented only up to a computational taboo, i.e. the decidability of a problem known to be undecidable, instead of an actual contradiction.

In this section, we address these shortcomings one-by-one, yielding the strongest form of incompleteness possible. Regarding the third improvement, the only way to derive a contradiction from a computation taboo is to assume an axiom that restricts the ambient constructive type theory to a computational interpretation. Concretely, we now assume a variant of Church's thesis [31], namely EPF for "enumerability of partial functions" [41, 9, 8]. It postulates a universal function $\Theta : \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N})$ computing all partial functions, i.e. for every $f : \mathbb{N} \rightharpoonup \mathbb{N}$ there is a code c such that Θ_c agrees with f (extensionally).

Axiom 13 (EPF). *There is a universal function $\Theta : \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N})$ satisfying:*

$$\forall f : \mathbb{N} \rightharpoonup \mathbb{N}.\, \exists c : \mathbb{N}.\, \forall xy.\, \Theta_c\, x \downarrow y \leftrightarrow f\, x \downarrow y$$

This assumption induces a canonical undecidable problem:

Definition 14 (Halting Problem). *We define the self-halting problem by $\mathsf{K}_\Theta\, x := \Theta_x\, x \downarrow$.*

The self-halting problem for Θ can be easily shown undecidable by the usual diagonalisation argument. Following this argument in a constructively more informative way, we show that every potential decider for K_Θ necessarily diverges on a concretely constructed input.

Fact 15. *K_Θ is enumerable, but for every candidate decider $d : \mathbb{N} \rightharpoonup \mathbb{B}$ with*

$$\forall x.\, \mathsf{K}_\Theta\, x \leftrightarrow d\, x \downarrow \mathsf{tt}$$

one can construct a concrete value x with $\neg \mathsf{K}_\Theta\, x$ such that $d\, x \uparrow$.

Proof. We first define the partial function $f : \mathbb{N} \rightharpoonup \mathbb{B}$ such that $f\,x \downarrow \mathsf{tt}$ whenever $d\,x \downarrow \mathsf{ff}$ and $f\,x \uparrow$ otherwise. Now using EPF we obtain a code c for f and deduce for $x := c$ that

$$d\,x \downarrow \mathsf{tt} \;\Leftrightarrow\; \mathsf{K}_\Theta\,x \;\Leftrightarrow\; \Theta_x\,x \downarrow \Leftrightarrow\; f\,x \downarrow \Leftrightarrow\; f\,x \downarrow \mathsf{tt} \;\Leftrightarrow\; d\,c \downarrow \mathsf{ff}$$

from which we conclude $d\,x \uparrow$. That K_Θ is not decidable follows since every decider $\mathbb{N} \to \mathbb{B}$ would induce a total candidate decider $\mathbb{N} \rightharpoonup \mathbb{B}$. Finally, enumerability of K_Θ is standard. ◀

We can now identify an intermediate refinement of the incompleteness theorem, providing a concrete independent sentence up to an actual contradiction, which corresponds to the result originally shown by Gödel (in the semantic form requiring soundness instead of ω-consistency).

Theorem 16 (Gödel's Incompleteness)**.** *If $\mathcal{S}$ weakly represents K_Θ, then any extension $\mathcal{S}'$ of $\mathcal{S}$ sound for K_Θ admits an independent sentence.*

Proof. Let $r : \mathbb{N} \to \mathbb{S}$ weakly represent K_Θ in $\mathcal{S}$, therefore also in all sound extensions $\mathcal{S}'$. The function $d := d_{\mathcal{S}'} \circ r$ is a candidate decider for K_Θ in the sense of Fact 15 since:

$$\mathsf{K}_\Theta\,x \;\Leftrightarrow \vdash r\,x \;\Leftrightarrow\; d_{\mathcal{S}'}(r\,x) \downarrow \mathsf{tt} \;\Leftrightarrow\; d \downarrow \mathsf{tt}$$

Then by Fact 15 there is a particular x with $d\,x \uparrow$ and we observe that the sentence $r\,x$ can neither be provable nor refutable since in either case $d\,x \downarrow$ by specification of $d_{\mathcal{S}'}$. ◀

In order to tackle the remaining improvement, namely the applicability to consistent extensions, we follow Kleene's idea to switch to a stronger notion of representability that is not affected by unsound formal systems. Since for weak representability of a predicate P it was crucial to obtain $P\,x$ from $\vdash r\,x$, so to extract information from a derivation, one might hope that this can be replaced by a proof of $\vdash \dot{\neg}(r\,x)$ from $\neg P\,x$, as this has a derivation in the conclusion and therefore transports along any extension. Unfortunately, this strong notion of representability can only be achieved for decidable predicates, thus ruling out the encoding of the undecidable K_Θ for a contradiction. However, it is possible to specify a very similar notion involving a second predicate Q, such that still all derivations appear in conclusions but P and Q can be instantiated with undecidable problems, respectively.

Definition 17 (Strong Separability)**.** *$\mathcal{S}$ strongly separates $P : X \to \mathbb{P}$ and $Q : X \to \mathbb{P}$ if there is a function $r : X \to \mathbb{S}$ such that $P\,x$ implies $\vdash r\,x$ and $Q\,x$ implies $\vdash \dot{\neg} r\,x$.*

The notion of strong separability can now be instantiated with any pair of recursively inseparable problems (i.e. problems excluding any total decider discriminating them) to derive essential incompleteness. The canonical pair of such recursively inseparable problems in the context of EPF refers to the self-halting problems for specific output.

Definition 18. *We define the problems $\mathsf{K}^1_\Theta\,x := \Theta_x\,x \downarrow 1$ and $\mathsf{K}^0_\Theta\,x := \Theta_x\,x \downarrow 0$.*

As done with the normal self-halting problem before (Fact 15), we do not just refute any discriminating decider but show that every partial decider actually diverges on an explicitly constructed input.

Fact 19. *K^1_Θ and K^0_Θ are enumerable, but for every candidate separator $s : \mathbb{N} \rightharpoonup \mathbb{B}$ with*

$$\forall x.\; (\mathsf{K}^1_\Theta\,x \to s\,x \downarrow \mathsf{tt}) \;\wedge\; (\mathsf{K}^0_\Theta\,x \to s\,x \downarrow \mathsf{ff})$$

one can construct a concrete value x with $\neg\mathsf{K}^1_\Theta\,x$ and $\neg\mathsf{K}^0_\Theta\,x$ such that $s\,x \uparrow$.

Proof. We define the partial function $f : \mathbb{N} \rightharpoonup \mathbb{B}$ such that $f\,x \downarrow \mathsf{ff}$ if $s\,x \downarrow \mathsf{tt}$, $f\,x \downarrow \mathsf{tt}$ if $s\,x \downarrow \mathsf{ff}$, and $f\,x \uparrow$ otherwise. Using EPF we obtain a code c for f and deduce for $x := c$ that

$$s\,x \downarrow \mathsf{tt} \;\Leftrightarrow\; f\,x \downarrow \mathsf{ff} \;\Leftrightarrow\; \Theta_x\,x \downarrow 0 \;\Leftrightarrow\; \mathsf{K}^0_\Theta\,x \;\Rightarrow\; s\,x \downarrow \mathsf{ff}$$
$$s\,x \downarrow \mathsf{ff} \;\Leftrightarrow\; f\,x \downarrow \mathsf{tt} \;\Leftrightarrow\; \Theta_x\,x \downarrow 1 \;\Leftrightarrow\; \mathsf{K}^1_\Theta\,x \;\Rightarrow\; s\,x \downarrow \mathsf{tt}$$

from which we conclude $s\,x \uparrow$. Again, enumerability of K^1_Θ and K^0_Θ is standard. ◀

The desired strong incompleteness theorem, now corresponding to Rosser's refinement of Gödel's result, follows for all formal systems that capture enough computation to strongly separate K^1_Θ and K^0_Θ.

Theorem 20 (Gödel-Rosser Incompleteness). *If $\mathcal{S}$ strongly separates K^1_Θ and K^0_Θ, then any extension $\mathcal{S}'$ of $\mathcal{S}$ admits an independent sentence, i.e. $\mathcal{S}$ is essentially incomplete.*

Proof. Let $r : \mathbb{N} \to \mathbb{S}$ strongly separate K^1_Θ and K^0_Θ in $\mathcal{S}$, therefore also in all consistent extensions $\mathcal{S}'$. The function $s := d_{\mathcal{S}'} \circ r$ is a candidate separator for K^1_Θ and K^0_Θ since:

$$\mathsf{K}^1_\Theta\,x \;\Rightarrow\; \vdash r\,x \quad\Leftrightarrow\; d_{\mathcal{S}'}(r\,x) \downarrow \mathsf{tt} \;\Leftrightarrow\; s \downarrow \mathsf{tt}$$
$$\mathsf{K}^0_\Theta\,x \;\Rightarrow\; \vdash \dot\neg r\,x \;\Leftrightarrow\; d_{\mathcal{S}'}(r\,x) \downarrow \mathsf{ff} \;\Leftrightarrow\; s \downarrow \mathsf{ff}$$

Then by Fact 19 there is a particular x with $s\,x \uparrow$ and we observe that the sentence $r\,x$ can neither be provable nor refutable since in either case $s\,x \downarrow$ by specification of $d_{\mathcal{S}'}$. ◀

To emphasise the connection with computational incompleteness, we observe essential undecidability of formal systems of the same expressivity as required in Theorem 20.

Theorem 21 (Essential Undecidability). *If $\mathcal{S}$ strongly separates K^1_Θ and K^0_Θ, then any extension $\mathcal{S}'$ of $\mathcal{S}$ is undecidable, i.e. $\mathcal{S}$ is essentially undecidable.*

Proof. Given $r : \mathbb{N} \to \mathbb{S}$ strongly separating K^1_Θ and K^0_Θ and $d : \mathbb{S} \to \mathbb{B}$ deciding $\mathcal{S}'$, the (total) function $s := d \circ r$ would recursively separate K^1_Θ from K^0_Θ, contradicting Fact 19. ◀

5 Essential Incompleteness of Robinson Arithmetic

We next instantiate the abstract approach to incompleteness from the previous sections to the case of first-order arithmetic. To this end, we now make precise that every consistent axiomatisation $\mathcal{A}$ induces a formal system $\mathcal{S}_\mathcal{A} = (\mathbb{S}_\mathcal{A}, \dot\neg_\mathcal{A}, \vdash_\mathcal{A})$ where

- $\mathbb{S}_\mathcal{A}$ is the type of closed formulas $\varphi : \mathbb{F}$,
- $\dot\neg_\mathcal{A}$ is the negation function $\dot\neg\varphi$ restricted to closed φ,
- $\vdash_\mathcal{A}$ is the provability predicate $\mathcal{A} \vdash \varphi$ restricted to closed φ, and
- $\vdash_\mathcal{A} \varphi$ simultaneous to $\vdash_\mathcal{A} \dot\neg\varphi$ is ruled out by the consistency of $\mathcal{A}$.

We then say that $\mathcal{A}$ is complete if its induced formal system $\mathcal{S}_\mathcal{A}$ is complete, i.e. if either $\mathcal{A} \vdash \varphi$ or $\mathcal{A} \vdash \dot\neg\varphi$ for all closed φ. Similarly, we say that $\mathcal{A}$ admits an independent sentence if $\mathcal{S}_\mathcal{A}$ does, i.e. if there is some closed φ with neither $\mathcal{A} \vdash \varphi$ nor $\mathcal{A} \vdash \dot\neg\varphi$. Note that here, as our notational convention suggests, we deliberately include both the intuitionistic and the classical ND system, so our treatment of incompleteness applies to both flavours.

Since reductions $P \preceq \mathcal{A}^\vdash$ establish that $\mathcal{S}_\mathcal{A}$ weakly represents P, we can immediately derive a weak form of incompleteness from previous results.

Theorem 22 (Weak Incompleteness, cf. [23]). *If PA is complete, then H_{10} is decidable.*

Proof. We have $\mathsf{H}_{10} \preceq \mathsf{PA}^{\vdash}$ by Fact 6, so $\mathcal{S}_{\mathsf{PA}}$ weakly represents H_{10}. Then if PA were complete, H_{10} were decidable by Theorem 12. ◀

Note that this result also applies to all sound extensions of PA, i.e. extensions $\mathcal{A}$ such that from $\mathcal{A} \vdash \varphi$ one can derive $\mathcal{N} \vDash \varphi$, as well as to all weaker (and hence vacuously incomplete) fragments, in particular Q. We refer the reader to [23] for more detail on this weak form of incompleteness obtained from the reduction of H_{10} in a synthetic sense.

To obtain the stronger result concerning merely consistent extensions, we prepare to instantiate Theorem 20 to the case of Q, as this axiomatisation exactly provides the needed representability requirements. For this instantiation, note that although EPF is an axiom strong enough to yield undecidable problems, it does not necessarily restrict the function space $\mathbb{N} \rightharpoonup \mathbb{N}$ to a concrete model of computation expressible in Q. We therefore need to assume a more explicit form of Church's thesis to derive the desired representability within Q. An elegant strategy is to directly assume Church's thesis for Q itself (CT_{Q}) as introduced by Hermes and Kirst [20], instantiate Theorem 20 with elementary arguments, and afterwards deliver the rather involved argument that CT_{Q} follows from a more conventional explicit form of EPF for μ-recursive functions.

To state CT_{Q}, we first identify the semantically well-behaved class of Σ_1-formulas.

Definition 23 (Δ_1- and Σ_1-formulas, cf. [20]). *We say that $\varphi : \mathbb{F}$ is a Δ_1-formula if for all substitutions σ such that $\sigma\, n$ is closed for all $n : \mathbb{N}$ we have $\mathsf{Q} \vdash \varphi[\sigma]$ or $\mathsf{Q} \vdash \dot{\neg}\varphi[\sigma]$. Moreover, we say that $\psi : \mathbb{F}$ is a Σ_1-formula if there is a Δ_1-formula ψ such that $\varphi = \dot{\exists} \ldots \dot{\exists}\psi$.*

CT_{Q} then states that any function $\mathbb{N} \rightharpoonup \mathbb{N}$ is fully captured by a Σ_1-formula.

Axiom 24 (CT_{Q}). *For all partial $f : \mathbb{N} \rightharpoonup \mathbb{N}$ there exists a Σ_1-formula $\varphi(x, y)$ with:*

$$\forall xy.\, f\, x \downarrow y \;\leftrightarrow\; \mathsf{Q} \vdash \dot{\forall} y'.\, \varphi(\overline{x}, y') \dot{\leftrightarrow} y' \equiv \overline{y}$$

To enable the usage of the results from the previous section solely assuming CT_{Q}, we show that CT_{Q} yields a universal function Θ as formerly postulated with EPF.

Fact 25. *Given that we now assume CT_{Q}, in particular EPF holds.*

Proof. We choose as universal function $\Theta : \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N})$ the partial function that on input c and x enumerates all derivations from Q and terminates with value y if a derivation $\mathsf{Q} \vdash \forall y'.\, \varphi_c(\overline{x}, y') \dot{\leftrightarrow} y' \equiv \overline{y}$ is found for φ_c being the c-th formula.

Then given a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$, the assumption of CT_{Q} guarantees that f is captured by some Σ_1-formula $\varphi = \varphi_c$ for some c. Then we deduce for all x and y

$$\Theta_c\, x \downarrow y \;\Leftrightarrow\; \mathsf{Q} \vdash \forall y'.\, \varphi_c(\overline{x}, y') \dot{\leftrightarrow} y' \equiv \overline{y} \;\Leftrightarrow\; f\, x \downarrow y$$

as desired to establish that Θ is universal. ◀

In the case of total functions, the capturing condition can be slightly simplified, which yields the actual formulation of CT_{Q} used in [20].

Fact 26 (Total CT_{Q}, cf. [20]). *For all $f : \mathbb{N} \to \mathbb{N}$ there exists a Σ_1-formula $\varphi(x, y)$ with:*

$$\forall x.\, \mathsf{Q} \vdash \dot{\forall} y'.\, \varphi(\overline{x}, y') \dot{\leftrightarrow} y' \equiv \overline{f\, x}$$

From CT_{Q} we can derive all the representability conditions employed in Section 3. In fact, we obtain more precise conditions involving Σ_1-formulas $\varphi(x)$ providing uniform encoding functions $r\, n := \varphi(\overline{n})$.

Definition 27. *Given $P, P' : \mathbb{N} \to \mathbb{P}$ and a Σ_1-formula $\varphi(x)$ we say that*

- *φ weakly Σ_1-represents P if $P\,n \leftrightarrow \mathsf{Q} \vdash \varphi(\overline{n})$ and*
- *φ strongly Σ_1-separates P and P' if $P\,n \to \mathsf{Q} \vdash \varphi(\overline{n})$ and $P'\,n \to \mathsf{Q} \vdash \dot{\neg}\varphi(\overline{n})$.*

So if φ for instance Σ_1-represents $P : \mathbb{N} \to \mathbb{P}$, then $r\,n := \varphi(\overline{n})$ witnesses that the system $\mathcal{S}_{\mathsf{Q}}$ weakly represents P in the sense of Definition 11, analogously for strong Σ_1-separability.

Theorem 28 (Representability, cf. [20])**.** *Q can represent predicates as follows:*

1. *Every enumerable predicate over $\mathbb{N}$ is weakly Σ_1-representable.*
2. *Every pair of disjoint enumerable predicates over $\mathbb{N}$ is strongly Σ_1-separable.*

Proof. We establish both claims independently:

1. An enumerator e of P can be recast as a function $\mathbb{N} \to \mathbb{N}$ with $P\,x$ iff $\exists n.\, e\,n = x + 1$. Applying CT_{Q}, we obtain a Σ_1-formula φ capturing e and deduce:

$$P\,x \;\Leftrightarrow\; \exists n.\, e\,n = x + 1 \;\Leftrightarrow\; \exists n.\, \mathsf{Q} \vdash \overline{e\,n} = S\,\overline{x} \;\Leftrightarrow\; \exists n.\, \mathsf{Q} \vdash \varphi(\overline{n}, S\,\overline{x}) \;\Leftrightarrow\; \mathsf{Q} \vdash \dot{\exists} k.\, \varphi(k, S\,\overline{x})$$

   Thus $\psi(x) := \exists k.\, \varphi(k, S\,x)$ weakly Σ_1-represents P.
2. A partial decider $d : \mathbb{N} \rightharpoonup \mathbb{B}$ can be constructed with $P\,x$ iff $d\,x \downarrow \mathsf{tt}$, and $P'\,x$ iff $d\,x \downarrow \mathsf{ff}$, analogously to the partial decider defined in Lemma 9. Applying CT_{Q}, we obtain a Σ_1-formula φ capturing d and deduce:

$$\begin{aligned} P\,x \;&\Rightarrow\; d\,x \downarrow \mathsf{tt} \;\Rightarrow\; \mathsf{Q} \vdash \varphi(x, \overline{1}) \\ P'\,x \;&\Rightarrow\; d\,x \downarrow \mathsf{ff} \;\Rightarrow\; \mathsf{Q} \vdash \varphi(x, \overline{0}) \;\Rightarrow\; \mathsf{Q} \vdash \dot{\neg}\varphi(x, \overline{1}) \end{aligned}$$

   Thus $\psi(x) := \varphi(x, \overline{1})$ strongly Σ_1-separates P and P'. ◀

Note that the weak representability property (1) of Theorem 28 could be used to obtain independent sentences for all sound extensions of Q based on the intermediate result Theorem 16. Already given the strong separability property (2), however, we immediately conclude the stronger essential incompleteness of Q based on Theorem 20.

Theorem 29. *Any consistent axiomatisation $\mathcal{A} \supseteq \mathsf{Q}$ admits an independent sentence.*

Proof. We apply Theorem 20, so we only need to show that Q strongly separates K^1_Θ and K^0_Θ. Since these are enumerable, this follows from (2) of Theorem 28. ◀

Similarly, we can observe the essential undecidability of Q based on Theorem 21.

Theorem 30. *Any consistent axiomatisation $\mathcal{A} \supseteq \mathsf{Q}$ is undecidable.*

Proof. We apply Theorem 21 and then argue as in the proof of Theorem 29. ◀

6 Deriving Church's Thesis for Robinson Arithmetic

Arguably, by the assumption of CT_{Q} we have sidestepped much of the actual work needed to establish the essential incompleteness of Q. To showcase that most of this work concerned with the representability properties can actually be done feasibly and only an axiom connecting the synthetic level with a concrete model of computation is necessary, we now derive CT_{Q} from a common version of Church's thesis for μ-recursive functions (EPF_μ). Note that Church's thesis for any Turing complete model could be consistently assumed as discussed by Forster [9] and thus by the upcoming derivation we in particular justify the consistency of CT_{Q}. We also remark that our derivation relies on the heavy-weight DPRM theorem as mechanised by Larchey-Wendling and Forster [32], however, one could also give a less informative but more direct arithmetisation of formal computation.

We refer to [32] for full detail about an encoding of μ-recursive functions in CIC and only require a step-indexed interpreter $\Theta^\mu : \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N})$. For Θ^μ we then state EPF_μ which will only be used to show that the graph of a given partial function is μ-enumerable, and therefore Diophantine by the DPRM theorem.

Definition 31. *EPF_μ states that Θ^μ is universal for all partial functions:*

$$\forall f : \mathbb{N} \rightharpoonup \mathbb{N}.\, \exists c : \mathbb{N}.\, \forall xy.\, \Theta^\mu_c\, x \downarrow y \leftrightarrow f\, x \downarrow y$$

To prepare the result that EPF_μ implies CT_Q, we need a bit more machinery about Σ_1-formulas φ, especially the completeness property that for deriving $\mathsf{Q} \vdash \varphi$ it suffices to show $\mathcal{N} \vDash \varphi$. This and forthcoming observations can be simplified by the fact that a prefix of existential quantifiers can be compressed into a single existential quantifier:

Lemma 32. *For every Σ_1-formula φ there is a Δ_1-formula ψ with $\mathsf{Q} \vdash \varphi \dot{\leftrightarrow} \dot{\exists}\psi$.*

Proof. By induction on the length of the quantifier prefix of φ. For the inductive step it suffices to show that two quantifiers can be merged into one, i.e. that for a given Δ_1-formula φ there is a Δ_1-formula ψ with $\mathsf{Q} \vdash (\dot{\exists}x.\, \dot{\exists}y.\, \varphi(x,y)) \dot{\leftrightarrow} (\dot{\exists}z.\, \psi(z))$. We set:

$$\psi(z) \;:=\; \dot{\exists}x.\, (\dot{\exists}k.\, z \equiv x \oplus k) \mathbin{\dot{\wedge}} \dot{\exists}y.\, (\dot{\exists}k.\, z \equiv k \oplus y) \mathbin{\dot{\wedge}} \varphi(x,y)$$

The sought equivalence is not hard to establish as one can instantiate $z := x \oplus y$. Proving that ψ is Δ_1 is more tedious but less insightful as this requires to establish decidability of bounded quantifications via their equivalence to iterated disjunctions formally in Q. ◀

Note that from now on we use $x \dot{\leq} y$ as the common notation for $\dot{\exists}k.\, y \equiv x \oplus k$ but that we indeed also need to employ the symmetric variant $\dot{\exists}k.\, y \equiv k \oplus x$ in the previous proof since Q does not recognise addition as commutative.

Fact 33 (Σ_1-completeness, cf. [20])**.** *If φ is closed and Σ_1, then $\mathcal{N} \vDash \varphi$ implies $\mathsf{Q} \vdash \varphi$.*

Proof. By Lemma 32 we may assume that φ has the form $\dot{\exists}\psi$ where ψ is Δ_1. Then from $\mathcal{N} \vDash \varphi$ we obtain $n : \mathbb{N}$ such that $\mathcal{N} \vDash \psi(\overline{n})$. Now since $\psi(\overline{n})$ is closed we have either $\mathsf{Q} \vdash \psi(\overline{n})$ or $\mathsf{Q} \vdash \dot{\neg}\psi(\overline{n})$ by the definition of Δ_1, where the former immediately yields $\mathsf{Q} \vdash \varphi$ and where the latter contradicts $\mathcal{N} \vDash \varphi$ via soundness. ◀

We can now give a proof that EPF_μ implies CT_Q based on a technique resembling Rosser's trick in his refinement of Gödel's original incompleteness proof. To provide some intuition, the idea is to refine a formula weakly Σ_1-representing a predicate such that a witness not only guarantees a solution but also that all potential smaller solutions show similar behaviour.

Fact 34. *EPF_μ implies CT_Q.*

Proof. Let $f : \mathbb{N} \rightharpoonup \mathbb{N}$ be given, the goal is to capture f by some Σ_1-formula φ. From EPF_μ we obtain some c such that f is computed by Θ^μ_c. Now since Θ^μ_c is μ-recursive, we can apply the DPRM theorem to obtain a polynomial equation $p = q$ recognising the graph of Θ^μ_c. From the reduction verified in [23] we obtain that solvability of $p = q$ agrees with derivability of $\varphi_{p,q} = \dot{\exists}^N\, p^* \equiv q^*$ in Q:

$$f\, x \downarrow y \;\leftrightarrow\; \mathsf{Q} \vdash \varphi_{p,q}(\overline{x}, \overline{y})$$

This intermediate result states that the graph of f is weakly Σ_1-representable and can be refined to a capturing as needed in CT_Q using a general variant of Rosser's trick. First, with Lemma 32 we refine $\varphi_{p,q}(x,y)$ to a formula $\dot{\exists}k.\, \psi(x,y,k)$ where ψ is Δ_1. Secondly, we set

$$\varphi'(x,y,k) \;:=\; \psi(x,y,k) \mathbin{\dot{\wedge}} \dot{\forall}y'k'.\, y' \oplus k' \dot{\leq} y \oplus k \mathbin{\dot{\to}} \psi(x,y',k') \mathbin{\dot{\to}} y' \equiv y$$

followed by $\varphi(x,y) := \dot{\exists}k.\, \varphi'(x,y,k)$ and verify that φ captures f as desired for CT_Q:

- Assuming $f\,x \downarrow y$, we want to derive $\dot{\forall} y'.\, \varphi(\overline{x}, y') \dot{\leftrightarrow} y' \equiv \overline{y}$ formally within Q. Note that from $f\,x \downarrow y$ we obtain some natural number k with $\psi(\overline{x}, \overline{y}, \overline{k})$ as base. Using Σ_1-completeness, we can in fact derive $\varphi'(\overline{x}, \overline{y}, \overline{k})$ as this is straightforward to verify in the standard model $\mathcal{N}$.
  This establishes the backwards direction of the sought equivalence, for the forward direction assume $\varphi(\overline{x}, y')$ for some variable y'. Hence $\varphi'(\overline{x}, y', k')$ for some variable k', complementing $\varphi'(\overline{x}, \overline{y}, \overline{k})$ from before. As Q can derive that either $\overline{y} \oplus \overline{k} \leq y' \oplus k'$ or $y' \oplus k' \leq \overline{y} \oplus \overline{k}$, we obtain $y' \equiv \overline{y}$ in either case from the construction of φ'.
- If conversely $\mathsf{Q} \vdash \dot{\forall} y'.\, \varphi(\overline{x}, y') \dot{\leftrightarrow} y' \equiv \overline{y}$, then in particular $\mathsf{Q} \vdash \dot{\exists} k.\, \psi(\overline{x}, \overline{y}, k)$ from which we obtain $f\,x \downarrow y$ by the representability property of $\varphi_{p,q}$. ◀

In fact, we also expect that CT_{Q} implies EPF_{μ} as this basically boils down to the same proof as in Fact 25, with the difference that all computability arguments are done for μ-recursive functions instead of synthetically.

7 Discussion

In this paper, we first gave generic incompleteness proofs of different strengths for abstract formal systems with a negation operation, translating ideas of Kleene to the framework of synthetic computability. The strongest version states essential incompleteness of formal systems strongly separating canonical enumerable and disjoint predicates. Secondly, we instantiated our results to first-order logic over the axiomatisation of Robinson arithmetic Q. The instantiation was first approximated assuming CT_{Q} and then using EPF_{μ}, the DPRM theorem, and Rosser's trick to show strong Σ_1-separability of disjoint enumerable predicates.

The remaining assumption of EPF_{μ} is a common formulation of Church's thesis, already mentioned as a consistent axiom for constructive mathematics in the textbook by Troelstra and van Dalen [52]. Though no consistency proof for the specific case of EPF_{μ} in CIC has been conducted, equivalent formulations of Church's thesis have been shown consistent in closely related type theories by Swan and Uemura [47] and Yamada [55], see also Forster's discussion [9] for an overview of formulations of Church's thesis in CIC.

7.1 Coq Mechanisation

The mechanisation consists of two main parts: the abstract incompleteness proofs and their instantiation to first-order logic. The former consists of roughly 400 lines of code, of which only around 200 are required for the strongest incompleteness proofs, while the latter consists of around 2500 lines of code. The development is based on Coq libraries of undecidability proofs [13] and first-order logic [24], from which code particularly on synthetic computability, the DPRM theorem, as well as the encoding of first-order logic is reused, respectively.

Mechanising and working with partial functions and Church's thesis is straightforward. The paper proofs, however, tend to follow a slightly different structure than their mechanised counterparts, in particular when dealing with equivalences, such as in Fact 15. Otherwise, the mechanisation of Sections 3 and 4 is remarkably unremarkable.

Mechanising the instantiation to first-order logic, however, was a lot more work. We build upon an existing mechanisation of first-order logic by Kirst et al. [24] that includes most fundamental definitions and lemmas for working with first-order logic. As opposed to the definitions presented in this text, it defines formulas and terms to be parametric in a signature, i.e. types of predicate and function symbols with their corresponding arities, and uses de Bruijn indices instead of explicit naming to implement binding. While the former

difference did not affect the mechanisation other than requiring some boilerplate code, the latter repeatedly caused us problems. Mechanising structures that include binders, such as predicate logic or programming languages, is well known to be much more tedious than dealing with them on paper, where many lemmas on and properties of substitutions are largely glossed over.

Notably, a lot of work (almost half of the mechanisation of the instantiation, by lines of code) went into mechanising Q-decidability of bounded quantification and Σ_1-completeness due to the technicality of these results. These proofs relied heavily on the first-order proof mode for Coq by Koch, as described in [22], allowing us to use tactics similar to the ones included with Coq to show statements within first-order logic. The proof mode also provides translations between a de Bruijn representation of logical formulas and a named representation, which greatly improves the ergonomics of working with first-order logic. This project would have been much more tedious if we did not have the proof mode available.

7.2 Related Work

Variants of Gödel's incompleteness theorems. The Gödel-Rosser approach to incompleteness was developed in the 1930s, primarily by Gödel [16] and Rosser [42]. Kleene presented his approach to incompleteness prominently in both of his books [29, 30], as well as multiple papers [26, 27, 28, 29, 30]. Turing mentioned similar ideas to show incompleteness in his seminal paper on the *Entscheidungsproblem* [53].

Different proofs of Gödel's first incompleteness theorem, among them some abstract ones, have been considered by Beklemishev [2], Smullyan [46], as well as Popescu and Traytel [39]. Our approach especially shares similarities with the former two, as they also consider Kleene's computational proofs in an abstract setting, while the latter approach is mechanised but based on the Gödel-Rosser strategy. Another computational account of Gödel's incompleteness theorem was anticipated independently by Post [40].

Synthetic computability theory in CIC. The basic principles of synthetic computability theory as introduced by Richman and Bauer [41, 1] were first applied to CIC by Forster et al. [10]. An investigation of Church's thesis [31, 52] to enhance the expressivity and applicability of synthetic computability theory in CIC was conducted by Forster [7, 9, 8]. Note that Forster uses an axiomatic notion of partial functions which can be instantiated with our representation (Definition 3). Moreoever, the obtained framework was used to mechanise various undecidability results for several decision problems [13], including the solvability of Diophantine equations [32] by Larchey-Wendling and Forster.

Hermes and Kirst [20] use synthetic methods to analyse Tennenbaum's theorem [50] in constructive type theory, stating that the standard model over $\mathbb{N}$ is the only computable model of PA. In their development, they assume CT_{Q} for total functions (Fact 26) and leave the derivation of CT_{Q} from a more common axiom for synthetic computability such as EPF_{μ} for future work. They also introduce a related but stronger semantic notion of Σ_1-formulas based on decidability properties (compared to our Definition 23) and derive corresponding versions of weak Σ_1-representability (Theorem 28) and Σ_1-completeness (Fact 33).

Mechanisations of Gödel's incompleteness theorems. The earliest mechanisation of Gödel's first incompleteness theorem was developed by Shankar in 1994 [43] using Nqthm [3], also called the Boyer-Moore theorem prover, a proof assistant based on Lisp. He does not mechanise incompleteness of arithmetic, but of a finite set theory, which simplifies encoding recursive structures, such as formulas and proofs, immensely. His development consists of

around 20 000 lines of code. A mechanisation of incompleteness of first-order arithmetic, based on an axiomatisation similar to Robinson arithmetic, was first developed by O'Connor in 2005 [35] using Coq, consisting of almost 44 000 lines of code. Another mechanisation of incompleteness of arithmetic using HOL Light [18] was developed by Harrison in 2009 [19].

More recently, both of Gödel's incompleteness theorems were mechanised by Paulson in 2014 [37] in around 12 000 lines of Isabelle [34] code. He showed incompleteness of a finite set theory slightly different from the one used by Shankar. To our knowledge, he was the first to give a complete mechanisation of Gödel's second incompleteness theorem, relying on a proof outline by Swierczkowski [48]. Also using Isabelle, Popescu and Traytel [38, 39] in 2019 mechanised both incompleteness theorems using the Gödel-Rosser approach abstractly, based on a much more subtle notion of formal systems than ours, additionally incorporating substitutions, soundness, arithmetic, and more.

None of the mechanisations mentioned above used Kleene's approach to incompleteness, let alone a synthetic approach to computability arguments. However, for example O'Connor used the representability of primitive recursive functions as an intermediate step to show weak representability of first-order provability, similarly as in Gödel's original proof.

The weak computational form of incompleteness for first-order arithmetic and set theory in Coq was mechanised by Kirst and Hermes [23], as a by-product of a general approach to the undecidability of first-order axiom systems. Their result differs from ours in two ways: First, they do not obtain essential incompleteness since they rely on Kleene's early proof using the halting problem (see Theorem 12). Instead, they give an abstract notion of formal systems incorporating soundness, and use it to deduce incompleteness of all sound extensions of their axiomatisation. Secondly, their development does not deduce falsity from the assumption of incompleteness, instead constructing a decider for the halting problem of Turing machines, which also prevents them from constructing an independent sentence.

7.3 Future Work

We have not considered the conditions under which Rosser's trick is applicable abstractly but just gave the concrete proof of strong separability derived from weak representability for Q in Section 6. Generalising this proof could simplify future instantiations of the stronger incompleteness results, as long as the abstraction is sufficiently simple.

Similarly on the abstract level, it is conceivable that instead of working with EPF to internalise that every function $\mathbb{N} \rightharpoonup \mathbb{N}$ is computable from the start, one could also axiomatise a predicate $(\mathbb{N} \rightharpoonup \mathbb{N}) \to \mathbb{P}$ describing the computable functions with enough closure properties to perform the intermediate constructions. Then one can still assume EPF to obtain the same results for the trivially true predicate, but also an assumption-free version (then better comparable to the related mechanisations) could be obtained if the predicate refers to a specific model of computation for which the necessary closure properties are verified.

Our instantiation to first-order logic with Robinson's Q currently relies on Larchey-Wendling and Forster's mechanisation of the DPRM theorem [32]. The DPRM theorem, however, is a much stronger statement than the representability property we actually need, and is considerably harder to show. Using our mechanisation of Σ_1-completeness, it appears feasible to obtain weak representability of μ-enumerable predicates (or predicates enumerable in any equivalent model of computation) for Q directly by just finding first-order formulas that define these predicates in the standard model. Similar approaches have been taken by O'Connor [35] and Paulson [37].

In Section 6, we showed that EPF_μ implies Church's thesis for Q. Along the lines of Fact 25, we expect the converse to be provable as well by first showing that, given any partial function by Q, its graph is μ-enumerable, which suffices for its μ-computability.

Mechanising this fact, however, would be challenging because we would have to implement our first-order logic, that is, substitution, enumerability of provable formulas, etc. using μ-recursive functions. Automatic extractions of such functions for first-order logic, specifically into a lambda calculus, have already been investigated by Forster, Kirst, and Wehr [11] using a tool by Forster and Kunze [12].

References

1 Andrej Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006.

2 Lev D. Beklemishev. Gödel incompleteness theorems and the limits of their applicability. i. *Russian Mathematical Surveys*, 65(5):857, 2010.

3 Robert S. Boyer, Matt Kaufmann, and J S. Moore. The Boyer-Moore theorem prover and its interactive enhancement. *Computers & Mathematics with Applications*, 29(2):27–62, 1995.

4 Alonzo Church. A note on the Entscheidungsproblem. *The journal of symbolic logic*, 1(1):40–41, 1936.

5 Thierry Coquand and Gérard Huet. *The calculus of constructions.* PhD thesis, INRIA, 1986.

6 Martin Davis, Hilary Putnam, and Julia Robinson. The decision problem for exponential Diophantine equations. *Annals of Mathematics*, pages 425–436, 1961.

7 Yannick Forster. Church's thesis and related axioms in Coq's type theory. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *LIPIcs*, pages 21:1–21:19, Dagstuhl, Germany, 2021.

8 Yannick Forster. *Computability in constructive type theory.* PhD thesis, Saarland University, 2021.

9 Yannick Forster. Parametric Church's thesis: Synthetic computability without choice. In *International Symposium on Logical Foundations of Computer Science*, pages 70–89. Springer, 2022.

10 Yannick Forster, Dominik Kirst, and Gert Smolka. On synthetic undecidability in Coq, with an application to the Entscheidungsproblem. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2019.

11 Yannick Forster, Dominik Kirst, and Dominik Wehr. Completeness theorems for first-order logic analysed in constructive type theory: Extended version. *Journal of Logic and Computation*, 31(1):112–151, 2021.

12 Yannick Forster and Fabian Kunze. A certifying extraction with time bounds from Coq to call-by-value lambda calculus. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ITP.2019.17`.

13 Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. A Coq library of undecidable problems. In *CoqPL 2020*, New Orleans, LA, United States, 2020. URL: `https://github.com/uds-psl/coq-library-undecidability`.

14 Torkel Franzén. *Gödel's theorem: an incomplete guide to its use and abuse.* AK Peters/CRC Press, 2005.

15 Kurt Gödel. *Über die Vollständigkeit des Logikkalküls.* PhD thesis, University of Vienna, 1929.

16 Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für mathematik und physik*, 38(1):173–198, 1931.

17 Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, 37:349–360, 1930. URL: `https://zbmath.org/?q=an%3A56.0046.04`.

18 John Harrison. HOL Light: a tutorial introduction. In *Formal Methods in Computer-Aided Design*, pages 265–269. Springer Berlin Heidelberg, 1996.

19 John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

20 Marc Hermes and Dominik Kirst. An analysis of Tennenbaum's theorem in constructive type theory. In *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, 2022.

21 Douglas R. Hofstadter. *Gödel, Escher, Bach*. Basic books New York, 1979.

22 Johannes Hostert, Mark Koch, and Dominik Kirst. A toolbox for mechanised first-order logic. In *The Coq Workshop*, 2021.

23 Dominik Kirst and Marc Hermes. Synthetic undecidability and incompleteness of first-order axiom systems in Coq (extended version). To appear.

24 Dominik Kirst, Johannes Hostert, Andrej Dudenhefner, Yannick Forster, Marc Hermes, Mark Koch, Dominique Larchey-Wendling, Niklas Mück, Benjamin Peters, Gert Smolka, and Dominik Wehr. A Coq library for mechanised first-order logic. In *The Coq Workshop*, 2022.

25 Dominik Kirst and Dominique Larchey-Wendling. Trakhtenbrot's Theorem in Coq: Finite Model Theory through the Constructive Lens. *Logical Methods in Computer Science*, Volume 18, Issue 2, June 2022. doi:10.46298/lmcs-18(2:17)2022.

26 Stephen C. Kleene. General recursive functions of natural numbers. *Mathematische annalen*, 112(1):727–742, 1936.

27 Stephen C. Kleene. Recursive predicates and quantifiers. *Transactions of the American Mathematical Society*, 53(1):41–73, 1943.

28 Stephen C. Kleene. A symmetric form of Gödel's theorem. *Journal of Symbolic Logic*, 16(2), 1951.

29 Stephen C. Kleene. Introduction to Metamathematics, 1952.

30 Stephen C. Kleene. *Mathematical Logic*. Dover books on mathematics. Dover Publications, 2002.

31 Georg Kreisel. Church's thesis: a kind of reducibility axiom for constructive mathematics, 1970.

32 Dominique Larchey-Wendling and Yannick Forster. Hilbert's Tenth Problem in Coq (Extended Version). *Logical Methods in Computer Science*, Volume 18, Issue 1, March 2022.

33 Juri V. Matijasevic. Enumerable sets are Diophantine. In *Soviet Math. Dokl.*, volume 11, pages 354–358, 1970.

34 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283. Springer Science & Business Media, 2002.

35 Russell O'Connor. Essential incompleteness of arithmetic verified by Coq. In *International Conference on Theorem Proving in Higher Order Logics*, pages 245–260. Springer, 2005.

36 Christine Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *International Conference on Typed Lambda Calculi and Applications*, pages 328–345. Springer, 1993.

37 Lawrence C. Paulson. A mechanised proof of Gödel's incompleteness theorems using Nominal Isabelle. *Journal of Automated Reasoning*, 55(1):1–37, 2015.

38 Andrei Popescu and Dmitriy Traytel. A formally verified abstract account of Gödel's incompleteness theorems. In *International Conference on Automated Deduction*, pages 442–461. Springer, 2019.

39 Andrei Popescu and Dmitriy Traytel. Distilling the requirements of Gödel's incompleteness theorems with a proof assistant. *Journal of Automated Reasoning*, 65(7):1027–1070, 2021.

40 Emil L. Post. Absolutely unsolvable problems and relatively undecidable propositions–account of an anticipation (1941). *Collected Works of Post*, pages 375–441, 1994.

41 Fred Richman. Church's thesis without tears. *The Journal of symbolic logic*, 48(3):797–803, 1983.

42 Barkley Rosser. Extensions of some theorems of Gödel and Church. *The journal of symbolic logic*, 1(3):87–91, 1936.

43 Natarajan Shankar. *Proof-checking metamathematics*. PhD thesis, The University of Texas at Austin, 1986.

44 Peter Smith. *An introduction to Gödel's theorems*. Cambridge University Press, 2013.

45 Peter Smith. Gödel without (too many) tears, 2021.

46 Raymond M. Smullyan. *Gödel's incompleteness theorems*. Oxford University Press on Demand, 1992.

47 Andrew W. Swan and Taichi Uemura. On Church's thesis in cubical assemblies. *Mathematical Structures in Computer Science*, pages 1–20, 2019.

48 Stanislaw Swierczkowski. Finite sets and Gödel's incompleteness theorems. *Dissertationes Mathematicae*, 422:1–58, 2003.

49 The Coq Development Team. The Coq proof assistant, January 2022. doi:10.5281/zenodo.5846982.

50 Stanley Tennenbaum. Non-Archimedean models for arithmetic. *Notices of the American Mathematical Society*, 6(270):44, 1959.

51 Amin Timany and Matthieu Sozeau. Consistency of the predicative calculus of cumulative inductive constructions (pCuIC). *CoRR*, abs/1710.03912, 2017. arXiv:1710.03912.

52 Anne S. Troelstra and Dirk Van Dalen. *Constructivism in Mathematics*. Vol. 121 of Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1988.

53 Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

54 Benno Van den Berg and Jaap Van Oosten. Arithmetic is categorical, 2011. Technical report.

55 Norihiro Yamada. Game semantics of Martin-Löf type theory, part III: its consistency with Church's thesis. *arXiv e-prints*, 2020. arXiv:2007.08094.

Finite Model Theory and Proof Complexity Revisited: Distinguishing Graphs in Choiceless Polynomial Time and the Extended Polynomial Calculus

Benedikt Pago ✉
Mathematical Foundations of Computer Science, RWTH Aachen University, Germany

Abstract

This paper extends prior work on the connections between logics from finite model theory and propositional/algebraic proof systems. We show that if all non-isomorphic graphs in a given graph class can be distinguished in the logic *Choiceless Polynomial Time* with counting (CPT), then they can also be distinguished in the *bounded-degree extended polynomial calculus* (EPC), and the refutations have roughly the same size as the resource consumption of the CPT-sentence. This allows to transfer lower bounds for EPC to CPT and thus constitutes a new potential approach towards better understanding the limits of CPT. A super-polynomial EPC lower bound for a PTIME-instance of the graph isomorphism problem would separate CPT from PTIME and thus solve a major open question in finite model theory.
Further, using our result, we provide a model theoretic proof for the separation of bounded-degree polynomial calculus and bounded-degree *extended* polynomial calculus.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases finite model theory, proof complexity, graph isomorphism

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.31

Related Version *Full Version*: `https://arxiv.org/abs/2206.05086` [24]

Acknowledgements I would like to thank Daniel Wiebking for extensively answering my numerous questions on Deep Weisfeiler Leman and the properties of coherent configurations.

1 Introduction and results

In recent years, a close connection between propositional proof complexity and finite model theory has been discovered and investigated – this is fruitful in particular because it allows the transfer of lower bounds between the two fields. In [3], Berkholz and Grohe showed that, with respect to the *graph isomorphism problem*, *fixed-point logic with counting* (FPC) has the same expressive power as the *bounded-degree monomial calculus*. In [14] it was shown more generally that there are mutual simulations between different variants of fixed-point logic and resolution/monomial calculus, not only for the graph isomorphism problem, but for deciding any classes of finite structures. These simulations preserve the relevant complexity parameters: The number of variables in a fixed-point sentence is reflected in the width/degree of the corresponding resolution/monomial calculus refutation, and vice versa. Therefore, known lower bounds for these respective parameters can be transferred between proof complexity and finite model theory. We extend this line of research from the rather well-understood fixed-point logics to a stronger model of computation, *Choiceless Polynomial Time* (CPT). This logic is an extension of FPC with a mechanism to construct (isomorphism-invariant) higher-order objects, i.e. nested sets, over the input structure. This power to create new objects puts CPT in a realm beyond formalisms with bounded variable number, bounded width, or bounded degree. As it turns out, a proof system that can naturally simulate

this mechanism is the bounded-degree *polynomial calculus with extension axioms* over $\mathbb{Q}$. *Extension axioms* can be added to any proof system; they allow to introduce new variables in a proof as abbreviations for more complex expressions, which generally allows for considerably shorter proofs.

We consider a similar setting as in [3], that is, we compare the logic and the proof system with respect to their power to *distinguish non-isomorphic graphs*. We say that CPT distinguishes all graphs in a graph class $\mathcal{K}$ if there exists a polynomial resource bound $p(n)$ such that for all pairs of non-isomorphic graphs $G, H \in \mathcal{K}$, there exists a CPT-sentence with time and space bound $p(n)$ that evaluates to true in one of the graphs and false in the other one (Definition 4). A proof system distinguishes G and H if it can refute the statement "G and H are isomorphic", encoded in a natural way as a propositional formula/system of polynomial equations $P_{\text{iso}}(G, H)$ (Definition 7). Our main result reads as follows:

▶ **Theorem 1.** *Let $\mathcal{K}$ be a class of graphs such that* CPT *distinguishes all graphs in $\mathcal{K}$. Then the degree-*3 *extended polynomial calculus over $\mathbb{Q}$ (denoted* EPC_3*) distinguishes all graphs in $\mathcal{K}$ with refutations of polynomial size.*
Moreover, the EPC_3*-refutation uses only extension axioms $\frac{}{X_f - f}$ for polynomials of the form $f = X \cdot Y$ or $f = \frac{1}{n} \cdot \left(\sum_{i=1}^{n^2} X_i \right)$. That is, only monomials and certain "averaged sums" are replaced with new variables.*

This has two main consequences. Most importantly, it establishes a new potential approach for the difficult open problem of proving strong lower bounds for CPT. A central topic in finite model theory is the quest for a logic that captures Ptime (see [6, 15, 17, 25]). At the moment, CPT is arguably the most prominent candidate logic for this, after rank logic has been ruled out [22]. That is, evaluating any fixed CPT-sentence in a given input structure is in Ptime, and as of yet, no decision problem in Ptime is known that cannot be defined by a CPT-sentence. However, the isomorphism-invariance of CPT is a severe limitation. Intuitively, it means that every classical algorithm involving choices or ordered iterations, such as e.g. Gaussian elimination, has to be executed in parallel for all possible orderings of the input structure, at least if it is implemented in the naive way in CPT. This requires exponential space and time resources. Thus, CPT can only be equal to P if there exists some clever trick that allows to simulate ordered iterations in a symmetry-invariant way. One quite well-studied problem that is conjectured to be hard for CPT is solving linear equation systems over finite fields – particularly hard instances of this problem arise as encodings of the isomorphism problem of *Cai-Fürer-Immerman graphs* [5] or *multipedes* [18]. Thus, if CPT does not capture Ptime, then it is quite likely that the graph isomorphism problem on a suitable graph class is a witness for that. Unfortunately, only few techniques for proving limitations of CPT are known (essentially the symmetry-based ones employed in [10, 26, 23]). Theorem 1 opens up a new perspective, as it enables us to transfer proof-theoretic lower bounds to CPT:

▶ **Theorem 2.** *If there is a class $\mathcal{K}$ of graphs on which the isomorphism problem is in* Ptime*, but which cannot be distinguished in* EPC_3 *with polynomial-size refutations using only extension axioms of the form mentioned in Theorem 1, then* CPT $\neq$ Ptime*.*

Interesting candidate graph classes are the said CFI-graphs or multipedes; crucially, without any order relation, because on certain ordered versions of these graphs, the isomorphism problem is already known to be in CPT ([10], [1]). Recently, a super-polynomial lower bound for EPC was found [2]. It concerns the *bit-value principle* and is based on the bit-complexity of the coefficients required for a refutation. It might be a starting point in the search for

graph isomorphism lower bounds, even though it seems that its proof is not directly adaptable to this problem. A second consequence of Theorem 1, together with known results from finite model theory and proof complexity ([10, 3]), is the separation of the bounded-degree polynomial calculus and EPC_3:

▶ **Theorem 3.** *There exists a sequence of pairs of non-isomorphic graphs $(G_n, H_n)_{n\in\mathbb{N}}$ such that $P_{iso}(G_n, H_n)$ has a polynomial-size refutation in the degree-3 extended polynomial calculus (using only extension axioms of the aforementioned form) but there is no $k \in \mathbb{N}$ such that the degree-k polynomial calculus can refute $P_{iso}(G_n, H_n)$ for all n.*

To our knowledge, the separation of these two bounded-degree proof systems has not explicitly been stated before – specifically for the graph isomorphism problem. In the unbounded-degree setting, an exponential separation between PC and EPC is known, even if one only allows extension axioms of the form $\overline{X} = 1 - X$, as in *polynomial calculus resolution* [11]. The main value of Theorem 3 is that it demonstrates how finite-model-theoretic lower and upper bounds can directly lead to corresponding results in proof complexity. Other examples of finite-model-theoretic proofs for results in proof-complexity were given in [14].

2 Preliminaries

All structures in this article are finite and relational. Formally, we assume that all relations are binary (whenever we need unary relations, we encode them as binary relations). We use the words "graphs" and "binary structures" interchangeably, so in particular, graphs can be vertex- or edge-coloured. For a τ-structure A and relation symbol $R \in \tau$, $R(A)$ denotes the corresponding relation in the structure A. The universe of A is denoted $V(A)$. We need the following concepts from finite model theory:

Weisfeiler Leman algorithm. The k-dimensional Weisfeiler Leman algorithm (k-WL) is an incomplete graph isomorphism test that computes a canonical colouring of the k-tuples of vertices. Two graphs G and H are distinguished by k-WL if there is a colour class whose size is different in the colouring of G and of H. For a precise definition and a survey, see e.g. [21].

***k*-variable counting logic.** We denote by $\mathcal{C}^k$ the k-variable fragment of first-order logic augmented with counting quantifiers $\exists^{\geq i}$, for every $i \in \mathbb{N}$. For two structures G, H we write $G \equiv_{\mathcal{C}^k} H$ if G and H satisfy exactly the same $\mathcal{C}^k$-sentences. It is well-known (see Theorem 2.2 in [21]) that k-WL distinguishes G and H if and only if $G \not\equiv_{\mathcal{C}^{k+1}} H$. In fact, the colour classes of the stable k-WL colouring correspond to the $\mathcal{C}^{k+1}$-types of the k-tuples. In this paper, we are mainly concerned with 2-WL and $\mathcal{C}^3$-types of vertex-pairs in graphs. The $\mathcal{C}^3$-type of a pair (v, w) in a structure A is the collection of all $\mathcal{C}^3$-formulas $\varphi(x, y)$ such that $A \models \varphi(v, w)$. It contains a lot of (non-local) information, e.g. whether or not v and w are connected, the length of the shortest path between them, etc.

The bijective *k*-pebble game. This game is played by two players, Spoiler and Duplicator, on two structures G and H. A position of a play is a set of pebble-pairs $\pi \subseteq V(G) \times V(H)$ with $|\pi| \leq k$. In every round, Spoiler selects a subset $\pi' \subseteq \pi$ with $|\pi'| < k$ of the current pebbles, which remain on the board. Duplicator then specifies a bijection $f : V(G) \longrightarrow V(H)$. Spoiler chooses a $v \in V(G)$, leading to the new position $\pi' \cup \{(v, f(v))\}$. Spoiler wins if the pebbles do not induce a local isomorphism between the pebbled substructures. Duplicator wins if she can play infinitely avoiding the pebbling of non-isomorphic substructures. Spoiler has a winning strategy for the bijective k-pebble game on G and H if and only if $G \not\equiv_{\mathcal{C}^k} H$ [19].

Fixed-point logic with counting (FPC). FPC is a standard logic of reference in algorithmic model theory. For the purposes of this paper, it suffices to know that for every sentence $\psi \in \mathrm{FPC}$, there is a k such that whenever it holds $G \equiv_{\mathcal{C}^k} H$ for two structures, then $G \models \psi$ if and only if $H \models \psi$. See [9] for a survey on this logic and its expressive power.

3 Choiceless Polynomial Time

By CPT we always mean Choiceless Polynomial Time *with counting*. For details and various ways to define CPT formally, we refer to the literature: A concise survey can be found in [13]. The work that originally introduced CPT as an abstract state machine model is [4]; later, more "logic-like" presentations of CPT were invented, such as Polynomial Interpretation Logic (see [12, 27]) and BGS-logic [26]. In short, CPT is FPC plus a mechanism to construct isomorphism-invariant hereditarily finite sets of polynomial size. When a CPT-sentence Π is evaluated in a finite structure A, then Π may augment A with hereditarily finite sets over its universe. The total number of distinct sets appearing in them (i.e. the sum over the sizes of the transitive closures of the h.f. sets) and the number of computation steps is bounded by $p(|A|)$, where $p(n)$ is a polynomial that is explicitly part of the sentence Π. We also write $\mathrm{CPT}(p(n))$ for the set of all CPT-sentences whose polynomial bound is at most $p(n)$. For the sake of illustration, we sketch the definition of BGS-logic:

The sentences of BGS-logic are called *programs*. A program is a tuple $\Pi = (\Pi_{\mathrm{step}}(x), \Pi_{\mathrm{halt}}(x), \Pi_{\mathrm{out}}(x), p(n))$. Here, $\Pi_{\mathrm{step}}(x)$ is a BGS-term, Π_{halt} and Π_{out} are BGS-formulas, and $p(n)$ is a polynomial that bounds the time and space used by the program. BGS-terms take as input hereditarily finite sets and output a hereditarily finite set. Examples of such terms are $\mathsf{Pair}(x, y)$, which evaluates to $\{x, y\}$, or $\mathsf{Union}(x) = \bigcup_{y \in x} y$. Furthermore, if s and t are terms, x is a variable, and φ a formula, then $\{s(x) \ : \ x \in t \ : \ \varphi(t)\}$ is a comprehension term. It applies the term s to all elements of the set defined by t that satisfy φ, and outputs the set of the resulting objects $s(x)$. When a program is evaluated in a given finite structure A, then the term $\Pi_{\mathrm{step}}(x)$ is iteratively applied to its own output, starting with $x_0 = \emptyset$. The iteration stops in step i if the computed set $x_i = (\Pi_{\mathrm{step}})^i(\emptyset)$ satisfies $A \models \Pi_{\mathrm{halt}}(x_i)$. The formula Π_{out} defines, in dependence of x_i, whether the run is accepting or rejecting, that is, whether $A \models \Pi$ or not. If the length of the run or the size of the transitive closure of x_i exceeds $p(|A|)$ at some point, then the computation is aborted, and $A \not\models \Pi$.

Recently, the computation model *Deep Weisfeiler Leman* (DWL) has been introduced by Grohe, Schweitzer and Wiebking [16]. DWL and CPT mutually simulate each other, and DWL is better suited to establish the connection to proof complexity. We present DWL in detail in Section 7, since our proof of Theorem 1 actually goes via DWL. When we say that CPT *distinguishes* certain graphs, we formally mean this:

▶ **Definition 4** (Distinguishing relational structures in CPT). *Let $\mathcal{K}$ be a class of τ-structures. We say that* CPT *distinguishes all structures in $\mathcal{K}$ if there exists a polynomial $p(n)$ and a constant $k \in \mathbb{N}$ such that for any two structures $G, H \in \mathcal{K}$ which are non-isomorphic, there exists a sentence $\Pi \in \mathrm{CPT}(p(n))$ with $\leq k$ variables such that $G \models \Pi$ and $H \not\models \Pi$.*

This definition is perhaps non-standard because we allow the distinguishing sentence to be different for every pair of graphs in $\mathcal{K}$, whereas normally, one would expect a single sentence that distinguishes all graphs in the class. Our definition, however, matches the situation in proof complexity. As we explain in the next section, a proof system distinguishes all graphs in $\mathcal{K}$, if there exists an efficient proof for non-isomorphism of each pair of non-isomorphic graphs. This proof can of course be a different one for each pair of graphs, and the distinguishing

CPT-sentences will play the role of the non-isomorphism proofs. The constant bound on the variable number is needed because with an unbounded number of variables, we could already find a first-order sentence for every graph that describes it up to isomorphism. Later on, in the DWL framework, this variable bound becomes irrelevant because DWL algorithms naturally correspond to bounded-variable CPT programs.

The distinguishing power of CPT is strictly greater than that of FPC. This can for example be seen by considering CFI-graphs over linearly ordered *base graphs* (every CFI-graph is obtained by applying the construction from [5] to a given connected base graph). We can summarise the situation like this:

▶ **Theorem 5** ([10]). *There is a family of pairs of graphs $(G_n, H_n)_{n\in\mathbb{N}}$, that are equipped with a total preorder on the vertex set (encoded as a binary relation $\preceq$), such that:*

- *$G_n \ncong H_n$ for all $n \in \mathbb{N}$.*
- *For every* FPC*-sentence ψ and all large enough $n \in \mathbb{N}$: $G_n \models \psi$ if and only if $H_n \models \psi$.*
- *There is a* CPT*-sentence Π such that for all $n \in \mathbb{N}$: $G_n \models \Pi$ and $H_n \not\models \Pi$.*

4 The (extended) polynomial calculus

The *polynomial calculus* (PC) was introduced in [8]. It is applicable to the following problem: Given a set P of multivariate polynomials over a fixed field (in our case, $\mathbb{Q}$), decide if the polynomials in P have a common zero with respect to $\{0,1\}$-assignments. The polynomial 1 is derivable from P if and only if the polynomials in P have no common zero over $\{0,1\}$. A derivation of the 1-polynomial is formally a sequence $p_1, p_2, ..., p_n = 1$ of polynomials such that each p_i is either in P or an axiom of the polynomial calculus or is obtained from one or multiple p_j, for $j < i$, with the application of one of the derivation rules listed below. A derivation of the 1-polynomial from P is called a *refutation* of P.
A restricted variant of the polynomial calculus, the monomial calculus, has been introduced in [3]. The derivation rules of the polynomial/monomial calculus are the following:

▶ **Definition 6** (Inference rules of the (extended) polynomial calculus). *Let P be the set of input polynomials/axioms, $a, b \in \mathbb{Q}$, X a variable, and f, g polynomials with rational coefficients.*

$$\frac{p \in P}{p}\ \textit{(Axioms)} \qquad \frac{}{X^2 - X}\ \textit{(Boolean axioms)}$$

$$\frac{f}{Xf}\ \textit{(Multiplication rule)} \qquad \frac{g \quad f}{ag + bf}\ \textit{(Linear combination rule)}$$

In the extended polynomial calculus *(*EPC*),* extension axioms *of the form $\frac{}{X_f - f}$ may be used whenever X_f is a fresh variable not occurring in f. The Boolean axioms do not apply to these extension variables.*

The *monomial calculus* (MC) is a restriction of PC that permits the use of the multiplication rule only in the cases where f is either a monomial or the product of a monomial and an axiom. For MC, PC, and EPC, we also consider the degree-k restrictions denoted MC_k, PC_k, and EPC_k. Proofs in these degree-restricted calculi may only consist of polynomials of degree at most k. In general, these proof systems are not complete any more, but bounding the degree by a constant yields natural fragments that admit efficient proof search via Gröbner basis computation (at least for MC_k and PC_k, this is the case; see [8]).
The *size* of a refutation $p_1, p_2, ..., p_n = 1$ is the total number of occurrences of monomials in all its polynomials. Its *bit-complexity* is the maximum number of bits required to represent any of the occurring coefficients, where values in $\mathbb{Q}$ are stored as a fraction of two binary numbers.

Intuitively, the effect of the extension axioms in the bounded-degree setting is that the degree-bound of three may be "locally" violated: Monomials like $A \cdot B \cdot C \cdot D$ can be written as $X_{AB} \cdot X_{CD}$, where X_{AB} and X_{CD} are fresh extension variables such that $X_{AB} = A \cdot B$, and $X_{CD} = C \cdot D$. Thus, with the help of extension axioms, we can implicitly use monomials of larger degree than allowed. If we restrict ourselves to refutations of polynomial size, then we can think of EPC_3 as a version of degree-3 polynomial calculus where the degree bound can be violated a limited number of times.

5 Applying algebraic proof systems to the graph isomorphism problem

Let G and H be fixed graphs, potentially with a colouring of the vertices or with multiple edge relations. We consider the following polynomial axiom system $P_{\mathrm{iso}}(G, H)$ that expresses the existence of a (colour-preserving) isomorphism between G and H. A refutation of $P_{\mathrm{iso}}(G, H)$ in any variant of the polynomial calculus then witnesses that G and H are non-isomorphic. This definition of $P_{\mathrm{iso}}(G, H)$ is almost the same as in [3].

▶ **Definition 7** ($P_{iso}(G, H)$, [3]). *Let G and H be two graphs (potentially vertex-coloured). Let $\sim \subseteq V(G) \times V(H)$ be the relation "vertex $v \in V(G)$ and $w \in V(H)$ have the same colour". The system $P_{iso}(G, H)$ consists of the following polynomials in the variables $\{X_{vw} \mid v \in V(G), w \in V(H), v \sim w\}$.*

$$\sum_{\substack{v \in V(G) \\ v \sim w}} X_{vw} - 1 \qquad \text{for all } w \in V(H) \tag{1}$$

$$\sum_{\substack{w \in V(H) \\ v \sim w}} X_{vw} - 1 \qquad \text{for all } v \in V(G) \tag{2}$$

$$X_{vw} X_{v'w'} \qquad \begin{array}{l} \text{for all } v, v' \in V(G), w, w' \in V(H) \\ \text{with } v \sim w \text{ and } v' \sim w' \\ \text{such that } \{(v, w), (v', w')\} \text{ is not} \\ \text{a local isomorphism.} \end{array} \tag{3}$$

The intended meaning of the variable X_{vw} being set to one is "v is mapped to w". When we say that a certain variant of the polynomial calculus *distinguishes* two graphs G, H, we mean that the polynomial equation system $P_{\mathrm{iso}}(G, H)$ has a refutation in that proof system. The main result from [3] links graph distinguishability in this sense to graph distinguishability by the k-dimensional Weisfeiler Leman algorithm.

▶ **Theorem 8** (Theorem 4.4 in [3]). *Let $k \in \mathbb{N}$ and let G and H be graphs. The axiom system $P_{iso}(G, H)$ has a refutation in the degree-k monomial calculus iff the $(k-1)$-dimensional Weisfeiler Leman algorithm distinguishes G and H.*

In [3], this is not stated for vertex- or edge-coloured graphs, but it can be checked that the proof still goes through in these cases. For our result, we need some of the technical ingredients from the proof of Theorem 8: What is shown in [3] is that Spoiler's winning positions in the bijective k-pebble game on G and H are derivable in MC_k from $P_{\mathrm{iso}}(G, H)$. A position in the game is a set of pebble pairs $\pi \subseteq V(G) \times V(H)$ of size $|\pi| \leq k$. The position π corresponds to a monomial in the variables from $P_{\mathrm{iso}}(G, H)$. We denote this monomial as $X_\pi := \prod_{(v,w) \in \pi} X_{vw}$ (so the X_{vw} are the variables, whereas X_π is shorthand for a *product* of variables). We will use the following central technical result as a blackbox:

▶ **Lemma 9** (Lemma 4.2 in [3]). *Let $k \geq 2$ and G, H be graphs (such that for every vertex-colour Q, there are exactly as many vertices of colour Q in G as in H). If Spoiler has a winning strategy for the bijective k-pebble game on G, H with initial position π, then there is an MC_k-derivation of the monomial X_π from $P_{iso}(G, H)$.*

This lemma accounts for one direction of Theorem 8 because if $G \not\equiv_{\mathcal{C}^k} H$, then Spoiler wins the bijective k-pebble game from the initial position $\emptyset$, and we have $X_\emptyset = 1$.

6 Separating the bounded-degree extended polynomial calculus from its non-extended version

Before we come to the more technical part, we show how Theorem 3 follows from Theorem 1. According to Theorem 6.2 in [3], for every $n \in \mathbb{N}$, there exist pairs $G_n, \widetilde{G}_n$ of non-isomorphic CFI-graphs of size $\mathcal{O}(n)$ such that $P_{\text{iso}}(G_n, \widetilde{G}_n)$ has no degree-n polynomial calculus refutation (over $\mathbb{Q}$). A closer examination of the construction in [3] reveals that the axioms (1) and (2) in $P_{\text{iso}}(G_n, \widetilde{G}_n)$ are for coloured versions of the respective CFI-graphs: Each vertex-gadget and each edge-gadget of G_n and $\widetilde{G}_n$, respectively, forms a distinct colour class and the axioms restrict possible isomorphisms to colour-preserving ones (the precise definition of the said vertex- and edge-gadgets is not essential here, so we refer to [10] for the presentation of the CFI construction). Therefore, we have $P_{\text{iso}}(G_n, \widetilde{G}_n) = P_{\text{iso}}((G_n, \preceq), (\widetilde{G}_n, \preceq))$ for any preorder $\preceq$ on $V(G_n)$ that is obtained from a linear order on the respective base graph; in other words, a preorder that linearly orders the CFI-gadgets without ordering the vertices inside each gadget (for details, see [5] or [10]). Note that our definition of P_{iso} also works for graphs with multiple edge relations, and we can simply view the binary relation $\preceq$ as another type of edge relation. Now the system $P_{\text{iso}}((G_n, \preceq), (\widetilde{G}_n, \preceq))$ does have a polynomial-size refutation in EPC_3, for any choice of the ordering, because CFI-graphs over linearly ordered base graphs can be distinguished in CPT [10] and thus, a refutation exists by our Theorem 1.

7 Deep Weisfeiler Leman

Before we are ready to prove Theorem 1, we have to introduce the technical details of *Deep Weisfeiler Leman*, a computation model equivalent to CPT that we will simulate in EPC_3. A DWL-algorithm is a deterministic Turing machine that gets as input a finite structure with binary relations. All DWL-computations are isomorphism-invariant: The machine does not have access to the input structure directly, but only to its so-called *algebraic sketch*. This is a certain invariant of the structure, similar to its 2-dimensional Weisfeiler Leman colouring. The machine is not only able to read information about its input structure but it can also modify the structure in an isomorphism-invariant way. These modifications correspond to the creation of higher-order objects in Choiceless Polynomial Time.

Before we can introduce the Deep Weisfeiler Leman framework in detail, we have to say precisely what the algebraic sketch of a structure is. It is a representation of its *coarsest coherent configuration* (or "coherent colouring"), that is defined below. The coarsest coherent configuration of a structure is also known as its stable 2-dimensional Weisfeiler Leman colouring (equivalent to the partition of all pairs into their $\mathcal{C}^3$-types). The following presentation closely follows the one in [16].

7.1 Coherent configurations of binary structures

▶ **Definition 10** (Notions concerning binary relations, [16]).

- *The* converse *of a relation R is the relation $R^{-1} := \{(v,u) \mid (u,v) \in R\}$.*
- *For a set V, the* diagonal *of V is the relation* $\mathrm{diag}(V) := \{(v,v) \mid v \in V\}$. *For a relation $R \subseteq V^2$ we let $R^{\mathrm{diag}} := R \cap \mathrm{diag}(V)$ be the diagonal elements in R. We call R a* diagonal relation *if $R = R^{\mathrm{diag}}$.*
- *The* strongly connected components *(*SCC*s) of a relation R are defined in the usual way as inclusionwise maximal sets S such that for all $u, v \in S$ there is an R-path of length at least 1 from u to v. (A singleton set $\{u\}$ can be a strongly connected component only if $(u,u) \in R$.) We write $\mathrm{scc}(R)$ to denote the set of strongly connected components of R. Moreover, we let $R^{\mathrm{scc}} := \bigcup_{S \in \mathrm{scc}(R)} S^2$ be the relation describing whether two elements are in the same strongly connected component.*

▶ **Definition 11** (Coherent configurations, [16]). *Let σ be a vocabulary. A* coherent σ-configuration *C is a σ-structure C with the following properties.*

- *$\{R(C) \mid R \in \sigma\}$ is a partition of $V(C)^2$.*
- *For each $R \in \sigma$ the relation $R(C)$ is either a subset of or disjoint from the diagonal $\mathrm{diag}(V(C))$.*
- *For each $R \in \sigma$ there is an $R^{-1} \in \sigma$ such that $R^{-1}(C) = (R(C))^{-1}$.*
- *For all $R_1, R_2, R_3 \in \sigma$ there is a number $q = q(R_1,R_2,R_3) \in \mathbb{N}$ such that for all $(u,v) \in R_1(C)$ there are exactly q elements $w \in V(C)$ such that $(u,w) \in R_2(C)$ and $(w,v) \in R_3(C)$.*

The numbers $q(R_1,R_2,R_3)$ are called the intersection numbers *of C and the function $q : \sigma^3 \to \mathbb{N}$ is called the* intersection function.

A coherent σ-configuration C is at least as *fine* as, or *refines*, a τ-structure A (we write $C \sqsubseteq A$) if $V(A) = V(C)$, and for each $R \in \sigma$ and each $E \in \tau$ it holds that $R(C) \subseteq E(A)$ or that $R(C) \subseteq A^2 \setminus E(A)$. We say that a coherent configuration C is a *coarsest coherent configuration refining* a structure A if $C \sqsubseteq A$ and $C' \sqsubseteq C$ for every coherent configuration C' satisfying $C' \sqsubseteq A$. In the following, we will usually write τ for the vocabulary of a given structure and σ for the vocabulary of the corresponding coherent configuration, without further specifying σ.

Every binary structure A has a coarsest coherent configuration refining it, which can be computed efficiently with the 2-dimensional Weisfeiler Leman algorithm (Theorem 2.1 in [16]). This configuration is unique up to the renaming of relation symbols. We write $C(A)$ for the coarsest coherent configuration of A with canonical names of the relation symbols, as for example produced by a fixed implementation of 2-WL. We call the relation symbols in σ *colours* to distinguish them from the relation symbols in τ. In the following, we often identify the symbols in τ and σ with binary strings, because this is how they are represented in a Turing machine.

The *algebraic sketch* of a structure contains information about the colours appearing in its coarsest coherent configuration, which relations of the structure they refine, and the intersection function q. Formally, the algebraic sketch of a structure A is the tuple $D(A) = (\tau, \sigma, \subseteq_{\sigma,\tau}, q)$. The relation $\subseteq_{\sigma,\tau}$ relates the colours in σ with the relations in τ they refine: $\subseteq_{\sigma,\tau} := \{(R,E) \in \sigma \times \tau \mid R(C(A)) \subseteq E(A)\}$. In order to feed $D(A)$ to a Turing machine, we have to agree on some encoding in binary. If the binary string encodings of the relation symbols are fixed, then there is a canonical encoding of $D(A)$, based on the lexicographic ordering of the relation names and ordering of the intersection numbers $q(R_1,R_2,R_3)$. The string that encodes $D(A)$ is the initial tape content in a DWL-computation on the structure A.

7.2 The Deep Weisfeiler Leman computation model

A DWL-algorithm is a two-tape Turing machine M with an additional storage device that the authors of [16] have named "the *cloud*". It contains a structure A together with its coarsest coherent configuration $C(A)$. The storage that the machine itself can use is a *work tape* and an *interaction tape*, which allows for interaction with the cloud. The input of a DWL-program M is a binary τ-structure A. Initially, the cloud contains the coherently coloured structure $(A, C(A))$, and on the interaction tape, the algebraic sketch $D(A)$ is written. The work tape is empty.

The Turing machine works as a standard Turing machine with two special transitions that can modify the structure in the cloud. To execute these, the machine writes a binary string $s \in \{0,1\}^*$ on the interaction tape and enters one of the two distinguished states $q_{\texttt{addPair}}, q_{\texttt{contract}}$. If s is a colour $R \in \sigma$, then we say that M executes $\texttt{addPair}(R)$ or $\texttt{contract}(R)$. Executing $\texttt{addPair}(R)$ creates a new vertex for each vertex-pair whose colour in $C(A)$ is R. The operation $\texttt{contract}(R)$ contracts every SCC formed by pairs of colour R into a single vertex.

- $\texttt{addPair}(R)$: The machine adds a fresh vertex for each pair in $R(C(A))$ to A, i.e. it updates $V(A)$ to $V(A) \uplus R(C(A))$. These pairs are then connected with their elements in A. Therefore, τ is updated to $\tau \cup \{E_{\text{left}}, E_{\text{right}}\} \uplus \{D_R\}$. The new relation D_R identifies the newly added vertices, i.e. $D_R(A) := \text{diag}(R(C(A)))$. The symbol D_R is chosen as the lexicographically smallest binary string that is not yet used as a relation symbol.
  Furthermore, $E_{\text{left}}(A)$ is updated to $E_{\text{left}}(A) \cup \{(u,(u,v)) \in V(A)^2 \mid (u,v) \in R(C(A))\}$, and $E_{\text{right}}(A)$ is set to $E_{\text{right}}(A) \cup \{(v,(u,v)) \in V(A)^2 \mid (u,v) \in R(C(A))\}$ (where initially, $E_{\text{left}}(A) = E_{\text{right}}(A) = \emptyset$).
- $\texttt{contract}(R)$: Let $\mathcal{S} := \text{scc}(R)$ be the set of strongly connected components of the relation R. Let $U := V(A) \setminus \bigcup \mathcal{S}$ be the set of vertices that are not in one of the strongly connected components. The components in $\mathcal{S}$ are contracted. That means we update $V(A)$ to $U \uplus \mathcal{S}$, and τ to $\tau \uplus \{D_R\}$. Again, $D_R(A) := \text{diag}(\mathcal{S})$. For each relation $E \in \tau$, we update $E(A)$ to $(E(A) \cap U^2) \cup \{(u,S) \mid \exists v \in S \in \mathcal{S} : (u,v) \in E(A)\} \cup \{(S,v) \mid \exists u \in S \in \mathcal{S} : (u,v) \in E(A)\} \cup \{(S_1,S_2) \mid \exists u \in S_1 \in \mathcal{S}, \exists v \in S_2 \in \mathcal{S} : (u,v) \in E(A)\}$.

Each of these special transitions modifies the structure A in the cloud in an isomorphism-invariant way. After that, the cloud storage device computes $C(A)$ (for example, with the 2-WL algorithm) and stores the coherently coloured structure $(A, C(A))$. The algebraic sketch $D(A)$ of the new structure is written on the interaction tape.

A DWL-algorithm decides a class $\mathcal{K}$ of τ-structures in the usual sense, i.e. the algorithm halts with output 1 on input A if $A \in \mathcal{K}$, and else, it halts with output 0. We say that a DWL-algorithm *runs in polynomial time* if the number of computation steps of the Turing machine and the size of the structure in the cloud is bounded by a polynomial in the size of the input structure. The original definition of DWL in [16] also has two more operations, `create` and `forget`, but it can be shown that these do not increase the expressive power; the proof is similar to the proof in [16] showing that "pure DWL" simulates DWL, so we omit it.

7.3 Distinguishing graphs in Deep Weisfeiler Leman

In [16], the authors say that a DWL-algorithm *decides isomorphism* on a structure class $\mathcal{K}$ if it gets as input the disjoint union $A := G \uplus H$ of two connected binary structures and correctly decides whether $G \cong H$. A crucial technical result in [16] shows that one can always assume that at any stage of the computation, the structure in the cloud is the disjoint union of two connected structures: It is never necessary for the algorithm to produce connections

between the two components, i.e. `addPair` and `contract` are only executed for colours R with $R(C(A)) \subseteq V(G)^2 \cup V(H)^2$. A DWL-algorithm that maintains this invariant is called *normalised* in [16].

▶ **Definition 12** (Distinguishing structures in DWL). *The computation model* DWL distinguishes *all structures in a class $\mathcal{K}$ (of connected τ-structures) in* polynomial time *if: There is a polynomial $p(n)$ such that for any two non-isomorphic structures $G, H \in \mathcal{K}$, there exists a* normalised *DWL-algorithm M which, given $G \uplus H$ as input, terminates with a structure $G' \uplus H'$ in the cloud such that $D(G') \neq D(H')$, and takes time and space at most $p(|G|+|H|)$.*

This is simply the DWL-version of Definition 4 for distinguishing structures in CPT. The main difference to the CPT-setting is that here, the constant bound on the number of variables is already implicit in the definition of DWL (because DWL only accesses a structure via its coherent configuration, and two structures with the same configuration are $\mathcal{C}^3$-equivalent). Let us elaborate on what is meant precisely by $D(G') \neq D(H')$. Whenever $A = A_1 \uplus A_2$ is a τ-structure consisting of two separate connected components, then we write $D(A)[A_1]$ and $D(A)[A_2]$ for the restrictions of the algebraic sketch $D(A)$ to the σ-colours that occur as colours of pairs in $V(A_1)^2$ or $V(A_2)^2$, respectively. It follows from the properties of normalised DWL-computations (Lemma 8 in [16]) that $D(A)[A_i]$ is in fact the algebraic sketch of A_i, so the sketch of $A_1 \uplus A_2$ is composed of the sketches of the two structures:

▶ **Lemma 13.** *Let $A = A_1 \uplus A_2$ be the disjoint union of two connected τ-structures. For $i \in \{1, 2\}$, $D(A)[A_i]$ is an algebraic sketch and equal to $D(A_i)$, up to a renaming of the colours in σ.*

Thus, when we write $D(G') \neq D(H')$, we are formally referring to the respective restrictions of $D(G' \uplus H')$, but these are equivalent to $D(G')$ and $D(H')$, respectively. The algebraic sketches being distinct means that 2-WL distinguishes the structures. By standard results (see e.g. [21]), one can infer:

▶ **Lemma 14.** *Let G, H be connected τ-structures. It holds $D(G) \neq D(H)$ if and only if Spoiler has a winning strategy for the bijective 3-pebble game on G and H.*

We can say even more, namely that Spoiler can distinguish pairs of different colours. The following is a variation of a standard result (Theorem 2.2 in [21]). The standard result concerns the setting where the graphs G and H are considered separately with their respective coarsest coherent configurations. Here, we have to work with their disjoint union. It is perhaps not surprising that in this setting, the correspondence between Weisfeiler-Leman colourings and pebble games also exists, but we are not aware of a formal proof for this statement for $G \uplus H$ in the literature. Thus, for completeness, we provide one in the full version of the paper [24].

▶ **Lemma 15.** *Let G, H be two connected τ-structures and $A := G \uplus H$ with its coarsest coherent σ-configuration $C(A)$. Let $(v, v') \in V(G)^2, (w, w') \in V(H)^2$ such that there is no $R \in \sigma$ with $(v, v') \in R(C(A))$ and $(w, w') \in R(C(A))$.*
Then Spoiler has a winning strategy for the bijective 3-pebble game on G and H with initial position $\{(v, w), (v', w')\}$.

As shown below, if CPT distinguishes all structures in a class $\mathcal{K}$, then also DWL distinguishes all structures in $\mathcal{K}$ in polynomial time. Hence, in our proof of Theorem 1, we can indeed start with the assumption that DWL polynomially distinguishes all graphs in $\mathcal{K}$.

▶ **Lemma 16.** *Let $\mathcal{K}$ be a class of connected structures that are distinguished by* CPT *in the sense of Definition 4. Then DWL distinguishes all structures in $\mathcal{K}$ in polynomial time in the sense of Definition 12.*

Proof sketch. Let $p(n)$ be the resource bound for the distinguishing CPT-programs for the class $\mathcal{K}$ that exists by Definition 4. Fix two τ-structures $G, H \in \mathcal{K}$ such that $G \ncong H$. Let $\Pi \in \mathrm{CPT}(p(n))$ be a distinguishing sentence. By Theorem 21 in [16], there exists a polynomial time DWL-algorithm M which simulates Π (and the polynomial resource bound of M depends only on $p(n)$, not on G and H). That means M w.l.o.g. accepts G and rejects H. The sequence of executed cloud-interaction operations from $\{\texttt{addPair}, \texttt{contract}\}$ is the same in the run of M on G as in the run on H, up to the point where the respective structures in the cloud have distinct algebraic sketches (because the behaviour of M only depends on the algebraic sketch of the structure in the cloud). At that point, we can stop the simulation of Π by M because we do not actually care about the acceptance behaviour as long as the machine produces distinct sketches on G and H. Now the same sequence of cloud-interaction-operations can be simulated by an appropriate DWL-algorithm M' on input $G \uplus H$, which leads to a structure $G' \uplus H'$ with $D(G') \neq D(H')$. Such an M' can be constructed because of Lemma 13. ◀

8 Properties of coherent configurations

Here is a small collection of lemmas concerning coherent configurations. We will need them in our construction of the EPC_3-refutation in the next section.

▶ **Lemma 17.** *Let A be a τ-structure and $C(A)$ its coarsest coherent σ-colouring. Let $R \in \sigma$. There are diagonal colours $D_1, D_2 \in \sigma$ such that for all pairs $(u, v) \in R(C(A))$ we have $(u, u) \in D_1(C(A))$, and $(v, v) \in D_2(C(A))$.*

Proof. This is Corollary 2.1.7 in [7] (the term "fibers" there means the same as our "diagonal colours"). ◀

▶ **Corollary 18.** *Let A be a τ-structure and $C(A)$ its coarsest coherent σ-colouring. Let $R \in \sigma$. If for any $(u, v) \in R(C(A))$, (u, u) or (v, v) is in some relation $E(A)$, for $E \in \tau$, then for all other $(u', v') \in R(C(A))$, it also holds $(u', u') \in E(A)$, or $(v', v') \in E(A)$, respectively.*

Proof. Follows from the previous lemma and the fact that the coarsest coherent configuration of A is a refinement of the relations of A. ◀

▶ **Lemma 19.** *Let A be a τ-structure and let $C(A)$ be its coarsest coherent σ-configuration. Let $R \in \sigma$ and let $\mathcal{S} = \mathrm{SCC}(R)$ be the set of R-SCCs in $C(A)$. There is a diagonal relation $P \in \sigma$ such that* $\mathrm{diag}(\bigcup \mathcal{S}) = P(C(A))$.

Proof. First, we show that $\mathrm{diag}(\bigcup \mathcal{S}) \subseteq P(C(A))$. Any vertex $v \in \bigcup \mathcal{S}$ has some outgoing R-neighbour w that is in the same SCC (possibly, $w = v$). That is, we have $(v, w) \in R(C(A))$. By Lemma 17, there are specific diagonal relations D_1, D_2 such that $(v, v) \in D_1(C(A)), (w, w) \in D_2(C(A))$, and all endpoints of R-edges have these diagonal colours. But since w is itself the left entry in some other R-edge (w, w'), we must have $D_1 = D_2 =: P$.
It remains to prove $P(C(A)) \subseteq \mathrm{diag}(\bigcup \mathcal{S})$. For any $v \in \mathrm{diag}(\bigcup \mathcal{S})$, there exists an R-path of length ≥ 1 from v to itself. We have already argued that $(v, v) \in P(C(A))$. It follows that any other vertex w with $(w, w) \in P(C(A))$ also has an R-path to itself and is thus in $\bigcup \mathcal{S}$. To see this, recall that $C(A)$ corresponds to the stable 2-WL-colouring [16], which in turn

partitions A^2 into $\mathcal{C}^3$-types [21]. Hence, all vertices with the same diagonal colour satisfy exactly the same $\mathcal{C}^3$-formulas. The existence of an R-path from a vertex to itself (in the fixed structure A) is expressible in $\mathcal{C}^3$ using standard techniques: Namely, for every fixed number $d \leq |A|$, we can write a $\mathcal{C}^3$-formula $\varphi_d(x, y)$ that asserts the existence of a path of length d from x to y. Only 3 variables are needed because one can alternately requantify used variables (see e.g. Proposition 3.2 in [20]). In the formula, we have access to the relation R because it is itself $\mathcal{C}^3$-definable: Essentially, R is a $\mathcal{C}^3$-type of vertex-pairs in A, and it is known that on finite structures, such a type is definable with a single formula. ◀

▶ **Corollary 20.** *Let A be a τ-structure such that for every $v \in V(A)$, (v, v) is in exactly one diagonal relation $P(A)$, for $P \in \tau$ (i.e. A is a graph with vertex colours). Let $C(A)$ be the coarsest coherent σ-configuration of A. Let $R \in \sigma$ and let $\mathcal{S} = \mathrm{SCC}(R)$. There is a colour (i.e. a diagonal relation) $P \in \tau$ such that for every SCC $S \in \mathcal{S}$,* $\mathrm{diag}(S) \subseteq P(A)$.

Proof. The coarsest coherent configuration $C(A)$ is a refinement of A. Therefore, the diagonal relations in $C(A)$ are subsets of the diagonal relations in A. Now the statement follows directly from Lemma 19. ◀

▶ **Lemma 21.** *Let $A, C(A), R \in \sigma$, and $\mathcal{S}$ be as above. All R-SCCs in $\mathcal{S}$ are of equal size.*

Proof. For any number $k \in \mathbb{N}$, we can write a $\mathcal{C}^3$-formula $\varphi_k(x)$ asserting that the size of the R-SCC of x is exactly k. To do this, we can just use a counting quantifier and the fact that the existence of an R-path between two vertices (and back) is $\mathcal{C}^3$-definable (see proof of Lemma 19). Now since all vertices in R-SCCs have the same diagonal colour (Lemma 19), and colours coincide with $\mathcal{C}^3$-types, they all satisfy the same φ_k and thus, all SCCs have equal size. ◀

9 Refuting graph isomorphism in the extended polynomial calculus - Proof of Theorem 1

Let $\mathcal{K}$ be a class of connected binary structures such that CPT distinguishes all structures in $\mathcal{K}$. By Lemma 16, then also DWL distinguishes all structures in $\mathcal{K}$ in polynomial time. Now Theorem 1 follows from Lemma 23 below that establishes the link between DWL-distinguishability and the extended polynomial calculus. Before we can prove Lemma 23, we have to state the key technical result that it depends on:

▶ **Lemma 22.** *Let G, H be two connected binary τ-structures, which are potentially vertex-coloured in such a way that for every vertex-colour Q, there are as many vertices with colour Q in G as in H. Let* $\mathrm{op} \in \{\texttt{addPair}, \texttt{contract}\}$ *and let $R \in \sigma$, where σ is the vocabulary of the coarsest coherent configuration $C := C(G \uplus H)$. Assume that $R(C) \subseteq V(G)^2 \cup V(H)^2$. Let $G' \uplus H'$ be the result of executing* $\mathrm{op}(R)$ *on $G \uplus H$.*

Then the polynomial axiom system $P_{iso}(G', H')$ is derivable from $P_{iso}(G, H)$ in EPC_3*, up to a renaming of variables. The number of extension variables used in the derivation is at most $|V(G')|^2$, and the derivation has polynomial size and uses only coefficients with polynomial bit-complexity. Moreover, for every extension axiom $\frac{}{X_f - f}$ used in the derivation, f is of the form $f = X \cdot Y$ or $f = \frac{1}{n} \cdot \left(\sum_{i=1}^{n^2} X_i\right)$.*

The proof is quite lengthy and would interrupt the proof of Theorem 1 at this point; therefore, we first present the lemma and proof that explains how Theorem 1 follows from Lemma 22. Afterwards, we provide the actual polynomial calculus derivations whose existence is claimed in Lemma 22.

▶ **Lemma 23.** *Let G, H be two connected binary τ-structures. Let $p(n)$ be a polynomial and M be a normalised* DWL*-algorithm which produces on input $G \uplus H$ a structure $G' \uplus H'$ with $D(G') \neq D(H')$, such that the length of the run and the size of the structure in the cloud is bounded by $p(|G| + |H|)$ at any time.*
Then the system $P_{iso}(G, H)$ has an EPC_3*-refutation that uses at most $p(|G| + |H|)^3$ many extension variables, has polynomial size and polynomial bit-complexity. Moreover, for every extension axiom $\frac{}{X_f - f}$ used in the derivation, f is of the form $f = X \cdot Y$ or $f = \frac{1}{n} \cdot \left(\sum_{i=1}^{n^2} X_i \right)$.*

Proof. Follows from Lemma 22 together with Lemma 14 and Lemma 9. In detail: Let $(\mathrm{op}_i(R_i))_{i \leq t}$ with $\mathrm{op}_i \in \{\texttt{addPair}, \texttt{contract}\}$ be the sequence of cloud-interaction-operations in the run of M on $G \uplus H$. This sequence of operations produces a sequence of structures $(G \uplus H, G_1 \uplus H_1, G_2 \uplus H_2, ..., G_t \uplus H_t)$ such that $D(G_t) \neq D(H_t)$.
For each i, R_i is a colour in the coarsest coherent configuration of the current structure $A_i := G_i \uplus H_i$ in the cloud. Since M is normalised, $R(C(A_i)) \subseteq V(G_i)^2 \cup V(H_i)^2$. Thus, we can inductively apply Lemma 22 to derive in EPC_3 polynomial axiom systems $P_{\mathrm{iso}}(G_i, H_i)$ for every $i \in [t]$. The induction requires that for every vertex-colour (i.e. diagonal relation), the colour classes always have the same size in G_i and H_i (this is a prerequisite of Lemma 22). This is satisfied because $D(G_i) = D(H_i)$, for $i < t$. Since $D(G_t) \neq D(H_t)$, it follows from Lemma 14 and Lemma 9 that the 1-polynomial is derivable from $P_{\mathrm{iso}}(G_t, H_t)$ in the degree-3 monomial calculus, so in total, it is derivable from $P_{\mathrm{iso}}(G, H)$ in EPC_3. In order to apply Lemma 9 to G_t and H_t, we need to argue that the vertex-colour-classes are of equal size in both graphs, even though $D(G_t) \neq D(H_t)$. If step t is $\texttt{addPair}(R)$, then we introduce equally many new pair-vertices in both graphs because $D(G_{t-1}) = D(H_{t-1})$, so the numbers of R-pairs are equal. If step t is $\texttt{contract}(R)$, we also produce the same number of new vertices. Namely, the number of R-SCCs in $C(G_{t-1})$ and $C(H_{t-1})$ is equal, because by Lemma 21, all R-SCCs have equal size, and by Lemma 19, vertices in R-SCCs receive the same diagonal colour P distinct from all diagonal colours outside SCCs (and $|P(C(G_{t-1}))| = |P(C(H_{t-1}))|$).

Finally, we bound the number of extension variables used in the derivation of $P_{\mathrm{iso}}(G_t, H_t)$: As stated in Lemma 22, for every i, the derivation of $P_{\mathrm{iso}}(G_i, H_i)$ from $P_{\mathrm{iso}}(G_{i-1}, H_{i-1})$ uses at most $|V(G_i)|^2$ many new extension variables. Therefore, the total number of extension variables that are used in the derivation of $P_{\mathrm{iso}}(G_t, H_t)$ is at most $\sum_{i \in [t]} |V(G_i)|^2$. Since $t \leq p(|G|+|H|)$ and $|V(G_i)| \leq p(|G|+|H|)$, for every $i \in [t]$, this sum is at most $p(|G|+|H|)^3$. Similarly we can bound the size and bit-complexity of the derivation: Each time we invoke Lemma 22, we only incur a polynomial cost in size, and this happens polynomially many times. The occurring coefficients can be encoded with polynomially many bits as the lemma asserts. Also, Lemma 22 uses only extension axioms of the required form. ◀

Proof of Lemma 22. First, we have to explain how the variables of the new system $P_{\mathrm{iso}}(G', H')$ are encoded as polynomials in the old variables. Recall that the variable set of $P_{\mathrm{iso}}(G, H)$ is

$$\mathcal{V}(G, H) := \{X_{vw} \mid v \in V(G), w \in V(H), v \text{ and } w \text{ have the same vertex-colour}\}.$$

The intended meaning of X_{vw} is "v is mapped to w". The graphs G', H' contain new vertices, which either represent contracted R-SCCs or pairs of colour R. The set of vertex-pairs (v, w) for which we need new variables is $\mathbf{NewPairs} := (V(G') \setminus V(G)) \times (V(H') \setminus V(H))$.
We would like to map each $(v, w) \in \mathbf{NewPairs}$ to a polynomial $f(vw)$ such that we can represent variables X_{vw} for $(v, w) \in \mathbf{NewPairs}$ as extension variables $X_{f(vw)}$, which we can introduce with the extension axiom $X_{f(vw)} = f(vw)$. If $\mathrm{op} = \texttt{addPair}$ and v is a new pair-vertex, then we let $\mathrm{pair}(v)$ be the vertex-pair that v corresponds to. If $\mathrm{op} = \texttt{contract}$ and v is a new SCC-vertex, then we let $\mathrm{scc}(v)$ denote the set of vertices in the SCC that is contracted into v. We define f as the following injective mapping $f : \mathbf{NewPairs} \to \mathbb{Q}[\mathcal{V}(G, H)]$.

$$f(vw) := \begin{cases} X_{v_1w_1}X_{v_2w_2} & \text{, if op} = \texttt{addPair} \text{ and} \\ & (v_1, v_2) = \text{pair}(v), (w_1, w_2) = \text{pair}(w). \\ \frac{1}{|\text{scc}(v)|} \cdot \sum_{(v',w') \in \text{scc}(v) \times \text{scc}(w)} X_{v'w'} & \text{, if op} = \texttt{contract}. \end{cases}$$

Note that $f(vw)$ is indeed always a polynomial in variables $\mathcal{V}(G, H)$: To see this, we have to check that in the pair-case, the vertex-colours of v_1, w_1 and of v_2, w_2, respectively, are equal, and in the SCC-case, the vertex-colours of all elements of $\text{scc}(v)$ and $\text{scc}(w)$ are equal. In the pair-case, this follows from Corollary 18, and in the SCC-case from Corollary 20.

If v and w are newly introduced pair-vertices with $\text{pair}(v) = (v_1, v_2)$ and $\text{pair}(w) = (w_1, w_2)$, then the variable X_{vw} will be the extension variable for the monomial $X_{v_1w_1}X_{v_2w_2}$. This makes sense because if X_{vw} is set to 1, then the bijection encoded by the assignment maps v to w; but then it also has to map v_1 to w_1 and v_2 to w_2. Similarly, if v and w are new SCC-vertices, then any bijection that takes v to w must also map the elements of $\text{scc}(v)$ to the elements of $\text{scc}(w)$ in any possible way. This is reflected in our representation of X_{vw} as the "average" over all possible mappings from $\text{scc}(v)$ to $\text{scc}(w)$.

Here are the new polynomial axioms that we have to derive in order to go from $P_{\text{iso}}(G, H)$ to $P_{\text{iso}}(G', H')$:

$$\sum_{v \in V(G') \setminus V(G)} X_{f(vw)} - 1 \qquad \text{for all } w \in V(H') \setminus V(H). \tag{4}$$

$$\sum_{w \in V(H') \setminus V(H)} X_{f(vw)} - 1 \qquad \text{for all } v \in V(G') \setminus V(G). \tag{5}$$

$$X_{f(vw)}X_{v'w'} \qquad \begin{array}{l} \text{for all } v, v' \in V(G'), w, w' \in V(H') \\ \text{such that } (v, w) \in \textbf{NewPairs} \\ \text{and } v', w' \in V(G) \cup V(H), v' \sim w' \\ \text{and } \{(v, w), (v', w')\} \text{ is not} \\ \text{a local isomorphism.} \end{array} \tag{6}$$

$$X_{f(vw)}X_{f(v'w')} \qquad \begin{array}{l} \text{for all } v, v' \in V(G'), w, w' \in V(H') \\ \text{such that } (v, w) \in \textbf{NewPairs} \\ \text{and } (v', w') \in \textbf{NewPairs} \\ \text{and } \{(v, w), (v', w')\} \text{ is not} \\ \text{a local isomorphism.} \end{array} \tag{7}$$

The relation $\sim$ is the same-colour-relation, as in Definition 7. Note that Axioms (4) and (5) only sum over vertices of the same colour as w and v, respectively (as Axioms (1) and (2) do), because $V(G') \setminus V(G)$ and $V(H') \setminus V(H)$ are the sets of newly added vertices. These vertices receive a new colour distinct from all other vertex colours in G and H (see definition of the DWL-operations in Section 7.2).

The next step is to verify that $P_{\text{iso}}(G', H')$ is indeed derivable from $P_{\text{iso}}(G, H)$. This is quite lengthy but does not require any substantial new arguments beyond the lemmas from Section 8. Therefore, we only show the derivation of Axioms (4) and (5) here. The derivation of the other axioms is similar and can be found in the full version of the paper [24].

Derivation of Axioms (4) and (5). Fix $w \in V(H') \setminus V(H)$. We show how to derive Axiom (4) for w. Two cases have to be distinguished, namely whether op = `addPair` or op = `contract`.

Case 1: op = `addPair`: Let $(w_1, w_2) = \text{pair}(w)$. Using the multiplication rule and linear combinations, we derive from Axiom (1) for w_2 in $P_{\text{iso}}(G, H)$:

$$\left(\sum_{\substack{v_1 \in V(G),\\ v_1 \sim w_1}} X_{v_1 w_1}\right) \cdot \left(\sum_{\substack{v_2 \in V(G),\\ v_2 \sim w_2}} X_{v_2 w_2} - 1\right)$$

$$= \sum_{\substack{v_1, v_2 \in V(G)\\ v_1 \sim w_1,\\ v_2 \sim w_2}} X_{v_1 w_1} X_{v_2 w_2} - \sum_{\substack{v_1 \in V(G),\\ v_1 \sim w_1}} X_{v_1 w_1}$$

Recall from the statement of Lemma 22 that $R \in \sigma$ is the colour such that op(R) is executed to obtain $G' \uplus H'$ from $G \uplus H$. Further, let $C = C(G \uplus H)$.
Since $(w_1, w_2) \in R(C)$, we can use Lemma 15 and Lemma 9 to derive from $P_{\text{iso}}(G, H)$ all monomials $X_{v_1 w_1} X_{v_2 w_2}$ where $(v_1, v_2) \notin R(C)$. Hence, we may cancel these monomials from the above sum with the linear combination rule. This yields:

$$\sum_{\substack{v_1, v_2 \in V(G)\\ (v_1, v_2) \in R(C)}} X_{v_1 w_1} X_{v_2 w_2} - \sum_{\substack{v_1 \in V(G)\\ v_1 \sim w_1}} X_{v_1 w_1}.$$

Here, we used that for all pairs $(v_1, v_2) \in V(G)^2 \cap R(C(A))$, it holds that $v_1 \sim w_1$ and $v_2 \sim w_2$. This follows from Corollary 18 and the fact that vertex-colours are represented by diagonal relations. Now we are almost done: We add Axiom (1) for w_1 to the above expression and replace each remaining monomial $X_{v_1 w_1} X_{v_2 w_2}$ with the new extension variable $X_{f(vw)}$, where $v \in V(G')$ is the respective new pair-vertex with $\text{pair}(v) = (v_1, v_2)$. One can see that

$$V(G') \setminus V(G) = \{v \in V(G') \mid \text{pair}(v) \in R(C) \cap V(G)^2\}.$$

Thus, we have indeed derived Axiom (4) for w.
Similarly, we get Axiom (5) for a vertex $v \in V(G') \setminus V(G)$ if we perform the same derivations from the Axioms (2) instead of (1).

Case 2: op = `contract`: We derive Axiom (4) for a fixed vertex $w \in V(H') \setminus V(H)$. Now w is a vertex that represents a contracted R-SCC $\text{scc}(w) \subseteq V(H)$.
For every vertex $w' \in \text{scc}(w)$, we have Axiom (1) for w' in $P_{\text{iso}}(G, H)$:

$$\sum_{\substack{v' \in V(G),\\ v' \sim w'}} X_{v' w'} - 1.$$

Now from this, we may cancel all $X_{v' w'}$ where (v', v') and (w', w') are in distinct diagonal relations in C. This is done again by deriving the respective variables $X_{v' w'}$ with Lemma 15 and Lemma 9. After that step, we have for each $w' \in \text{scc}(w)$:

$$\sum_{\substack{v' \in V(G),\\ \text{there is } v \in V(G') \setminus V(G) \text{ with } v' \in \text{scc}(v)}} X_{v' w'} - 1. \qquad (\star)$$

This holds because the vertex $v' \in V(G)$ has the same diagonal colour as $w' \in \text{scc}(w)$ in the coherent configuration C if and only if it is also contained in some R-SCC (Lemma 19).

Next, we use the variable introduction rule and introduce the variables $X_{f(vw)}$ for every $v \in V(G') \setminus V(G)$. That means, we obtain the following polynomials:

$$\frac{1}{|\mathrm{scc}(v)|} \sum_{(v',w') \in \mathrm{scc}(v) \times \mathrm{scc}(w)} X_{v'w'} - X_{f(vw)} \qquad \text{for each } v \in V(G') \setminus V(G).$$

Now take the sum of all polynomials ($\star$) for all $w' \in \mathrm{scc}(w)$, multiplied by $\frac{1}{|\mathrm{scc}(w)|}$. From this, subtract the above polynomials for all $v \in V(G') \setminus V(G)$. This yields Axiom (4) for the vertex w because we have $\frac{1}{|\mathrm{scc}(v)|} = \frac{1}{|\mathrm{scc}(w)|}$ for all $v \in V(G') \setminus V(G)$, since all R-SCCs have equal size (Lemma 21). In a similar way we can derive Axiom (5) for an SCC-vertex $v \in V(G') \setminus V(G)$.

Due to space limitations, we omit the derivation of Axioms (6) and (7) in this version of the paper. The key idea is again to use that certain vertex pairs receive distinct colours in C and thus, the corresponding monomials can be cancelled with Lemma 15 and Lemma 9. For this, we also need some more properties of coherent configurations, which are not difficult to prove using the correspondence between coherent configurations, 2-WL, and counting logic.

In total, we can derive $P_{\mathrm{iso}}(G', H')$ from $P_{\mathrm{iso}}(G, H)$. The number of new variables is clearly bounded by $|V(G')|^2$. It is also not difficult to see that only polynomially many monomials occur in the derivation, and the binary encoding of the coefficients occurring in them has complexity at most $\mathcal{O}(\log |V(G)|)$ (this holds also in the omitted cases). The used extension axioms are all for polynomials that are averaged sums or degree-2 monomials, as mentioned in Lemma 22. The derivations obtained with Lemma 9 also have polynomial complexity because they can be carried out in MC_3. ◀

10 Discussion and future work

We have shown that the degree-3 extended polynomial calculus can simulate the pair- and contract-operations of Deep Weisfeiler Leman in the sense that the axiom system $P_{\mathrm{iso}}(G', H')$ is derivable from $P_{\mathrm{iso}}(G, H)$ if there is a sequence of DWL-operations that transforms $G \uplus H$ into $G' \uplus H'$. Together with the simulation of k-dimensional Weisfeiler Leman in the degree-k monomial calculus given in [3], this shows that EPC_3 can distinguish two graphs G and H if they can be distinguished in DWL, and the EPC_3-refutation has the same complexity as the DWL-algorithm. Since DWL-algorithms and CPT-programs mutually simulate each other, this result upper-bounds the graph distinguishing power of CPT by that of EPC_3.

This raises the question whether a super-polynomial lower bound for graph isomorphism can be established for EPC_3, preferably for graph classes such as unordered CFI-graphs or multipedes, whose isomorphism problem reduces to a linear equation system and is thus in Ptime. If such a lower bound is found, then by Theorem 2, we would also have that CPT $\neq$ Ptime. This would be a huge step forward in understanding the limitations of symmetry-invariant computation and thus in the quest for a logic for Ptime.

Unfortunately, we do not know how strong the system EPC_3 is, and in particular, if the degree-restriction is a true limitation. It may even be the case that EPC_3 polynomially simulates the *unbounded-degree* extended polynomial calculus. Then it would be as strong as extended Frege because in EPC, the extension variables can encode arbitrary polynomials and thereby arbitrary Boolean circuits. This would make it less useful for proving lower bounds against CPT, as extended Frege lower bounds seem to be out of reach at the moment. However, Theorem 1 also asserts that the simulation of CPT is possible using only extension axioms of a limited form, namely for degree-2 monomials and averaged sums. In this restricted

version of EPC_3, the obvious representation of Boolean circuits as polynomials is no longer possible: The Boolean functions $X \wedge Y, X \vee Y$, and $\neg X$ can naturally be represented as the polynomials $X \cdot Y, X + Y - X \cdot Y$, and $1 - X$. When we represent Boolean circuits using extension variables, then each extension variable corresponds to a gate in the circuit. If the only allowed extension axioms are $\frac{}{X_f - f}$ for $f = X \cdot Y$ or $f = \frac{1}{n}\sum_{i=1}^{n^2} X_i$, then the only gates that we can naturally express are AND-gates (with extension axioms of the first type). Neither NOT-gates nor OR-gates can be simulated (directly) by such extension axioms because this requires sums which are not of the form $\frac{1}{n}\sum X_i$. In particular, these extension axioms cannot be applied to polynomials where variables occur with a negative coefficient. Hence, the corresponding circuits are in some sense *monotone*. This is of course no formal proof that EPC_3 with restricted extension axioms is strictly weaker than extended Frege but at least it rules out the natural simulation of Boolean circuits in EPC_3. In total, the success chances of our suggested approach for CPT lower bounds via proof complexity depend highly on the true power of EPC_3 (with restricted extension axioms), and its relation to unrestricted EPC. Investigating this remains a problem for future work.

Symmetry-invariance of the refutations

Actually, our Theorem 1 could be strengthened more: A simulation of CPT in EPC_3 is even possible in a certain *symmetry-invariant* fragment of EPC_3. However, it seems tricky to give a precise definition of "*symmetric* EPC_3" that is both natural and fits the kind of symmetry we encounter in our CPT-simulation. A neat way to put it would be to say that the set of extension axioms used in a derivation has to be closed under symmetries. With the right definition of "symmetries", this is indeed true for the refutation constructed in Lemma 22. Namely, whenever an extension variable $X_{f(vw)}$ is introduced, where v and w are new pair- or SCC-vertices, then we introduce it for *all* $(v, w) \in V(G') \times V(H')$ that are new. The corresponding polynomials $f(vw)$ consist of variables that refer to the vertices in the respective pairs or SCCs of v and w. The automorphisms of the graphs G and H preserve the colours of all vertex-pairs in the coarsest coherent configuration. Therefore, the set of extension axioms that we introduce in each step of the refutation is closed under the automorphisms of G and H. The action of these automorphism groups on the set of variables of $P_{\mathrm{iso}}(G, H)$ is the natural one, i.e. if π is an automorphism of G and σ an automorphism of H, then they take X_{vw} to $X_{\pi(v)\sigma(w)}$. This extends naturally to the extension axioms, so for example, if v and w are pair-vertices with $\mathrm{pair}(v) = (v_1, v_2), \mathrm{pair}(w) = (w_1, w_2)$, then the extension axiom $X_{f(vw)} - X_{v_1w_1}X_{v_2w_2}$ is mapped to $X_{f(v'w')} - X_{\pi(v_1)\sigma(w_1)}X_{\pi(v_2)\sigma(w_2)}$, where v', w' are the newly introduced pair-vertices for $(\pi(v_1), \pi(v_2))$ and $(\sigma(w_1), \sigma(w_2))$ (such v', w' must exist because DWL is isomorphism-invariant and introduces new vertices for all pairs with the same colour). So in this sense, the extension axioms used in Lemma 22 are closed under all automorphism-pairs $(\pi, \sigma) \in \mathbf{Aut}(G) \times \mathbf{Aut}(H)$.
Unfortunately, this does not lead to a *general* definition of symmetric EPC_3 because it depends on the automorphisms of G and H, the graphs which are implicitly encoded in $P_{\mathrm{iso}}(G, H)$. When EPC_3 is applied to other polynomial axiom systems, then there might be no graphs "in the background". So for a general set of input polynomials $\mathcal{P}$, it would be natural to require that the set of extension axioms in a refutation be closed under the automorphisms of $\mathcal{P}$ – these are the permutations of the variables that extend to permutations of the polynomials in $\mathcal{P}$. However, this would no longer fit to our derivation from Lemma 22: The system $P_{\mathrm{iso}}(G, H)$ in general has more automorphisms than $\mathbf{Aut}(G) \times \mathbf{Aut}(H)$. Namely, $P_{\mathrm{iso}}(G, H)$ contains no information about where the edges and non-edges in G and H actually are; it just relates pairs $(v, v') \in V(G)^2$ with pairs $(w, w') \in V(H)^2$ where (v, v') is an edge

and (w, w') is not, or vice versa (Axiom (3)). Therefore, an automorphism of $P_{\text{iso}}(G, H)$ may swap all edges with non-edges, as long as it does so in both G and H (such examples can be constructed). But the automorphisms of the graphs must preserve edges and non-edges, so such an automorphism of $P_{\text{iso}}(G, H)$ does not correspond to one from $\mathbf{Aut}(G) \times \mathbf{Aut}(H)$. Our constructed refutation is only symmetric with respect to the latter. Thus, our simulation of CPT in EPC_3 is possible in a way that respects *specific* symmetries of $P_{\text{iso}}(G, H)$, but we do not know if this kind of symmetry-invariance can be formulated independently of the graph isomorphism problem as a general restriction to the proof system EPC_3. Perhaps future research will reveal a more generic way to define symmetrized versions of known proof systems. This could be of independent interest because it might be possible to prove lower bounds for symmetric versions of proof systems for which non-symmetric lower bounds seem to be out of reach.

Finally, another question that we have not answered is whether the converse to Theorem 1 also holds: Is there an algorithm that can find EPC_3-refutations (for graph isomorphism) and can be implemented in CPT? Since CPT is symmetry-invariant and EPC_3 is not, this seems unlikely. Furthermore, such a proof search algorithm would probably have to be non-deterministic. Thus, the only way to get an exact match in expressive power between the logic and the proof system might be by restricting EPC_3 to a symmetry-invariant fragment and extending CPT with some kind of non-determinism.

References

1 Faried Abu Zaid, Erich Grädel, Martin Grohe, and Wied Pakusa. Choiceless Polynomial Time on structures with small Abelian colour classes. In *Mathematical Foundations of Computer Science 2014*, volume 8634 of *Lecture Notes in Computer Science*, pages 50–62. Springer, 2014. URL: `http://logic.rwth-aachen.de/pub/pakusa/cptcan.pdf`.

2 Yaroslav Alekseev. A Lower Bound for Polynomial Calculus with Extension Rule. In Valentine Kabanets, editor, *36th Computational Complexity Conference (CCC 2021)*, volume 200 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CCC.2021.21`.

3 Christoph Berkholz and Martin Grohe. Limitations of algebraic approaches to graph isomorphism testing. In *International Colloquium on Automata, Languages, and Programming*, pages 155–166. Springer, 2015.

4 Andreas Blass, Yuri Gurevich, and Saharon Shelah. Choiceless polynomial time. *Annals of Pure and Applied Logic*, 100(1-3):141–187, 1999.

5 J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992.

6 Ashok K Chandra and David Harel. Structure and complexity of relational queries. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 333–347. IEEE, 1980. `doi:10.1109/SFCS.1980.41`.

7 Gang Chen and Ilia Ponomarenko. Lectures on coherent configurations. *Lecture notes*, 2019. Available at `http://www.pdmi.ras.ru/~inp/ccNOTES.pdf`.

8 Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 174–183, 1996.

9 Anuj Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1):8–21, 2015.

10 Anuj Dawar, David Richerby, and Benjamin Rossman. Choiceless polynomial time, counting and the Cai–Fürer–Immerman graphs. *Annals of Pure and Applied Logic*, 152(1-3):31–50, 2008.

11 Susanna F de Rezende, Massimo Lauria, Jakob Nordström, and Dmitry Sokolov. The power of negative reasoning. In *36th Computational Complexity Conference (CCC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

12 E. Grädel, W. Pakusa, S. Schalthöfer, and L. Kaiser. Characterising choiceless polynomial time with first-order interpretations. In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 677–688, 2015.

13 Erich Grädel and Martin Grohe. Is polynomial time choiceless? In *Fields of Logic and Computation II*, pages 193–209. Springer, 2015.

14 Erich Grädel, Martin Grohe, Benedikt Pago, and Wied Pakusa. A finite-model-theoretic view on propositional proof complexity. *Logical Methods in Computer Science*, 15, 2019.

15 Martin Grohe. The quest for a logic capturing PTIME. In *2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 267–271. IEEE, 2008. doi:10.1109/LICS.2008.11.

16 Martin Grohe, Pascal Schweitzer, and Daniel Wiebking. Deep Weisfeiler Leman, 2020. arXiv:2003.10935.

17 Yuri Gurevich. Logic and the Challenge of Computer Science. In *Current Trends in Theoretical Computer Science*. Computer Science Press, 1988.

18 Yuri Gurevich and Saharon Shelah. On finite rigid structures. *The Journal of Symbolic Logic*, 61(2):549–562, 1996.

19 Lauri Hella. Logical hierarchies in PTIME. *Information and Computation*, 129(1):1–19, 1996.

20 Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In *Complexity theory retrospective*, pages 59–81. Springer, 1990.

21 Sandra Kiefer. The Weisfeiler-Leman algorithm: an exploration of its power. *ACM SIGLOG News*, 7(3):5–27, 2020.

22 Moritz Lichter. Separating rank logic from polynomial time. *CoRR*, abs/2104.12999, 2021. arXiv:2104.12999.

23 Benedikt Pago. Choiceless Computation and Symmetry: Limitations of Definability. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:21, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2021.33.

24 Benedikt Pago. Finite model theory and proof complexity revisited: Distinguishing graphs in Choiceless Polynomial Time and the Extended Polynomial Calculus. *arXiv*, 2022. arXiv:2206.05086.

25 Wied Pakusa. *Linear Equation Systems and the Search for a Logical Characterisation of Polynomial Time*. PhD thesis, RWTH Aachen, 2015.

26 Benjamin Rossman. Choiceless computation and symmetry. In *Fields of logic and computation*, pages 565–580. Springer, 2010.

27 Svenja Schalthöfer. *Choiceless Computation and Logic*. PhD thesis, RWTH Aachen, 2020.

Adding Transitivity and Counting to the Fluted Fragment

Ian Pratt-Hartmann
Department of Computer Science, University of Manchester, UK
Institute of Computer Science, University of Opole, Poland

Lidia Tendera
Institute of Computer Science, University of Opole, Poland

Abstract

We study the impact of adding both counting quantifiers and a single transitive relation to the fluted fragment – a fragment of first-order logic originating in the work of W.V.O. Quine. The resulting formalism can be viewed as a multi-variable, non-guarded extension of certain systems of description logic featuring number restrictions and transitive roles, but lacking role-inverses. We establish the finite model property for our logic, and show that the satisfiability problem for its k-variable sub-fragment is in $(k{+}1)$-NExpTime. We also derive ExpSpace-hardness of the satisfiability problem for the two-variable, fluted fragment with one transitive relation (but without counting quantifiers), and prove that, when a second transitive relation is allowed, both the satisfiability and the finite satisfiability problems for the two-variable fluted fragment with counting quantifiers become undecidable.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases fluted logic, transitivity, counting, satisfiability, decidability, complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.32

Funding This work was supported by the Polish NCN, grant number 2018/31/B/ST6/03662.

1 Introduction

The *fluted fragment*, or $\mathcal{FL}$, is a fragment of first-order logic in which, roughly speaking, the sequence of quantification of variables coincides with the order in which those variables appear as arguments of predicates. The *fluted fragment with counting*, or $\mathcal{FLC}$, is the extension of $\mathcal{FL}$ with the standard counting quantifiers $\exists_{[\leq M]}$, $\exists_{[\geq M]}$ and $\exists_{[=M]}$, where M is a (numeral denoting a) positive integer. The following sentence is in $\mathcal{FLC}$:

$$\begin{array}{l}\text{At most three lecturers introduce a professor to at least five students}\\ \exists_{[\leq 3]}x_1\Big(\mathrm{lectr}(x_1)\wedge\exists x_2\big(\mathrm{prof}(x_2)\wedge\exists_{[\geq 5]}x_3(\mathrm{std}(x_3)\wedge\mathrm{intro}(x_1,x_2,x_3))\big)\Big)\end{array}\tag{1}$$

It was shown in [31] that $\mathcal{FL}$ has the finite model property, whence its satisfiability problem is decidable. The complexity bound of NExpTime claimed in [32] is incorrect, and as shown later in [28], the problem is non-elementary; however, the satisfiability problem for the k-variable sub-fragment of $\mathcal{FL}$ is known to be in $(k{-}2)$-NExpTime for $k \geq 3$ [29]. This result extends to the fragment $\mathcal{FLC}$, though with a best-known upper complexity bound of $(k{-}1)$-NExpTime for k in the same range [27].

It is impossible, within the fluted fragment, to express the property of transitivity: in particular, the formula $\forall x_1\forall x_2\big(r(x_1,x_2)\rightarrow\forall x_3\big(r(x_2,x_3)\rightarrow r(x_1,x_3)\big)\big)$ is not fluted, because the variable sequence in the atom $r(x_1,x_3)$ omits x_2. The question therefore arises as to whether the fragments $\mathcal{FL}$ or even $\mathcal{FLC}$ can be extended by declaring certain distinguished binary predicates to be interpreted as transitive relations. For $\mathcal{FL}$, this question was largely resolved in [30]. It was shown that $\mathcal{FL}$ with equality and one distinguished transitive

relation lacks the finite model property, but has decidable satisfiability and finite satisfiability problems; the former problem, restricted to the k-variable sub-fragment is shown to be in k-NExpTime, and the latter in $(k{+}1)$-NExpTime. In the presence of two transitive relations but without equality, the fluted fragment loses the finite model property, with the decidability of satisfiability and finite satisfiability both unknown. With either two transitive relations and equality or three transitive relations, satisfiability and finite satisfiability are both undecidable. In the present paper, we consider the combination of transitivity and counting. We show that, in the absence of equality, we may add a single transitive relation to the fragment $\mathcal{FLC}$ without losing the finite model property, and we establish an upper complexity bound of $(k{+}1)$-NExpTime on the satisfiability problem for the k-variable sub-fragment. In the presence of two transitive relations, however, the satisfiability and finite satisfiability problems for $\mathcal{FLC}$ are undecidable, even in the absence of equality.

The impact of counting quantifiers and transitivity on decidability and complexity of satisfiability has been widely studied in the context of other fragments of first-order logic. Of particular interest in this regard are *propositional modal logics* and *description logics*, as these are mapped to the fluted fragment via the standard translation. The simplest of these logics is the propositional modal logic K, in which the modalities $\Box$ and $\Diamond$ are interpreted with respect to a single accessibility relation. By allowing counting modalities of the form $\Diamond_{\leq n}$, $\Diamond_{=n}$ and $\Diamond_{\geq n}$, we obtain propositional graded modal logic GrK; and by imposing on these systems the requirement that the accessibility relation be interpreted as a *transitive* relation, we obtain, respectively, the logics known as K4 and GrK4. Thus, GrK4 is a sub-fragment of $\mathcal{FLC}$ with one transitive relation. The satisfiability problems for K, GrK and K4 are all PSpace-complete [25, 36]; the corresponding problem for GrK4 is NExpTime-complete [17]. It is instructive to consider the further extension of these logics with the *converse* modalities $\boxminus$ and $\Diamondinv$. Denoting the extension of K with converse modalities by K^c, and similarly for the other fragments, we find that the satisfiability problems for K^c, GrK^c and $\mathrm{K4}^c$ all remain in PSpace; but the corresponding problem for $\mathrm{GrK4}^c$, is undecidable (see [2, 37]). Thus, transitivity and counting can be combined in the context of modal logics; but decidability of satisfiability is lost when converse modalities are added. Yet converse modalities are precisely what we cannot express in fluted logic. It is therefore natural to ask whether it is not in fact fluting that is the critical factor here.

The basic description logic $\mathcal{ALC}$ is a notational variant of propositional multi-modal logic. Extensions of $\mathcal{ALC}$ are defined by allowing additional constructs, in particular: *number restrictions* (denoted $\mathcal{Q}$) corresponding to counting quantifiers as defined in this paper, *transitivity* of roles (denoted $\mathcal{S}$), role hierarchies ($\mathcal{H}$) corresponding to inclusions of binary relations, *nominals* ($\mathcal{O}$) and role "inverses" ($\mathcal{I}$).[1] The logic $\mathcal{SHOQ}$ constitutes a maximal description logic that can be embedded into $\mathcal{FLC}$ with transitive relations. Now it is known [15] that an unrestricted combination of number restrictions and transitivity leads to undecidability of concept satisfiability even in the smaller logic $\mathcal{SHQ}$. Indeed, it was shown in [18] that just three roles (two of them transitive) are sufficient for undecidability. In response to these negative results, description logics standardly impose the syntactic restriction that transitive roles or their super-roles do not appear in number restrictions. With these restrictions, decidability is restored: concept satisfiability for $\mathcal{SHOQ}$ is ExpTime-complete, and for $\mathcal{SHOIQ}$ it is NExpTime-complete [35]. On the other hand, there is no problem with allowing transitive relations to appear under number restrictions in description logics too weak to allow these roles to interact with each other. Thus, for example, concept

[1] In description logic, the term "inverse" has become established instead of the more normal "converse".

satisfiability for $\mathcal{SOQ}$ is decidable [16]: hardness is inherited from GrK4 [17] and the optimal upper bound has been recently shown in [13]. In the logic considered in the present paper, only one transitive relation is available, but it is allowed to combine freely with other relations. We show that this comparative freedom is, from the point of view of decidability of satisfiability, unproblematic as long as we remain within the confines of fluted logic – in effect, provided we have no access to role inverses.

In knowledge representation and database theory an important reasoning problem is the query entailment problem over incomplete databases enriched by ontologies described in some logic $\mathcal{L}$. This problem reduces to unsatisfiability of the conjunction $\varphi \wedge \neg q$, where φ describes the ontology and q is a Boolean query. So, (finite) query entailment under $\mathcal{L}$ is computationally no easier than the corresponding (finite) unsatisfiability problem. Decidability of query entailment for various fragments of restricted $\mathcal{SHOIQ}$ has already been shown [9, 10, 5, 11]. Finite entailment for $\mathcal{SHOIQ}$ was shown undecidable in [33], leaving decidability of the general entailment problem open. Since in real-life ontologies it is natural to find properties where transitivity interacts with counting (see e.g. [14]) the search for logics with decidable (finite) query entailment allowing such an interaction has a strong practical motivation. This quest has brought some positive results: decidability in 2-ExpTime of entailment of *regular path queries* for $\mathcal{SQ}$ [14], a corresponding lower bound from [12] as well as decidability of both general and finite entailment for $\mathcal{SQO}$ and for a restricted fragment of $\mathcal{SIQ}$, where inverse roles are not used under number restrictions. In the fluted fragment query entailment is decidable provided some form of guardedness is added [1].

The quest for decidable logics with counting and/or transitivity is not, of course, limited to description or modal logics. We mention in this context the *guarded fragment*, the *unary negation fragment* and the recently introduced *triguarded fragment*, each of them being as a different generalization of propositional modal logic (not subsuming fluted logic), enjoying the finite model property, and not being able to express transitivity or related properties. Considerable work has been done to understand the limits of decidability for extensions of these fragments where (restricted) transitivity is allowed. The general picture that emerges is that, when considering more than one transitive relation, the interaction between them must be restricted in order to secure decidability of satisfiability. For instance, satisfiability is undecidable for the the two-variable guarded fragment with two transitive relations [19], but is restored even in the presence of any number of transitive relations if the transitive relations are allowed to appear only in guard positions [34]. Similarly, satisfiability is undecidable for the two-variable guarded fragment over signatures with two *linear* orders, but is restored if we restrict the linear orders to guard positions and in addition forbid any other binary relations in the signature (a property analogous to the restriction of number restrictions to simple roles in description logics) [21]. Technically, the crucial property behind the decision procedures for the satisfiability problem in these cases is that satisfiable formulas exhibit some kind of tree-like models in which the transitive relations are defined independently of each other. For a wider picture we refer the reader to the recent papers on extensions of: the guarded fragment [24], the unary negation fragment [7, 8], the triguarded fragment [22, 23] and the references therein.

The rest of the paper is organised as follows. After defining the main notions in Section 2 we begin in Section 3 by showing the ExpSpace-hardness of the satisfiability problem for the two-variable fluted logic with one transitive relation. In Section 4 we discuss *Presburger quantifiers*, which are a generalization of counting quantifiers. In Section 5 we show that the two-variable fragment of $\mathcal{FLC}$ with one transitive relation has the finite model property

when restricted to signatures consisting of unary predicates and one distinguished transitive relation. In Section 6 we first lift this result to arbitrary signatures, and then drop the restriction to two-variables. In Section 7 we show that the satisfiability and finite satisfiability problems for $\mathcal{FLC}$ with two transitive relations are undecidable even in the two-variable case.

2 Preliminaries

In the sequel, we use Fraktur letters for structures and the corresponding Roman letters for their domains: thus, A is the domain of $\mathfrak{A}$ *etc.* Logical variables are taken from the sequence $\bar{x}_\omega = x_1, x_2, \ldots$, and all signatures are purely relational, i.e., there are no individual constants or function symbols. We begin by establishing the syntax of the fragment $\mathcal{FLC}$, *the fluted fragment with counting*. Define the sets of formulas $\mathcal{FLC}^{[k]}$, for all $k \geq 0$, by simultaneous structural recursion as follows:

- **(i)** any atom $p(x_\ell, \ldots, x_k)$, where $x_\ell, \ldots, x_k$ is a contiguous subsequence of $\bar{x}_\omega$ and p a (non-equality) predicate of arity $k - \ell + 1$, is in $\mathcal{FLC}^{[k]}$;
- **(ii)** $\mathcal{FLC}^{[k]}$ is closed under boolean combinations;
- **(iii)** if φ is in $\mathcal{FLC}^{[k+1]}$, then $\exists x_{k+1}\, \varphi$ and $\forall x_{k+1}\, \varphi$ are in $\mathcal{FLC}^{[k]}$,
- **(iv)** if φ is in $\mathcal{FLC}^{[k+1]}$ and M a non-negative integer, then $\exists_{[\leq M]} x_{k+1}\, \varphi$, $\exists_{[\geq M]} x_{k+1}\, \varphi$ and $\exists_{[=M]} x_{k+1}\, \varphi$ are in $\mathcal{FLC}^{[k]}$.

It is intended that Clause (i) allows the case $\ell = k{+}1$ (empty sequence of arguments), so that the atoms in question are proposition letters. Clause (iv) speaks of *non-negative integers* occurring in formulas; when calculating the size $\|\varphi\|$ of an $\mathcal{FLC}^k$-formula φ, we assume these to be represented as bit-strings (i.e. binary coding of counting subscripts). Define $\mathcal{FL}^{[k]}$ to be the fragment of $\mathcal{FLC}^{[k]}$ in which no counting quantifiers occur, i.e. Clause (iv) is dropped. Define the fragment $\mathcal{FLC}$ to be the union $\bigcup_{k\geq 0} \mathcal{FLC}^{[k]}$; similarly $\mathcal{FL} = \bigcup_{k\geq 0} \mathcal{FL}^{[k]}$. By $\mathcal{FLC}{+}1\mathrm{Tr}$, we understand the same set of formulas as $\mathcal{FLC}$, but with a distinguished binary predicate $\mathfrak{t}$, which is required to be interpreted as a transitive relation; similarly for $\mathcal{FL}+1\mathrm{Tr}$. Finally, define $\mathcal{FLC}^k$ to be the fragment of $\mathcal{FLC}$ in which at most the variables $x_1, \ldots, x_k$ appear (free or bound); and similarly for $\mathcal{FL}^k$, $\mathcal{FLC}^k{+}1\mathrm{Tr}$ and $\mathcal{FL}^k + 1\mathrm{Tr}$. Incidentally, do not confuse $\mathcal{FLC}^k$ with $\mathcal{FLC}^{[k]}$: for example, formula (1) is in $\mathcal{FLC}^k$ for all $k \geq 3$ but in $\mathcal{FLC}^{[k]}$ only for $k = 0$. By *sentence*, we mean a formula with no free variables – that is, up to a shift of variable indices, a formula of $\mathcal{FLC}^{[0]}$.

Assuming, as we shall, that the arity of any predicate is fixed in advance, variables in fluted logic convey no information at all, and therefore can be omitted. For example, formula (1) can be unambiguously reconstructed – up to a shift of variable indices – from

$$\exists_{[\leq 3]} \Big(\mathrm{lectr} \wedge \exists \big(\mathrm{prof} \wedge \exists_{[\geq 5]} (\mathrm{std} \wedge \mathrm{intro}) \big) \Big). \tag{2}$$

Consequently, we employ this variable-free notation for $\mathcal{FLC}$ in the sequel, as it is more compact. The issue of ambiguity with respect to shifts in variable indexing bears emphasis, however. Under variable-free notation, any formula of $\mathcal{FLC}^{[k]}$ is, *without lexical change*, also a formula of $\mathcal{FLC}^{[k+1]}$. For example, the sub-formula $\exists(\mathrm{prof} \wedge \exists_{[\geq 5]}(\mathrm{std} \wedge \mathrm{intro}))$ of (2) may be reconstructed not only as the $\mathcal{FLC}^{[1]}$-formula $\varphi(x_1)$ given by $\exists x_2(\mathrm{prof}(x_2) \wedge \exists_{[\geq 5]} x_3(\mathrm{std}(x_3) \wedge \mathrm{intro}(x_1, x_2, x_3)))$, but also as the (logically equivalent) $\mathcal{FLC}^{[2]}$-formula $\varphi'(x_1, x_2)$ given by $\exists x_3(\mathrm{prof}(x_3) \wedge \exists_{[\geq 5]} x_4(\mathrm{std}(x_4) \wedge \mathrm{intro}(x_2, x_3, x_4)))$, and so on. This feature will actually be useful in Sec. 6, when we come to reduce the number of variables in $\mathcal{FLC}{+}1\mathrm{Tr}$-formulas.

Using variable-free notation, we say that an $\mathcal{FLC}^k$-formula ($k \geq 2$) is in *normal-form* if it conforms to the pattern

$$\bigwedge_{h=1}^{m} \forall^{k-1}(\alpha_h \to \exists_{[\lesseqgtr_h M_h]}\, \beta_h), \tag{3}$$

where the α_h are quantifier-free $\mathcal{FLC}^{k-1}$-formulas, the β_h are quantifier-free $\mathcal{FLC}^k$-formulas, the M_h are non-negative integers and the symbols $\lesseqgtr_h$ are chosen from the set $\{\leq, \geq, =\}$.

▶ **Lemma 1.** *Let φ be a sentence of $\mathcal{FLC}^k$ ($k \geq 2$). Then we may compute, in time bounded by a polynomial function of $\|\varphi\|$, a normal-form $\mathcal{FLC}^k$-sentence ψ such that:* (i) $\models \psi \to \varphi$*; and* (ii) *any model of φ can be expanded to a model of ψ.*

Proof. The proof is essentially the same as for $\mathcal{FL}$, via standard re-writing techniques. See, e.g. [29, Lemma 4.1]. Note that a formula of the form $\forall^k\theta$, with $\theta \in \mathcal{FLC}^k$ quantifier-free, is logically equivalent to the normal-form conjunct $\forall^{k-1}(\top \to \exists_{[=0]}\neg\theta)$. ◀

If $\mathcal{L}$ is any language, we denote the satisfiability problem for $\mathcal{L}$ by $\mathrm{Sat}(\mathcal{L})$. Since all of the problems $\mathrm{Sat}(\mathcal{L})$ with which we shall be concerned have relatively high complexity (well above NPTime), we may assume henceforth that proposition letters do not occur in formulas, as their truth-values can simply be guessed.

In this paper, we shall make some limited use of Presburger arithmetic, the first-order theory of the structure $\mathfrak{N} = (\mathbb{N}, <, 0, 1, +)$, where $\mathbb{N} = \{0, 1, 2 \ldots\}$ is the domain of natural numbers, and the symbols $<$, 0, 1, $+$ have their usual interpretations. A *Presburger formula* is any formula Θ in this language. We call Θ *existential* if it is of the form $\exists\bar{v}\,\Xi$, where $\bar{v}$ is a (possibly empty) tuple of variables and Ξ is quantifier-free. If $\bar{a} = a_1, \ldots, a_k$ is a tuple of numbers and M a number, we write $\bar{a} \leq M$, to mean $a_i \leq M$ for all i ($1 \leq i \leq k$). If Θ is a Presburger formula featuring only the variables $\bar{w}$, where $\bar{w}$ and $\bar{a}$ have the same arity, we say $\bar{a}$ *satisfies* $\Theta(\bar{w})$ if $\mathfrak{N} \models \Theta[\bar{a}]$. We allow constants for all natural numbers in Presburger formulas, since these can be succinctly defined using 0, 1 and $+$. We also allow ourselves to use *relativized* quantification $\exists(\bar{v} \leq M)$, again defined in the obvious way.

3 ExpSpace-hardness

In this section, we show that $\mathrm{Sat}(\mathcal{FL}^2 + 1\mathrm{Tr})$ is ExpSpace-hard. This improves the already known NExpTime-lower bounds from GrK4 [17] and $\mathcal{FL}^2$ [29]. Note that $\mathcal{FL}^2 + 1\mathrm{Tr}$, the two-variable fluted fragment with one transitive relation, features neither equality nor counting quantifiers. The material also serves as an opportunity for familiarization with the variable-free syntax for fluted logic just introduced. By the well-known correspondence between deterministic and alternating complexity classes [6], it suffices to show a polynomial time reduction from the acceptance problem for an alternating Turing machine working in exponential time. The reduction can be effected using similar ideas as in, e.g. [20], where the complexity of satisfiability for the two-variable guarded fragment with one-way transitive guards was studied. However, special care needs to be taken to encode the required properties using only fluted formulas. In particular, it is essential for us to consider signatures featuring several binary predicates, in contrast to the non-fluted case [20], where ExpSpace-hardness is shown even when the distinguished transitive relation is the only binary predicate.

We employ a natural representation of integers as bit strings. Let $\bar{s} = s_{z-1}, \ldots, s_0$ be a bit-string representing an integer $n = \sum_{i=0}^{z-1} s_i \cdot 2^i$ (s_0 is the least significant bit). Within a structure $\mathfrak{A}$ interpreting unary predicates $p_0, \ldots, p_{z-1}$, each element b can be associated with the integer value between 0 and $2^z - 1$ represented by the bit-string $s_{z-1}, \ldots, s_0$, where, for all i ($0 \leq i < z$), $s_i = 1$ if $\mathfrak{A} \models p_i[b]$, and 0 otherwise.

Turning to the actual reduction, let M be an alternating Turing machine working in time 2^{n^k}, for some $k \geq 1$. Let Σ be the tape alphabet of M and Q the set of states; we write $Q = Q_\exists \cup Q_\forall$, where $Q_\exists$ ($Q_\forall$) is the set of existential (universal) states. Without loss of generality, we assume that for each combination of a state and alphabet symbol M has exactly two transitions, which we think of as leading to "left" and "right" successor-configurations. This notion refers only to the organization of the computation tree of M and not to the directions of the head defined by the transitions of M. We also assume that M never moves left from the initial tape cell and that it accepts or rejects in exactly the 2^{n^k}-th step. The idea of the reduction is to write a formula Φ such that each of its models encodes a binary tree whose nodes correspond to configurations of M on input w: the root of the tree encodes the initial configuration, and successors of a node encode successor configurations corresponding to transitions on M. An extra unary predicate lft is used to distinguish the "left" successor-configuration of a node. A configuration is encoded by 2^{n^k} elements, each of them corresponding to a single cell of the tape of M.

Fix some input w for M of length n, and, letting $z = n^k$, take two sets of unary predicates $p_0, \ldots, p_{z-1}$, $c_0, \ldots, c_{z-1}$ that will encode, for each element b in any structure $\mathfrak{A}$ interpreting them, two integer values $\mathrm{val}^{\mathfrak{A}}_P(b)$ and $\mathrm{val}^{\mathfrak{A}}_C(b)$ in the range $[0, 2^z - 1]$, as described above. Thinking of b as a tape cell in some node of the computation tree of M on input w, we are invited to read $\mathrm{val}_P(b)$ as the *position* of the tape cell in question and $\mathrm{val}_C(b)$ as the *time step* of the node in question (i.e. *depth* in the computation tree). For each $D \in \{C, P\}$, we add to the signature the unary predicates zero_D and max_D and the binary predicates pred_D, succ_D and eq_D. We then write a satisfiable formula Φ_{count} with the property that any model $\mathfrak{A} \models \Phi_{count}$ satisfies the following conditions:

(c1) $\mathfrak{A} \models \mathrm{zero}_D[b] \Leftrightarrow (\mathrm{val}^{\mathfrak{A}}_D(b) = 0)$, and $\mathfrak{A} \models \mathrm{max}_D[b] \Leftrightarrow (\mathrm{val}^{\mathfrak{A}}_D(b) = 2^z - 1)$,
(c2) $\mathfrak{A} \models \mathrm{eq}_D[b, b'] \Leftrightarrow \mathrm{val}^{\mathfrak{A}}_D(b) = \mathrm{val}^{\mathfrak{A}}_D(b')$,
(c3) $\mathfrak{A} \models \mathrm{pred}_D[b, b'] \Leftrightarrow \mathrm{val}^{\mathfrak{A}}_D(b') = \mathrm{val}^{\mathfrak{A}}_D(b) - 1$ modulo 2^z,
(c4) $\mathfrak{A} \models \mathrm{succ}_D[b, b'] \Leftrightarrow \mathrm{val}^{\mathfrak{A}}_D(b') = \mathrm{val}^{\mathfrak{A}}_D(b) + 1$ modulo 2^z.

Thus, we may informally read $\mathrm{zero}_C(x)$ as "x corresponds to some tape cell in the *root* node (i.e. time=0); $\mathrm{succ}_P(x, y)$ as "x corresponds to a tape cell in some node, and y to the tape cell immediately to its *right* (in some possibly different node); and so on. The construction of Φ_{count} is routine; for details, see [29, pp. 1026-27]).

The predicate $\mathfrak{t}$ is used to encode the structure of the computation tree. Intuitively, we take $\mathfrak{t}[b, b']$ to mean that b and b' are tape cells in the same node with b lying to the left of b', or that b and b' are tape cells in different nodes, with the latter being a proper descendant of the former in the computation tree. We denote by Φ_{tree} the conjunction $\Phi_1 \wedge \Phi_2 \wedge \Phi_3$ establishing that any model of Φ_{tree} includes a tree substructure, where

$$\Phi_1 \equiv \exists(\mathrm{zero}_C \wedge \mathrm{zero}_P \wedge \mathrm{lft})$$
$$\Phi_2 \equiv \forall\big(\neg\mathrm{max}_P \to (\mathrm{lft} \to \exists(\mathfrak{t} \wedge \mathrm{eq}_C \wedge \mathrm{succ}_P \wedge \mathrm{lft})\big) \wedge \big(\neg\mathrm{lft} \to \exists(\mathfrak{t} \wedge \mathrm{eq}_C \wedge \mathrm{succ}_P \wedge \neg\mathrm{lft})\big)$$
$$\Phi_3 \equiv \forall\big(\mathrm{max}_P \wedge \neg\mathrm{max}_C \to \exists(\mathfrak{t} \wedge \mathrm{lft} \wedge \mathrm{zero}_P \wedge \mathrm{succ}_C) \wedge \exists(\mathfrak{t} \wedge \neg\mathrm{lft} \wedge \mathrm{zero}_P \wedge \mathrm{succ}_C)\big).$$

To help the reader further with the variable-free notation we give the formula Φ_3 in standard first-order syntax:

$$\forall x_1\big(\mathrm{max}_P(x_1) \wedge \neg\mathrm{max}_C(x_1) \to \exists x_2(\mathfrak{t}(x_1, x_2) \wedge \mathrm{lft}(x_2) \wedge \mathrm{zero}_P(x_2) \wedge \mathrm{succ}_C(x_1, x_2)) \wedge \exists x_2(\mathfrak{t}(x_1, x_2) \wedge \neg\mathrm{lft}(x_2) \wedge \mathrm{zero}_P(x_2) \wedge \mathrm{succ}_C(x_1, x_2))\big).$$

The conjunct Φ_1 specifies that each model contains an element belonging to the root of the tree, for which both counters are set to 0. This element is also marked as satisfying lft and it will correspond to the first tape cell of the initial configuration of M on w. The

conjunct Φ_2 provides remaining elements belonging to the same node. They will correspond to successive tape cells within the configuration encoded in this node: these elements form a chain connected by $\mathfrak{t}$ with the P-counter increasing from 0 to 2^z-1, and the C-counter stable. The predicate lft is uniformly true or false for all elements in such a chain, depending on whether the first element (with P-counter 0) is marked lft. The conjunct Φ_3 provides elements that belong to successors of a given node in the tree; they will correspond to the first tape cell in the configurations encoded by the successor nodes. Specifically, each element with $\mathrm{val}_P = 2^z-1$ and $\mathrm{val}_C < 2^z-1$ has two witnesses connected by $\mathfrak{t}$, each with the C-counter incremented by 1 and the P-counter set to 0: one satisfying lft, the other one $\neg$lft. Note that by transitivity of $\mathfrak{t}$ each element belonging to a given node is connected by $\mathfrak{t}$ to all elements in descendant nodes.

To encode a configuration of M, we employ further unary predicates: h, $a_0,\dots,a_s$, $q_0,\dots,q_r$. Here, h indicates the position of the head of M, $a_0,\dots,a_s$ correspond to the symbols of the alphabet of M, and $q_0,\dots,q_r$ correspond to the states of M. We assume that a_0 represents the blank symbol, a_s a special start-of-tape symbol, q_0 the initial state and q_r the only rejecting state. Note that we use the same letters for the alphabet symbols of M and the predicates representing them in the signature of Φ; similarly for states.

Let Φ_M be a conjunction of formulas describing the following basic properties of an accepting computation tree of M:

(m1) every tape cell contains exactly one symbol,
(m2) in each configuration the head is scanning at most one cell: $\forall(h \to \forall(\mathfrak{t} \wedge \mathrm{eq}_C \to \neg h))$,
(m3) the information about the current state is stored in elements scanned by the head,
(m4) the root of the computation tree contains the initial configuration of M on input w,
(m5) when moving from a configuration to its successor configurations only tape cells scanned by the head are allowed to change: $\bigwedge_{0\le i\le s}\forall(a_i \wedge \neg h \to \forall(\mathfrak{t} \wedge \mathrm{eq}_P \wedge \mathrm{succ}_C \to a_i))$.

Properties (m1), (m3) and (m4) are one-variable formulas, so they clearly belong to $\mathcal{FL}^2$.

To encode the transition function of M we consider separately transitions from existential and from universal states. Suppose M has the following possible transitions for an existential state q and letter a: $\delta(q,a) = \{(q',a',R),(q'',a'',L)\}$. We want to ensure that the left child of nodes corresponding to configurations of M in state q reading a encodes one of the possible next configurations. This can be formalized by the formula $\Phi^{\exists}_{q,a}$ below:

$$\Phi^{\exists}_{q,a} \equiv \forall\Big(q \wedge a \wedge h \to \big(\forall(\mathfrak{t} \wedge \mathrm{lft} \wedge \mathrm{succ}_C \to (\mathrm{eq}_P \to a') \wedge (\mathrm{succ}_P \to q' \wedge h)) \quad \vee \\ \forall(\mathfrak{t} \wedge \mathrm{lft} \wedge \mathrm{succ}_C \to (\mathrm{pred}_P \to q'' \wedge h) \wedge (\mathrm{eq}_P \to a''))\big)\Big).$$

When encoding the computation at universal states we require that one transition is encoded by the left child and the other one by the right child. In particular, when M has the transitions $\delta(q,a) = \{(q',a',R),(q'',a'',L)\}$ and $q \in Q_\forall$, we define the formula $\Phi^{\forall}_{q,a}$ as follows:

$$\Phi^{\forall}_{q,a} \equiv \forall\Big(q \wedge a \wedge h \to \forall\big(\mathfrak{t} \wedge \mathrm{lft} \wedge \mathrm{succ}_C \to (\mathrm{eq}_P \to a') \wedge (\mathrm{succ}_P \to q' \wedge h)\big) \quad \wedge \\ \forall\big(\mathfrak{t} \wedge \neg\mathrm{lft} \wedge \mathrm{succ}_C \to (\mathrm{pred}_P \to q'' \wedge h) \wedge (\mathrm{eq}_P \to a'')\big)\Big).$$

Finally, let $\Phi \equiv \Phi_{count} \wedge \Phi_{tree} \wedge \Phi_M \wedge \bigwedge_{a\in\Sigma}(\bigwedge_{q\in Q_\exists}\Phi^{\exists}_{q,a} \wedge \bigwedge_{q\in Q_\forall}\Phi^{\forall}_{q,a}) \wedge \forall\neg q_r$. The role of the last conjunct of Φ is to ensure that M never enters a rejecting state. The length of Φ is polynomial in the length of w, regarding M as fixed.

It is not difficult to show that Φ is satisfiable if and only if M accepts w. For the only-if direction, suppose that there is an accepting computation tree of M with input w; we construct a model $\mathfrak{A}$ of Φ in the form of a tree. The initial configuration of M is transformed

into the root of $\mathfrak{A}$ as suggested by part (m4) of Φ_M. Then we proceed recursively. When a configuration $\mathcal{C}$ in the computation tree of M corresponding to a node b in $\mathfrak{A}$ is universal, we transform the left subtree of $\mathcal{C}$ into the left subtree of b in $\mathfrak{A}$, and the right subtree of $\mathcal{C}$ into the right subtree of b. If $\mathcal{C}$ is existential we transform the accepting subtree of $\mathcal{C}$ into both the left and the right subtree of b. Since M is accepting, the formula Φ is true in the model.

For the opposite direction, suppose $\mathfrak{A} \models \Phi$. We construct an accepting computation tree for M on input w, starting with the initial configuration, and then proceed recursively. Suppose we have constructed a configuration $\mathcal{C}$ of M on w of depth $d < 2^z - 1$. The formula Φ_{tree} ensures that $\mathfrak{A}$ contains two chains of elements $\bar{b} = b_0, \ldots, b_{2^z-1}$ and $\bar{b'} = b'_0, \ldots, b'_{2^z-1}$ (i.e. for all i $(0 \leq i < 2^z - 1)$ $\mathfrak{A} \models \mathfrak{t}[b_i, b_{i+1}] \wedge \mathfrak{t}[b'_i, b'_{i+1}]$) connected to the elements representing $\mathcal{C}$ by $\mathfrak{t}$ such that for all i $(0 \leq i \leq 2^z - 1)$ we have: $\mathfrak{A} \models \mathrm{val}_P[b_i] = \mathrm{val}_P[b'_i] = i$, $\mathfrak{A} \models \mathrm{val}_C[b_i] = \mathrm{val}_C[b'_i] = d + 1$, $\mathfrak{A} \models \mathrm{lft}[b_i] \wedge \neg\mathrm{lft}[b'_i]$. If $\mathcal{C}$ is existential we translate the information encoded by the chain $\bar{b}$ of elements $b_0, \ldots, b_{2^z-1}$ into a successor configuration of $\mathcal{C}$. If $\mathcal{C}$ is universal we translate the information encoded by the chain $\bar{b}$ into the left successor of $\mathcal{C}$, and the information encoded by the chain $\bar{b'}$ into the right successor of $\mathcal{C}$. The conjuncts $\Phi^{\exists}_{q,a}$ and $\Phi^{\forall}_{q,a}$ ensure that the tree constructed in such a way is a computation tree of M, and by the conjunct $\forall\neg q_r$ of Φ, the tree is accepting.

Hence, we have

▶ **Theorem 2.** $\mathrm{Sat}(\mathcal{FL}^2 + 1\mathrm{Tr})$ *is* ExpSpace*-hard.*

4 Presburger quantifiers

In Sec. 3, we obtained a lower complexity bound on $\mathrm{Sat}(\mathcal{FLC}^2{+}1\mathrm{Tr})$ by investigating the corresponding problem for the *sub*-fragment of that logic without counting quantifiers. In Secs. 5–6, we will obtain upper complexity bounds on $\mathrm{Sat}(\mathcal{FLC}^k{+}1\mathrm{Tr})$ $(k \geq 2)$ by investigating the corresponding problem for a *super*-fragment of that logic in which the notion of counting quantifier has been generalized. These generalized counting quantifiers were introduced in [27] in the context of $\mathcal{FLC}$ (but see [3] for a closely related idea); this section provides a more streamlined presentation.

Fix $k \geq 1$. A *fluted k-atom* over Σ is a predicate $p \in \Sigma$ of arity $d \leq k$. A *fluted k-literal* over Σ is a fluted k-atom over Σ or its negation. A *fluted k-type* over Σ is a maximal consistent set of fluted literals over Σ. It is worth explaining how variable-free notation is to be understood here: we think of a literal $\pm p$ in a fluted k-type as "really" being the formula $\pm p(x_{k-d+1}, \ldots, x_k)$, so that variables are aligned in such a way that all literals notionally have the same last variable. We denote the set of fluted k-types over Σ by Ftp^{Σ}_k. If $M \geq 0$, an *M-bounded, fluted k-profile* over Σ is a function $\zeta : \mathrm{Ftp}^{\Sigma}_{k+1} \to \{0, \ldots, M\}$. An *$M$-bounded, fluted k-star-type* over Σ is a pair $\langle \pi, \zeta \rangle$, where π is a fluted k-type over Σ and ζ an M-bounded, fluted k-profile over Σ. In all cases, reference to Σ is suppressed when clear from context. (The intuition behind these definitions will be explained presently.) We use π to range over fluted 1-types, ρ, σ, τ to range over fluted k-types (for various k), and ζ, η to range over M-bounded, fluted k-profiles (for various M and k). To reduce notational clutter, and assuming that Σ is finite, we silently identify fluted k-types with their conjunctions where convenient, thus allowing ourselves to write, for instance, τ for the quantifier-free $\mathcal{FLC}^k$-formula $\bigwedge \tau$.

Let $\mathfrak{A}$ be a structure interpreting a finite signature Σ and $\bar{a}$ a k-tuple of elements of A. The set of fluted k-literals over Σ satisfied by $\bar{a}$ in $\mathfrak{A}$ is evidently a fluted k-type, τ. We say that $\bar{a}$ *realizes* τ, and refer to it as *the fluted k-type of $\bar{a}$ in* $\mathfrak{A}$, denoted $\mathrm{ftp}^{\mathfrak{A}}[\bar{a}]$. Intuitively, a fluted k-type is to be thought of as a specification, for some k-tuple of elements, of which

fluted literals are satisfied by that k-tuple. Now consider the $(k{+}1)$-tuples $\bar{a}b$ extending $\bar{a}$ by a single element b. Each such $(k{+}1)$-tuple has some fluted $(k{+}1)$-type in $\mathrm{Ftp}^{\Sigma}_{k+1}$, and we may *count*, up to some bound M, the number of times each $(k{+}1)$-type in $\mathrm{Ftp}^{\Sigma}_{k+1}$ arises in this way. More precisely, the *M-bounded, fluted profile of* $\bar{a}$ in $\mathfrak{A}$ is the function $\mathrm{fpr}^{\mathfrak{A}}_{M}[\bar{a}] : \mathrm{Ftp}^{\Sigma}_{k+1} \to \{0, \ldots, M\}$ defined by

$$\mathrm{fpr}^{\mathfrak{A}}_{M}[\bar{a}](\tau) = \min(|\{b \in A : \mathrm{ftp}^{\mathfrak{A}}[\bar{a}b] = \tau\}|, M).$$

Clearly, $\mathrm{fpr}^{\mathfrak{A}}_{M}[\bar{a}]$ is an M-bounded, fluted k-profile, as defined in the preceding paragraph. Intuitively, an M-bounded, fluted k-profile is to be thought of as a specification, for some k-tuple of elements, of how many different $(k{+}1)$-tuples it can be extended to (up to a ceiling of M) satisfying each of the possible fluted $(k{+}1)$-types. Finally, we say that the *M-bounded, fluted star-type of* $\bar{a}$ in $\mathfrak{A}$ is the pair $\mathrm{fst}^{\mathfrak{A}}_{M}[\bar{a}] = \langle \mathrm{ftp}^{\mathfrak{A}}[\bar{a}], \mathrm{fpr}^{\mathfrak{A}}_{M}[\bar{a}] \rangle$. Intuitively, an M-bounded, fluted k-star-type is simply a combination of a fluted k-type and an M-bounded, fluted k-profile.

We now turn to the promised generalization of counting quantifiers. By way of motivation, let β be a quantifier-free $\mathcal{FLC}^{k+1}$-formula, and consider the $\mathcal{FLC}^{k}$-formula $\exists_{\leq M}\beta$. Applied to any k-tuple of elements $\bar{a}$, this latter formula makes a statement about the fluted $(k{+}1)$-types satisfied by the various $(k{+}1)$-tuples $\bar{a}b$ as b ranges over the domain of quantification, namely, that for at most M such elements b, the fluted type of $\bar{a}b$ entails β. This formulation invites generalization. For each fluted $(k{+}1)$-type τ over the relevant signature, let v_τ be the number of elements b such that $\bar{a}b$ has fluted type τ, up to some fixed ceiling M (after which we stop counting). Then we can impose any set of conditions on the collection of integers v_τ (in the range $[0, M]$) thus obtained. Accordingly, we say that a *computable counting quantifier* is an expression $Q(k, \Sigma, M, \Theta)$, where $k \geq 1$, Σ is a signature, $M \geq 0$ and Θ a set of M-bounded fluted k-profiles over Σ, presented in some way which allows membership to be effectively determined. (For example, Θ could be presented as a Turing machine that terminates on all inputs.) We regard computable counting quantifiers as $\mathcal{FLC}^{k}$-formulas in their own right. If $\mathfrak{A}$ is any structure interpreting a signature including Σ, and $\bar{a}$ is a k-tuple of elements from A, then we define

$$\mathfrak{A} \models Q(k, \Sigma, M, \Theta)[\bar{a}] \quad \text{iff} \quad \mathrm{fpr}^{\mathfrak{A}}_{M}[\bar{a}] \in \Theta.$$

In particular, if β is a quantifier-free $\mathcal{FLC}^{k+1}$-formula, then the $\mathcal{FLC}^{k}$-formula $\exists_{\leq M}\beta$ can be equivalently written as $Q(k, \Sigma, M+1, \Theta)$, where Θ is the set of $(M+1)$-bounded, fluted k-profiles ζ satisfying the condition

$$\sum\{\zeta(\tau) : \tau \in \mathrm{Ftp}^{\Sigma}_{k+1},\ \models \tau \to \beta\} \leq M.$$

Thus, computable counting quantifiers generalize ordinary counting quantifiers applied to quantifier-free formulas. Conversely, computable quantifiers can be expressed in terms of counting quantifiers, using the fact that all numbers concerned are subject to the finite bound M. Thus, $Q(k, \Sigma, M, \Theta)$ can be written as the (huge) $\mathcal{FLC}^{k+1}$-formula

$$\bigvee_{\zeta \in \Theta} \Big(\bigwedge \{ \exists_{[=\zeta(\tau)]}\, \tau \mid \tau \in \mathrm{Ftp}^{\Sigma}_{k+1} \text{ s.t. } \zeta(\tau) < M \} \wedge \bigwedge \{ \exists_{[\geq M]}\, \tau \mid \tau \in \mathrm{Ftp}^{\Sigma}_{k+1} \text{ s.t. } \zeta(\tau) = M \} \Big).$$

Hence there is no difference in expressive power between ordinary counting quantifiers and computable counting quantifiers. However, there may be a complexity-theoretic difference: the condition Θ may be presented in a more compact way than ordinary counting quantifiers make possible. One such way will be of particular significance in this paper. Let $\bar{v}$ be a

collection of variables v_τ (in some fixed order) as τ ranges over $\mathrm{Ftp}^{\Sigma}_{k+1}$, and let $\Theta(\bar{v})$ be a formula of Presburger arithmetic in the variables $\bar{v}$. Now, any tuple of numbers chosen from $\{0, \ldots, M\}$ satisfying Θ corresponds to a function mapping each fluted $(k{+}1)$-type τ to the number assigned to the variable v_τ. Thus, Θ specifies a set of M-bounded, fluted k-profiles in a natural way. A computable counting quantifier in which Θ is presented as a formula of Presburger arithmetic in this way will be called a *Presburger quantifier*. If Θ is an existential Presburger formula, i.e. a formula of the form $\exists \bar{w}\, \Xi(\bar{v}, \bar{w})$, then we speak of an *existential Presburger quantifier*.

These considerations lead us to define a sequence of languages based on existential Presburger quantifiers. For any $k \geq 2$, let $\mathcal{FLU}^k$ be the collection of formulas of the form

$$\bigwedge_{h=1}^{m} \forall^{k-1}(\alpha_h \to Q(k-1, \Sigma, M, \Theta_h)) \tag{4}$$

where Σ is a purely relational signature, m a positive integer, the α_h quantifier-free $\mathcal{FLC}^{k-1}$-formulas over Σ, M a non-negative integer and the Θ_h existential Presburger formulas with free variables $\bar{v}$ indexed by the fluted k-types over Σ. The signature of such a formula is taken to be Σ. We need a careful parametrization of $\mathcal{FLU}^k{+}1\mathrm{Tr}$-formulas in the sequel. If ψ is of the form (4), define the *effective size of* ψ, denoted $\#(\psi)$, to be the quantity $|\Sigma| + \log M + m$. We define the fragment $\mathcal{FLU}^k{+}1\mathrm{Tr}$ to consist of the set of formulas of the form (4), over a signature featuring $\mathfrak{t}$, again required to be interpreted as a transitive relation.

It is simple to re-write any normal-form $\mathcal{FLC}^k$-formula φ of the form (3) over a signature Σ as a logically equivalent formula ψ of the form (4), where $M = \max(M_1, \ldots M_m) + 1$. Moreover, ψ can be computed in exponential time. (Exponential time is required, because the variables in the embedded Presburger formulas are indexed by fluted 1-types, of which there are exponentially many.) Note however that $\#(\psi) \leq \|\varphi\|$. Thus, using Lemma 1, we see that any decision procedure for the problem $\mathrm{Sat}(\mathcal{FLU}^k)$ yields a decision procedure for the problem $\mathrm{Sat}(\mathcal{FLC}^k)$. To obtain such a procedure, we show how to reduce $\mathrm{Sat}(\mathcal{FLU}^{k+1}{+}1\mathrm{Tr})$ to $\mathrm{Sat}(\mathcal{FLU}^k{+}1\mathrm{Tr})$ (but with exponential blow-up), where $k \geq 2$. The principal difficulty is to then establish the base case, namely, $\mathrm{Sat}(\mathcal{FLU}^2{+}1\mathrm{Tr})$.

We approach our task in two stages. We consider first a sub-fragment of $\mathcal{FLU}^2{+}1\mathrm{Tr}$ obtained by restricting attention to signatures featuring no predicates of arity higher than 1 except for the distinguished predicate $\mathfrak{t}$. Sec. 5 is devoted to a small model property for this sub-fragment. Then, in Sec. 6, we first lift this result to the whole of $\mathcal{FLU}^2{+}1\mathrm{Tr}$, and then generalize to $\mathrm{Sat}(\mathcal{FLU}^k{+}1\mathrm{Tr})$ for all $k \geq 2$. We remark that the same basic strategy was employed in [27] to decide $\mathrm{Sat}(\mathcal{FLC})$. There, however, one has the luxury of an easy base case: the logic $\mathcal{FLC}^2$ is contained in $\mathcal{C}^2$, the two-variable fragment of first-order logic with counting, for which decidability of satisfiability is known. Indeed, the same argument shows the decidability of $\mathrm{Sat}(\mathcal{FLC})$ even in the presence of a single distinguished binary predicate required to be interpreted as an *equivalence relation*, since the corresponding extension of $\mathcal{C}^2$ is known to have a decidable satisfiability problem [26]. Unfortunately, adding a single *transitive* relation to $\mathcal{C}^2$ yields a logic with undecidable satisfiability problem. Thus, the main work of this paper is to establish the decidability of $\mathrm{Sat}(\mathcal{FLU}^2{+}1\mathrm{Tr})$ from scratch.

Since satisfaction of a Presburger quantifier by a tuple $\bar{a}$ is determined by the profile of $\bar{a}$, the following observation is immediate:

▶ **Lemma 3.** *Let φ be a formula of $\mathcal{FLU}^k$ (or of $\mathcal{FLU}^k{+}1\mathrm{Tr}$) of the form (4), with $k \geq 2$, over a signature Σ. Let $\mathfrak{A}$, $\mathfrak{B}$ be structures interpreting Σ such that every M-bounded, fluted $(k-1)$-star-type realized in $\mathfrak{B}$ is realized in $\mathfrak{A}$. Then $\mathfrak{A} \models \varphi$ implies $\mathfrak{B} \models \varphi$.*

5 Two variables, unary predicates

Define $\mathcal{FLU}^2_u$+1Tr to be the sub-fragment of $\mathcal{FLU}^2$+1Tr in which only unary predicates or the distinguished binary predicate $\mathfrak{t}$ occur. In this section we show that $\mathcal{FLU}^2_u$+1Tr has the finite model property, and that Sat($\mathcal{FLU}^2_u$+1Tr) is in 2-NExpTime.

We briefly outline the technique employed. Because $\mathcal{FLU}^2_u$+1Tr is not guarded, we cannot easily build tree-like models of satisfiable formulas as with modal or description logics. Because $\mathcal{FLU}^2_u$+1Tr features a transitive relation, we cannot easily employ equational techniques as with the two-variable fragment with counting, $\mathcal{C}^2$. Instead, we show how, from an arbitrary model $\mathfrak{A}$ of some $\mathcal{FLU}^2$+1Tr-formula φ, we can extract a model $\mathfrak{B}$ of cardinality bounded by some doubly exponential function in the size of φ. *A priori*, the directed graph $(A, \mathfrak{t}^{\mathfrak{A}})$ may contain infinite or even dense paths, so that it is hard to know where to start removing vertices. Our solution, paradoxically, is to *add* vertices. By a judicious choice of such additions, we are subsequently able to remove edges in such a way that no long paths remain. It is then straightforward to obtain a model subject to the desired size bound.

Fix some signature Σ featuring only unary predicates and the distinguished binary predicate $\mathfrak{t}$. Observe that the fluted 2-types over Σ are simply the fluted 1-types over Σ together with either of the fluted literals $\mathfrak{t}$ or $\neg\mathfrak{t}$. For the remainder of this section, we generally suppress reference to Σ. We find it helpful to think of an M-bounded fluted 1-profile ζ as a pair of functions (ζ^+, ζ^-), each mapping Ftp^{Σ}_1 to $[0, M]$, given by:

$$\zeta^+(\pi) = \zeta(\pi \wedge \mathfrak{t}); \qquad \zeta^-(\pi) = \zeta(\pi \wedge \neg\mathfrak{t}).$$

Thus, the components ζ^+ and ζ^- simply separate out the "positive" and "negative" 2-types, from the point of view of the fluted atom $\mathfrak{t}$. If $\mathfrak{A}$ is a structure interpreting $\mathfrak{t}$, we call any maximal subset Q of A having the property that $\mathfrak{A} \models \mathfrak{t}[a, b]$ for all distinct $a, b \in Q$, a *clique*. Thus, A is partitioned into cliques. If Q is a clique with $|Q| > 1$, then, by transitivity, $\mathfrak{A} \models \mathfrak{t}[a, a]$ for all $a \in Q$. If $Q = \{a\}$, then it may or may not be the case that $\mathfrak{A} \models \mathfrak{t}[a, a]$. If Q and Q' are cliques and c an element, we write $\mathfrak{A} \models \mathfrak{t}[Q, c]$ to mean $\mathfrak{A} \models \mathfrak{t}[a, c]$ for some (equivalently, any) $a \in Q$; similarly for $\mathfrak{A} \models \mathfrak{t}[c, Q]$ and $\mathfrak{A} \models \mathfrak{t}[Q, Q']$.

Let $\zeta = (\zeta^+, \zeta^-)$ and $\eta = (\eta^+, \eta^-)$ be M-bounded fluted 1-profiles. We write $\zeta \preceq \eta$ if, for all $\pi \in \mathrm{Ftp}^{\Sigma}_1$, $\zeta^+(\pi) \geq \eta^+(\pi)$ and $\zeta^-(\pi) \leq \eta^-(\pi)$. Evidently, if $\mathfrak{A}$ is a structure and $\mathfrak{A} \models \mathfrak{t}[a, b]$, then $\mathrm{fpr}^{\mathfrak{A}}[a] \preceq \mathrm{fpr}^{\mathfrak{A}}[b]$. If ζ is an M-bounded, fluted 1-profile realized in $\mathfrak{A}$, we call the set of elements $B = \{b \in A : \mathrm{fpr}^{\mathfrak{A}}[b] = \zeta\}$ a *profile set*. A *cluster* is a weakly connected component of the directed graph $(B, \mathfrak{t}^{\mathfrak{A}} \cap B^2)$, where B is a profile set – that is, a maximal subset $C \subseteq B$ such that, for all distinct $a, b \in C$, there exists a sequence of elements $a = a_1, \ldots, a_s = b$ such that, for all i $(1 \leq i < s)$, either $\mathfrak{A} \models \mathfrak{t}[a_i, a_{i+1}]$ or $\mathfrak{A} \models \mathfrak{t}[a_{i+1}, a_i]$. Obviously, a cluster is a union of cliques. Observe that, for any cluster C, there is no triple of elements c, d, e with $c, e \in C$, $d \notin C$ such that $\mathfrak{A} \models \mathfrak{t}[c, d]$ and $\mathfrak{A} \models \mathfrak{t}[d, e]$. Indeed, otherwise, we have $\mathrm{fpr}^{\mathfrak{A}}[c] \preceq \mathrm{fpr}^{\mathfrak{A}}[d] \preceq \mathrm{fpr}^{\mathfrak{A}}[e] = \mathrm{fpr}^{\mathfrak{A}}[c]$ contradicting the supposition that $d \notin C$.

▶ **Lemma 4.** *Let C be a cluster in a structure $\mathfrak{A}$ whose elements have M-bounded, fluted 1-profile ζ, let $\pi \in \mathrm{Ftp}^{\Sigma}_1$ and suppose that either $\zeta^+(\pi) < M$ or $\zeta^-(\pi) < M$. Then any two elements of C are related by $\mathfrak{t}$ in $\mathfrak{A}$ to the same elements of A having fluted 1-type π.*

Proof. Since C is a weakly connected component of the directed graph $(B, \mathfrak{t}^{\mathfrak{A}} \cap B^2)$, where B is a profile set, it suffices to show that any two elements $c, d \in C$ such that $\mathfrak{A} \models \mathfrak{t}[c, d]$ are related by $\mathfrak{t}$ in $\mathfrak{A}$ to the same elements of A having fluted 1-type π. By transitivity, we have $\{b \in A : \mathfrak{A} \models \mathfrak{t}[c, b] \text{ and } \mathrm{ftp}^{\mathfrak{A}}[b] = \pi\} \supseteq \{b \in A : \mathfrak{A} \models \mathfrak{t}[d, b] \text{ and } \mathrm{ftp}^{\mathfrak{A}}[b] = \pi\}$, and similarly $\{b \in A : \mathfrak{A} \not\models \mathfrak{t}[c, b] \text{ and } \mathrm{ftp}^{\mathfrak{A}}[b] = \pi\} \subseteq \{b \in A : \mathfrak{A} \not\models \mathfrak{t}[d, b] \text{ and } \mathrm{ftp}^{\mathfrak{A}}[b] = \pi\}$. If $\zeta^+(\pi) < M$,

then the former two sets have equal finite cardinality, and hence are equal; if $\zeta^-(\pi) < M$, then the latter two sets are. Either way, c and d are related by $\mathfrak{t}$ in $\mathfrak{A}$ to the same elements of A having fluted 1-type π. ◀

A clique Q in a cluster C in a structure $\mathfrak{A}$ will be called *the superior clique* of C if $\mathfrak{A} \models \mathfrak{t}[a, Q]$ for every $a \in C$. It is possible for there to be no superior clique in C, but if it exists, it is unique.

▶ **Lemma 5.** *Let C be a cluster in a structure $\mathfrak{A}$ whose elements have M-bounded, fluted 1-profile ζ. Let $\pi \in \mathrm{Ftp}_1^\Sigma$. If either $\zeta^+(\pi) < M$ or $\zeta^-(\pi) < M$ and there exists $b \in C$ such that $\mathrm{ftp}^\mathfrak{A}[b] = \pi$ and $a \in C$ such that $\mathfrak{A} \models \mathfrak{t}[a, b]$, then the superior clique of C exists and contains every element $c \in C$ such that $\mathrm{ftp}^\mathfrak{A}[c] = \pi$ and $\mathfrak{A} \models \mathfrak{t}[a, c]$ for some $a \in C$.*

Proof. By Lemma 4, if such a and b exist, then every element of C is related to b by $\mathfrak{t}^\mathfrak{A}$, whence b is in the superior clique. But there can only be one superior clique. ◀

An important notion in the ensuing argument is that of a *stable* clique. If $\mathfrak{A}$ is a structure interpreting Σ, π a fluted 1-type and Q a clique of $\mathfrak{A}$ included in a cluster C with M-bounded, fluted 1-profile ζ, we say that π is *sensitive* for Q if

$$|\{b \in (A \setminus C) \cup Q : \mathfrak{A} \models \mathfrak{t}[Q, b] \text{ and } \mathrm{ftp}^\mathfrak{A}[b] = \pi\}| < \zeta^+(\pi).$$

Intuitively, π is sensitive for Q if the elements of Q need to be related by $\mathfrak{t}$ to elements having fluted 1-type π belonging to *other* cliques of C in order to make up the quota of witnesses demanded by ζ^+ – in other words, if there are not enough witnesses either outside C or inside Q. We call a clique Q *stable* if, for every fluted 1-type π sensitive for Q and every element $a \in A \setminus C$ such that $\mathfrak{A} \not\models \mathfrak{t}[a, Q]$, $|\{c \in C : \mathfrak{A} \not\models \mathfrak{t}[a, c] \text{ and } \mathrm{ftp}^\mathfrak{A}[c] = \pi\}| \geq M$. That is, Q is stable if, for every fluted 1-type π that is sensitive for Q, every element not related by $\mathfrak{t}$ to Q is not related to at least M elements of C satisfying π. A special case of stability is where Q has no sensitive fluted 1-types; in that case, we say that Q is *trivially stable*.

▶ **Lemma 6.** *If $\mathfrak{A}$ is a structure, and Q a clique of $\mathfrak{A}$ included in a cluster C, then either Q is stable, or there exists a stable clique $Q' \subseteq C$ such that $\mathfrak{A} \models \mathfrak{t}[Q, Q']$.*

Proof. Given the clique Q and cluster $C \supseteq Q$, define the procedure `stabilize` as shown in Fig. 1, in which the auxiliary variable f stores a function mapping the set of fluted 1-types Ftp_1^Σ to $[0, M]$. The goal of `stabilize` is to find the clique Q' guaranteed by the lemma. The idea is to examine the list Π of sensitive fluted 1-types for the clique currently under consideration (initially Q). By selecting a $\pi \in \Pi$ which has been encountered least often, we move along a $\mathfrak{t}$-edge to another clique of C in which π is realized. Before any execution of line 6, Π will have been assigned the set of fluted 1-types sensitive for P; hence $\pi \in \Pi$ implies that there exists $b \in (C \setminus P)$ such that $\mathfrak{A} \models \mathfrak{t}[P, b]$ and $\mathrm{ftp}^\mathfrak{A}[b] = \pi$, whence the instruction in line 7 can be executed. Furthermore, any run of `stabilize` terminates, since one of the values $f(\pi)$ less than $M + 1$ is incremented by every execution of line 8. Let Q' be the clique returned as the value of P in line 11. We claim that Q' is stable. Indeed, suppose π is sensitive for Q'. Then there is a clique path $Q_1, \ldots Q_M, Q'$ with Q_i containing an element of type π, say c_i, for all i $(1 \leq i \leq M)$. Hence, if $a \in A \setminus C$ with $\mathfrak{A} \not\models \mathfrak{t}[a, Q']$, then $\mathfrak{A} \not\models \mathfrak{t}[a, c_i]$ for all i $(1 \leq i \leq M)$, as claimed. ◀

We can now prove the key lemma concerning $\mathcal{FLU}_u^2$+1Tr. We say that a *clique path* in a structure $\mathfrak{A}$ is a sequence of distinct cliques $Q_1, \ldots, Q_s$ $(s \geq 1)$ such that $\mathfrak{A} \models \mathfrak{t}[Q_i, Q_{i+1}]$ for all i $(1 \leq i < s)$; the *length* of the clique path is $s - 1$. The *depth* of $\mathfrak{A}$ is the maximal length of any clique path plus 1 (∞ if there is no maximum).

```
1.  begin stabilize
2.      for all fluted 1-types π do f(π) ← 0
3.      P ← Q
4.      Π ← the set of fluted 1-types sensitive for P
5.      until f(π) = M + 1 for all π ∈ Π
6.          choose π ∈ Π for which f(π) is smallest
7.          choose a clique P′ ⊆ C containing some element of 1-type π
                with P′ ≠ P and 𝔄 ⊨ t[P, P′]
8.          f(π) ← f(π) + 1
9.          P ← P′
10.         let Π be the set of fluted 1-types sensitive for P
11.     return P
12. end stabilize
```

Figure 1 Procedure `stabilize` of Lemma 6.

▶ **Lemma 7.** *Let φ be a formula of $\mathcal{FLU}^2_u$+1Tr of the form (4) with $k = 2$, featuring the constant M. Let Σ be the signature of φ. If φ is satisfiable, then φ has a model of depth $(M+1)\cdot 2^{|\Sigma|+1}$.*

Proof. Suppose $\mathfrak{A} \models \varphi$. By the downward Löwenheim-Skolem theorem, we may assume that $\mathfrak{A}$ is finite or countably infinite.

Stage 1. The goal of this stage is to flatten any clusters having superior cliques. We modify $\mathfrak{A}$, proceeding cluster by cluster. Number the clusters of $\mathfrak{A}$ as $C_0, C_1, \ldots$, and let $\mathfrak{A}_0 = \mathfrak{A}$. Supposing $\mathfrak{A}_i$ to have been defined, we define $\mathfrak{A}_{i+1}$, over the same domain, A. If C_i lacks a superior clique, we do nothing, and set $\mathfrak{A}_{i+1} = \mathfrak{A}_i$. Otherwise, let Q be the superior clique of C_i; thus, $\mathfrak{A} \models \mathfrak{t}[b, Q]$ for all $b \in C_i$. We define $\mathfrak{A}_{i+1}$ to be the same as $\mathfrak{A}_i$, except that we take $\mathfrak{t}^{\mathfrak{A}_{i+1}}$ on Q to be $C_i \times Q$: i.e. $\mathfrak{A}_{i+1} \models \mathfrak{t}[c, e]$ if and only if either (i) $\mathfrak{A}_i \models \mathfrak{t}[c, e]$ and $\{c, e\} \not\subseteq C_i$, or (ii) $c \in C_i$ and $e \in Q$. Thus, $\mathfrak{t}^{\mathfrak{A}_{i+1}} \subseteq \mathfrak{t}^{\mathfrak{A}_i}$, and the two relations agree on pairs of elements which do not both lie in C_i. That $\mathfrak{t}^{\mathfrak{A}_{i+1}}$ is transitive follows from the fact that there is no triple of elements c, d, e with $c, e \in C_i$, $d \notin C_i$ such that $\mathfrak{A}_i \models \mathfrak{t}[c, d]$ and $\mathfrak{A}_i \models \mathfrak{t}[d, e]$.

We claim that the M-bounded, fluted 1-profiles are preserved in the transition from $\mathfrak{A}_i$ to $\mathfrak{A}_{i+1}$. Consider $a \in A$ and define $\zeta = \mathrm{fpr}^{\mathfrak{A}_i}_M[a]$ and $\eta = \mathrm{fpr}^{\mathfrak{A}_{i+1}}_M[a]$; we show that $\zeta = \eta$. We may assume that $a \in C_i$, for otherwise there is nothing to show. Fix a fluted 1-type π. Since $\mathfrak{t}^{\mathfrak{A}_{i+1}} \subseteq \mathfrak{t}^{\mathfrak{A}_i}$, $\zeta^+(\pi) \geq \eta^+(\pi)$ and $\zeta^-(\pi) \leq \eta^-(\pi)$. To show the reverse comparisons, observe that we can find $\zeta^+(\pi)$ elements $b \in (A \setminus C_i) \cup Q$ such that $\mathrm{ftp}^{\mathfrak{A}_i}[b] = \pi$ and $\mathfrak{A}_i \models \mathfrak{t}[Q, b]$. But for each such b, $\mathfrak{A}_{i+1} \models \mathfrak{t}[a, b]$, by construction of $\mathfrak{A}_{i+1}$. Hence $\eta^+(\pi) \geq \zeta^+(\pi)$. Turning now to ζ^-, suppose $\zeta^-(\pi) < M$. Then, by Lemma 5 the only elements $b \in C_i$ such that $\mathfrak{A}_i \models \mathfrak{t}[a, b]$ and $\mathrm{ftp}^{\mathfrak{A}_i}[b] = \pi$ must lie in Q, and these witnesses are preserved in $\mathfrak{A}_{i+1}$. Hence the number of elements b such that $\mathfrak{A}_{i+1} \not\models \mathfrak{t}[a, b]$ and $\mathrm{ftp}^{\mathfrak{A}_{i+1}}[b] = \pi$ is at most $\zeta^-(\pi)$. This establishes our claim that $\zeta = \eta$, and hence, that the M-bounded, fluted 1-profiles are preserved. In particular, $\mathfrak{A}_{i+1} \models \varphi$.

From the above construction, for any pair of integers i, j, the induced substructure $\mathfrak{A}_k\restriction(A_i \cup A_j)$ stays the same for all $k > \max(i, j)$. Thus, it makes sense to speak about the limit structure $\mathfrak{A}'$ obtained by running the above process to infinity. Moreover, since each $\mathfrak{t}^{\mathfrak{A}_i}$ is transitive, $\mathfrak{t}^{\mathfrak{A}'}$ is also transitive, and $\mathfrak{A}' \models \varphi$. Finally, in the model $\mathfrak{A}'$, no cluster having a superior clique contains any clique path of length greater than 1.

Stage 2. The goal of this stage is to add elements to stable cliques in clusters lacking a superior clique. We modify $\mathfrak{A}'$, proceeding clique by clique. Number the cliques of $\mathfrak{A}'$ as $Q_0, Q_1, \ldots$, and let $\mathfrak{A}'_0 = \mathfrak{A}'$. Supposing $\mathfrak{A}'_i$ to have been defined, we build the structure $\mathfrak{A}'_{i+1}$. If Q_i either occurs in a cluster having a superior clique, or is not stable, we do nothing, and set $\mathfrak{A}'_{i+1} = \mathfrak{A}'_i$. Otherwise, we form $\mathfrak{A}'_{i+1}$ as follows. Letting Π be the set of fluted 1-types sensitive for Q_i, for each $\pi \in \Pi$, we add M elements to Q_i with fluted 1-type π. More precisely, let X be a set of $M \cdot |\Pi|$ fresh elements, and let $A'_{i+1} = A_i \cup X$. We define $\mathfrak{A}'_{i+1}$ on this set by declaring M elements of X to have fluted 1-type π for each $\pi \in \Pi$, setting $\mathfrak{t}^{\mathfrak{A}'_{i+1}}$ to be total on $Q_i \cup X$, and finally setting $\mathfrak{A}'_{i+1} \models \mathfrak{t}[a, Q_i \cup X]$ if and only if $\mathfrak{A}'_i \models \mathfrak{t}[a, Q_i]$ and $\mathfrak{A}'_{i+1} \models \mathfrak{t}[Q_i \cup X, a]$ if and only if $\mathfrak{A}'_i \models \mathfrak{t}[Q_i, a]$, for all $a \in A'_i \setminus Q_i$.

We again claim that the M-bounded fluted profiles of existing elements do not change. Given $a \in A'_i$, define $\zeta = \mathrm{fpr}_M^{\mathfrak{A}'_i}[a]$ and $\eta = \mathrm{fpr}_M^{\mathfrak{A}'_{i+1}}[a]$, and fix a fluted 1-type $\pi \in \Pi$; we show that $\zeta(\pi) = \eta(\pi)$. Since elements having fluted 1-type π are added in the construction of $\mathfrak{A}'_{i+1}$, $\eta^+(\pi) \geq \zeta^+(\pi)$ and $\eta^-(\pi) \geq \zeta^-(\pi)$; we establish the reverse inclusions.

If $\mathfrak{A}'_i \models \mathfrak{t}[a, Q_i]$, then a is unrelated by $\mathfrak{t}$ to exactly the same elements in both structures, whence $\eta^-(\pi) = \zeta^-(\pi)$. Now, let C be the cluster containing Q_i. Since π is sensitive for Q_i, there exists an element $b \in C \setminus Q_i$ such that $\mathrm{ftp}^{\mathfrak{A}'_i}[b] = \pi$ and $\mathfrak{A}'_i \models \mathfrak{t}[Q_i, b]$. And since C has no superior clique, by Lemma 5 we have $\zeta^+(\pi) = M \geq \eta^+(\pi)$. If, on the other hand, $\mathfrak{A}'_i \not\models \mathfrak{t}[a, Q_i]$, then a is related by $\mathfrak{t}$ to exactly the same elements in both structures, whence $\eta^+(\pi) = \zeta^+(\pi)$. Furthermore, since $\pi \in \Pi$ and Q_i is stable, a is unrelated by $\mathfrak{t}^{\mathfrak{A}'_i}$ to at least M elements of fluted 1-type π, whence $\zeta^-(\pi) = M \geq \eta^-(\pi)$. This establishes that $\zeta = \eta$, and hence, our claim that the M-bounded fluted 1-profiles are preserved.

Since the elements of X, i.e. the new elements of $\mathfrak{A}'_{i+1}$, are part of the clique $Q_i \cup X$ of $\mathfrak{A}'_{i+1}$, we see that $\mathfrak{A}'_{i+1}$ and $\mathfrak{A}'_i$ realize the same M-bounded fluted 1-profiles. Indeed, they realize the same M-bounded fluted 1-star-types. For even if Q_i contains no elements of fluted 1-type $\pi \in \Pi$, some other element in the same cluster as Q_i will. Hence, $\mathfrak{A}_{i+1} \models \varphi$. We remark also that no fluted 1-types are sensitive for the clique $Q_i \cup X$ of $\mathfrak{A}'_{i+1}$: any required witnesses are provided by X. That is, the clique $Q_i \cup X$ is trivially stable. Using the same reasoning as in Stage 1, we may speak about the limit structure $\mathfrak{A}''$ obtained by running the above process to infinity; evidently, $\mathfrak{t}^{\mathfrak{A}''}$ is transitive, and $\mathfrak{A}'' \models \varphi$. Finally, in the model $\mathfrak{A}''$, no cluster having a superior clique contains any clique path of length greater than 1, and, in all remaining clusters, all stable cliques are trivially stable.

Stage 3. The goal of this stage is to flatten any remaining clusters. We finally modify $\mathfrak{A}''$, proceeding cluster by cluster. Number the clusters of $\mathfrak{A}''$ as $D_0, D_1, \ldots$ and let $\mathfrak{A}''_0 = \mathfrak{A}''$. Supposing $\mathfrak{A}''_i$ to have been defined, we build the structure $\mathfrak{A}''_{i+1}$. If D_i has a superior clique, we do nothing, and set $\mathfrak{A}''_{i+1} = \mathfrak{A}''_i$. Otherwise, we call those elements of D_i lying in a stable clique *stable*, and the remainder *unstable*, and we define $\mathfrak{A}''_{i+1}$ to be the same as $\mathfrak{A}''_i$ except that the relation $\mathfrak{t}^{\mathfrak{A}''_{i+1}}$ on D_i is taken to be

$$\{\langle a, b\rangle : \mathfrak{A}''_i \models \mathfrak{t}[a, b] \text{ and } \mathfrak{A}''_i \models \mathfrak{t}[b, a]\} \cup \{\langle a, b\rangle : \mathfrak{A}''_i \models \mathfrak{t}[a, b],\ a \text{ is unstable, and } b \text{ is stable}\}.$$

Thus, $\mathfrak{t}^{\mathfrak{A}''_{i+1}} \subseteq \mathfrak{t}^{\mathfrak{A}''_i}$, and the two relations agree on pairs of elements which do not both lie in D_i. To check that $\mathfrak{t}^{\mathfrak{A}''_{i+1}}$ is transitive, we recall that there is no triple of elements c, d, e with $c, e \in D_i$, $d \notin D_i$ such that $\mathfrak{A}''_i \models \mathfrak{t}[c, d]$ and $\mathfrak{A}''_i \models \mathfrak{t}[d, e]$. Furthermore, it is immediate from the above construction of $\mathfrak{A}''_{i+1}$ that no clique path in D_i has length greater than 1. Consider $a \in A''$ and define $\zeta = \mathrm{fpr}_M^{\mathfrak{A}''_i}[a]$ and $\eta = \mathrm{fpr}_M^{\mathfrak{A}''_{i+1}}[a]$; we show that $\zeta = \eta$. We may assume that $a \in D_i$ and D_i has no superior clique, for otherwise there is nothing to show. Since $\mathfrak{t}^{\mathfrak{A}''_{i+1}} \subseteq \mathfrak{t}^{\mathfrak{A}''_i}$, $\zeta^+(\pi) \geq \eta^+(\pi)$ and $\zeta^-(\pi) \leq \eta^-(\pi)$.

To show the reverse comparisons, consider first ζ^+ and η^+, and fix a fluted 1-type π. If the element a lies in a stable clique Q, then, by the construction of $\mathfrak{A}''$, Q is trivially stable, whence π is not sensitive for Q. In other words we can find $\zeta^+(\pi)$ elements $b \in (A'' \setminus D_i) \cup Q$ such that $\mathrm{ftp}^{\mathfrak{A}''_i}[b] = \pi$ and $\mathfrak{A}''_i \models \mathfrak{t}[Q, b]$. But for each such b, $\mathfrak{A}''_{i+1} \models \mathfrak{t}[a, b]$, by construction of $\mathfrak{A}''_{i+1}$. On the other hand, if a lies in an unstable clique Q, by Lemma 6, there is a stable clique $Q' \subseteq D_i$ such that $\mathfrak{A}''_i \models \mathfrak{t}[Q, Q']$, whence the same reasoning applies. Hence $\eta^+(\pi) \geq \zeta^+(\pi)$.

Turning now to ζ^- and η^-, suppose $\zeta^-(\pi) < M$ (for otherwise $\zeta^-(\pi) \geq \eta^-(\pi)$ trivially). Then, since D_i by assumption contains no superior clique, it follows by Lemma 5 that there is no $b \in D_i$ such that $\mathfrak{A}''_i \models \mathfrak{t}[a, b]$ and $\mathrm{ftp}^{\mathfrak{A}''_i}[b] = \pi$. That is: although a is unrelated to more elements in $\mathfrak{A}''_{i+1}$ than in $\mathfrak{A}''_i$, none of them has fluted 1-type π. This implies $\eta^-(\pi) \leq \zeta^-(\pi) < M$, since $\mathfrak{t}$ is unchanged on pairs of elements at least one of which is not in D_i. This establishes that $\zeta = \eta$. Hence, the same M-bounded fluted 1-star-types are realized in $\mathfrak{A}''_i$ and $\mathfrak{A}''_{i+1}$, and therefore $\mathfrak{A}''_{i+1} \models \varphi$.

Using the same reasoning as in Stage 1, we may speak about the limit structure $\mathfrak{B}$ obtained by running the above process to infinity; evidently, $\mathfrak{t}^{\mathfrak{B}}$ is transitive, and $\mathfrak{B} \models \varphi$. Finally, in the model $\mathfrak{B}$, no cluster contains any clique path of length greater than 1. Since, in any clique path $Q_1, \ldots, Q_s$, $\mathrm{fpr}^{\mathfrak{B}}[Q_i] \preceq \mathrm{fpr}^{\mathfrak{B}}[Q_{i+1}]$ for all i $(1 \leq i < s)$, and each $\mathrm{fpr}^{\mathfrak{B}}[Q_i]$ consists of $2^{|\Sigma|}$ integers in the range $[0, M]$, the depth of $\mathfrak{B}$ is at most $2(M+1)2^{|\Sigma|}$. ◀

▶ **Lemma 8.** *Let φ be a formula of $\mathcal{FLU}^2_u$+1Tr of the form (4) with $k = 2$, featuring the constant M. If φ has a model of depth D, then it has a model of depth D in which all cliques contain at most M elements having any given fluted 1-type π.*

Proof. Let $\mathfrak{A}$ be a model of depth D. For every clique Q in $\mathfrak{A}$ and fluted 1-type π, select M elements (all if there are fewer) in Q having fluted 1-type π, and let $\mathfrak{B}$ be the restriction of $\mathfrak{A}$ to the selected elements. It is obvious that $\mathrm{fst}^{\mathfrak{A}}_M[b] = \mathrm{fst}^{\mathfrak{B}}_M[b]$ for all $b \in B$ and indeed that $\mathfrak{A}$ and $\mathfrak{B}$ realize the same M-bounded, fluted 1-star-types. ◀

▶ **Lemma 9.** *Let φ be a formula of $\mathcal{FLU}^2_u$+1Tr of the form (4) with $k = 2$. If φ is satisfiable, then it has a model of size bounded by $2^{2^{O(\#(\varphi))}}$.*

Proof. Let Σ be the signature of φ. We may assume that the constant M featured in φ as given by (4) is at least 1. Write $L = 2^{|\Sigma|-1}$ for the number of fluted 1-types over Σ. By Lemmas 7 and 8, let $\mathfrak{A} \models \varphi$ with $\mathfrak{A}$ of depth at most $D = (M+1) \cdot 2^{|\Sigma|+1}$, and in which all cliques contain at most M elements having any given fluted 1-type π. Define the *level* of any clique Q of $\mathfrak{A}$ to be the length of the longest clique path in $\mathfrak{A}$ ending in Q plus 1. Thus, the minimum possible level is 1, and the maximum, D. For each i in this range, let A_i be the union of those cliques at level i.

We select cliques from A_i as follows. At each level i $(1 \leq i \leq D)$, select, for each fluted 1-type π, up to $(M+1)$ distinct cliques containing elements whose fluted 1-type is π (if there are fewer than $(M+1)$, then select all of them). The total number of cliques thus selected for each level i is therefore at most $(M+1)L \leq 2ML$. Next, considering successive levels i in the range $[2, D]$ (starting with $i = 2$), for each already selected clique $Q \subseteq A_1 \cup \cdots \cup A_{i-1}$, and for each fluted 1-type π, select up to M distinct cliques $Q' \subseteq A_i \cup \cdots \cup A_D$ containing elements whose fluted 1-type is π such that $\mathfrak{A} \models \mathfrak{t}[Q, Q']$, and select at least M distinct cliques $Q' \subseteq A_i \cup \cdots \cup A_D$ containing elements whose fluted 1-type is π such that $\mathfrak{A} \not\models \mathfrak{t}[Q, Q']$ (in both cases, if there are fewer than M, then select all of them). The number of selected cliques on level i is thus at most

$$(2ML) + (2ML)^2 + \cdots + (2ML)^i \leq (2ML)^{i+1}.$$

Let the sub-structure consisting of the selected cliques be $\mathfrak{B}$. It is easy to see that every element of $\mathfrak{B}$ realizes the same M-bounded fluted 1-star-type that it realizes in $\mathfrak{A}$. By Lemma 3, $\mathfrak{B} \models \varphi$. It remains to establish the size of $\mathfrak{B}$. Summing over all levels i from 1 to D, the total number of selected cliques is at most

$$(2ML)^2 + (2ML)^3 + \cdots + (2ML)^{D+1} \leq (2ML)^{D+2}.$$

Bearing in mind that each clique contains at most $M < 2M$ elements of each of the L fluted 1-types, we obtain $|B| < (2ML)^{D+3} = \left(M \cdot 2^{|\Sigma|}\right)^{(M+1)\cdot 2^{|\Sigma|+1}+3}$. ◀

6 Unrestricted signatures and any number of variables

We begin this section by showing that $\mathcal{FLU}^2$+1Tr has the finite model property, and that Sat($\mathcal{FLU}^2$+1Tr) is in 3-NExpTime. We proceed to show that the entire logic $\mathcal{FLU}$+1Tr has the finite model property, and that Sat($\mathcal{FLU}^k$+1Tr) is in (k+1)-NExpTime for all $k \geq 2$.

▶ **Lemma 10.** *Given an $\mathcal{FLU}^2$+1Tr-formula φ, there exists an $\mathcal{FLU}^2_u$+1Tr-formula ψ such that ψ and φ are satisfiable over the same domains, and $\#(\psi) \leq 2^{\#(\varphi)}$.*

Proof. Let φ, over a signature Σ, be given by

$$\bigwedge_{h=1}^{m} \forall(\alpha_h \to Q(1, \Sigma, M, \Theta_h)),$$

where m is a positive integer, the α_h quantifier-free $\mathcal{FLC}^1$-formulas over Σ, M a non-negative integer and the Θ_h existential Presburger formulas with free variables $\bar{v}$ indexed by the fluted 2-types over Σ. Let Σ^- be the result of removing from Σ all binary predicates other than the distinguished predicate $\mathfrak{t}$, and let $\bar{u}$ be the set of variables u_ρ indexed by the fluted 2-types over Σ^-. For any index set $H \subseteq [1, m]$, define the Presburger formula $\Theta_H(\bar{u})$ to be

$$\exists(\bar{v} \leq M) \left(\bigwedge_{h \in H} \Theta_h(\bar{v}) \wedge \bigwedge_{\rho \in \mathrm{Ftp}_2^{\Sigma^-}} \left(u_\rho \geq \sum \{v_\tau : \tau \in \mathrm{Ftp}_2^{\Sigma} \text{ s.t. } \models \tau \to \rho\} \right) \wedge \right.$$
$$\left. \bigwedge_{\rho \in \mathrm{Ftp}_2^{\Sigma^-}} \left(\bigwedge \{v_\tau < M : \tau \in \mathrm{Ftp}_2^{\Sigma} \text{ s.t. } \models \tau \to \rho\} \to u_\rho \leq \sum \{v_\tau : \tau \in \mathrm{Ftp}_2^{\Sigma} \text{ s.t. } \models \tau \to \rho\} \right) \right).$$

(We adopt the usual convention that $\bigwedge \emptyset = \top$.) Intuitively, $\Theta_H(\bar{u})$ is intended to characterize those fluted 1-profiles over Σ^- which can be consistently extended to a fluted 1-profile over Σ satisfying all of the existential Presburger formulas Θ_h, for $h \in H$. By renaming bound variables, any existential quantifiers in Θ_H can be moved to the front. Thus we may regard Θ_H as an existential Presburger formula. Now let $L = 2^{(|\Sigma| - |\Sigma^-|)}$, and let ψ be

$$\bigwedge_{H \subseteq [1,m]} \forall \left(\left(\bigwedge_{h \in H} \alpha_h \right) \to Q(1, \Sigma^-, LM, \Theta_H) \right).$$

It is immediate that $\#(\psi) = |\Sigma^-| + \log(LM) + 2^m \leq |\Sigma^-| + \left(|\Sigma| - |\Sigma^-| + \log M\right) + 2^m \leq 2^{\#(\varphi)}$. Note that the signature of ψ is Σ^-; in other words, ψ is an $\mathcal{FLU}^2_u$+1Tr-formula. We show that φ and ψ are satisfiable over the same domains.

Suppose $\mathfrak{A} \models \varphi$ and let $\mathfrak{A}^-$ be the reduct of $\mathfrak{A}$ to Σ^-. We show that $\mathfrak{A}^- \models \psi$. Fix $a \in A$, and suppose $H \subseteq \{h \in [1,m] \colon \mathfrak{A} \models \alpha_h[a]\}$. It suffices to show that the LM-bounded, fluted profile of a in $\mathfrak{A}^-$ satisfies $\Theta_H(\bar{u})$. Let us therefore assign values to the various u_ρ (for $\rho \in \mathrm{Ftp}_2^{\Sigma^-}$) accordingly: $u_\rho \leftarrow \min\big(|\{b \in A \colon \mathfrak{A}^- \models \rho[a,b]\}|, LM\big)$. To show satisfaction of Θ_H, we must find values for the existentially quantified variables. For each $\tau \in \mathrm{Ftp}_2^{\Sigma}$, we let the variable v_τ be assigned the value given by the M-bounded, fluted profile of a in $\mathfrak{A}$, namely: $v_\tau \leftarrow \min\big(|\{b \in A \colon \mathfrak{A} \models \tau[a,b]\}|, M\big)$. We consider the three groups of conjuncts in the body of Θ_H in turn. Since $\mathfrak{A} \models \varphi$, we certainly have (under this valuation) $\Theta_h(\bar{v})$ for all $h \in H$. For the second group of conjuncts, fix $\rho \in \mathrm{Ftp}_2^{\Sigma^-}$, and suppose first that $u_\rho < LM$. Then

$$\begin{aligned} u_\rho = |\{b \in A \colon \mathfrak{A}^- \models \rho[a,b]\}| &= \sum\{|\{b \in A \colon \mathfrak{A} \models \tau[a,b]\}| \,:\, \tau \in \mathrm{Ftp}_2^{\Sigma} \text{ s.t. } \models \tau \to \rho\} \\ &\geq \sum\{v_\tau \colon \tau \in \mathrm{Ftp}_2^{\Sigma} \text{ s.t. } \models \rho \to \tau\}, \end{aligned}$$

as required. And of course, if $u_\rho \geq LM$, then the conjunct follows from the fact that $\bar{v} \leq M$ and $|\{\tau \in \mathrm{Ftp}_2^{\Sigma} \colon \models \tau \to \rho\}| = L$. For the final group of conjuncts, again fix $\rho \in \mathrm{Ftp}_2^{\Sigma^-}$, and suppose that $v_\tau < M$ for all v_τ such that $\tau \in \mathrm{Ftp}_2^{\Sigma}$ and $\models \rho \to \tau$. Then, applying essentially similar reasoning,

$$\begin{aligned} u_\rho \leq |\{b \in A \colon \mathfrak{A}^- \models \rho[a,b]\}| &= \sum\{|\{b \in A \colon \mathfrak{A} \models \tau[a,b]\}| \,:\, \tau \in \mathrm{Ftp}_2^{\Sigma} \text{ s.t. } \models \tau \to \rho\} \\ &= \sum\{v_\tau \colon \tau \in \mathrm{Ftp}_2^{\Sigma} \text{ s.t. } \models \rho \to \tau\}. \end{aligned}$$

Thus, a satisfies the formula $Q(1, \Sigma^-, LM, \Theta_H)$ in $\mathfrak{A}^-$, and hence $\mathfrak{A}^- \models \psi$.

Finally, suppose $\mathfrak{B} \models \psi$, with $\mathfrak{B}$ interpreting Σ^-. We expand to a model $\mathfrak{B}^+ \models \varphi$ by interpreting the predicates in $\Sigma \setminus \Sigma^-$. All such predicates are of course binary. Consider any $a \in B$, and let $H = \{h \in [1,m] \colon \mathfrak{B} \models \alpha_h[a]\}$. Thus, the LM-bounded, fluted profile of a in $\mathfrak{B}$ satisfies $\Theta_H(\bar{u})$. To avoid notational clutter, we write the names of the variables in $\bar{u}$ to denote their values under this assignment, so that u_ρ denotes the value $\big(\mathrm{fpr}^{\mathfrak{B}}_{LM}[a]\big)(\rho)$. Similarly, we write the names of the variables in $\bar{v}$ to denote some collection of values witnessing the existentially quantified statement in Θ_H. Now fix some $\rho \in \mathrm{Ftp}_2^{\Sigma^-}$ and let $B_\rho = \{b \in B \colon \mathrm{ftp}^{\mathfrak{B}}[a,b] = \rho\}$. Thus, $u_\rho = \min(LM, |B_\rho|)$. We have two cases to consider. If $v_\tau < M$ for all $\tau \in \mathrm{Ftp}_2^{\Sigma}$ such that $\models \tau \to \rho$, then, by the second and third conjunct groups of Θ_H, $u_\rho = \sum\{v_\tau \colon \tau \in \mathrm{Ftp}_2^{\Sigma} \text{ s.t. } \models \tau \to \rho\}$, whence we may partition B_ρ into sets B_τ of cardinality v_τ, with τ varying over the fluted 2-types over Σ extending ρ. If, on the other hand, $v_\tau = M$ for some $\tau \in \mathrm{Ftp}_2^{\Sigma}$ such that $\models \tau \to \rho$, pick any such τ, say τ_0. By the second conjunct group of Θ_H, $u_\rho \geq \sum\{v_\tau \colon \tau \in \mathrm{Ftp}_2^{\Sigma} \text{ s.t. } \models \tau \to \rho\}$, so we may partition B_ρ into sets B_τ, such that B_τ has cardinality v_τ for all τ *except* τ_0, with B_{τ_0} mopping up any remaining elements. Observe that $|B_{\tau_0}| \geq M$. Having constructed the various sets B_τ, we partially define $\mathfrak{B}^+$ by interpreting the predicates of $\Sigma \setminus \Sigma^-$ (all binary) in such a way that $\mathfrak{B} \models \tau[a,b]$ for every $b \in B_\tau$, and every $\tau \in \mathrm{Ftp}_2^{\Sigma}$ such that $\models \tau \to \rho$. However $\mathfrak{B}^+$ is completed, we are assured that $\mathrm{ftp}^{\mathfrak{B}^+}[a] = \bar{v}$, and thus, by Θ_H, satisfies Θ_h for every $h \in H$. Now repeat this process for every $a \in B$. Because only *fluted* 2-types are being defined here, no pair of elements is re-assigned, and once all elements have been considered, $\mathfrak{B}^+$ will be fully defined, and will satisfy $\mathfrak{B}^+ \models \varphi$. ◀

It remains to show that $\mathcal{FLU}$+1Tr has the finite model property, and that $\mathrm{Sat}(\mathcal{FLU}^k$+1Tr) is in $(k$+1)-NExpTime for all $k \geq 2$. We proceed by reducing the problem $\mathrm{Sat}(\mathcal{FLU}^{k+1}$+1Tr) (with exponential blow-up) to the problem $\mathrm{Sat}(\mathcal{FLU}^k$+1Tr). Modulo some technical simplifications, this work repeats [27, Section 4], where a similar reduction was carried out

in the absence of a distinguished predicate interpreted as a transitive relation. Since this distinguished predicate is binary, it is not affected by the reduction in question. Due to space limits, the proof of the following Lemma is therefore omitted.

▶ **Lemma 11.** *Given an $\mathcal{FLU}^{k+1}$+1Tr-formula φ ($k \geq 2$), there exists an $\mathcal{FLU}^{k}$+1Tr-formula ψ such that ψ and φ are satisfiable over the same domains, and $\#(\psi)$ is $2^{O(\#(\varphi))}$.*

▶ **Theorem 12.** *Let φ be a $\mathcal{FLU}^{k}$+1Tr-formula ($k \geq 2$). If φ is satisfiable, then it has a model of cardinality bounded by some fixed $(k{+}1)$-tuply exponential function of $\#(\varphi)$.*

Proof. Induction on k: the base case ($k = 2$) follows from Lemmas 9 and 10; the inductive case is Lemma 11. ◀

▶ **Corollary 13.** *$\mathcal{FLC}$+1Tr has the finite model property, and* Sat($\mathcal{FLC}^{k}$+1Tr) *is in $(k{+}1)$-NExpTime for all $k \geq 2$.*

Proof. Let a formula $\varphi \in \mathcal{FLC}^{k}$+1Tr be given. By Lemma 1, we may assume that φ is in normal form. Now re-write φ as a logically equivalent $\mathcal{FLU}^{k}$+1Tr-formula ψ with $\#(\psi) \leq \|\varphi\|$. By Theorem 12, ψ – and hence φ – has a model of size bounded by a $(k{+}1)$-tuply exponential function of $\#(\psi)$. The result then follows by standard model-checking techniques. ◀

7 $\mathcal{FLC}$ and two transitive relations

In this section we show that the satisfiability and finite satisfiability problems for $\mathcal{FLC}^{2}$+2Tr are both undecidable. The result holds when the signature features – besides the two distinguished transitive relations – only unary predicates. Thus, we strengthen the undecidability result for $\mathcal{SHQ}$ [18] where three roles were used. The proof proceeds by reduction from undecidable tiling problems that are typical for two-variable logics. For instance, this technique was used in [30] to show undecidability of the (finite) satisfiability problems for $\mathcal{FL}^{2}$ in the presence of three transitive relations.

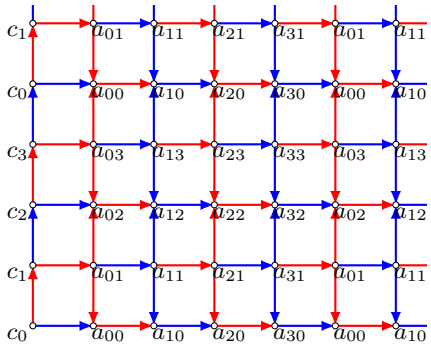
A *tiling system* is a tuple $\boldsymbol{\mathcal{C}} = (\mathcal{C}, H, V)$, where $\mathcal{C}$ is a finite set of *tiles*, and $H, V \subseteq \mathcal{C} \times \mathcal{C}$ are the *horizontal* and *vertical* constraints. A *tiling* of $\mathbb{N}^2$ for $\boldsymbol{\mathcal{C}}$ is a function $f : \mathbb{N}^2 \to C$, such that for all $i, j \in \mathbb{N}$, $(f(i,j), f(i+1,j)) \in H$ and $(f(i,j), f(i,j+1)) \in V$. A tiling is *periodic* if there exist m, n such that, for all i and j, $f(i+m, j) = f(i, j+n) = f(i,j)$. The *infinite* (*periodic*) *tiling problem* is the following: given a tiling system $\boldsymbol{\mathcal{C}}$, does there exist a (periodic) tiling of $\mathbb{N}^2$ for $\boldsymbol{\mathcal{C}}$? Our proof relies on the following result (see e.g. [4, p. 90]).

▶ **Proposition 14.** *The periodic tiling problem and the complement of the infinite tiling problem are recursively inseparable.*

To achieve the goal of this section, given a tiling system $\boldsymbol{\mathcal{C}}$, we construct an $\mathcal{FLC}^{2}$+2Tr-formula $\eta_{\boldsymbol{\mathcal{C}}}$ such that: (i) if $\mathbb{N}^2$ has a periodic tiling for $\boldsymbol{\mathcal{C}}$, then $\eta_{\boldsymbol{\mathcal{C}}}$ is finitely satisfiable; and (ii) if $\eta_{\boldsymbol{\mathcal{C}}}$ is satisfiable, then $\mathbb{N}^2$ has a tiling for $\boldsymbol{\mathcal{C}}$. The result then follows from Prop. 14.

The formula $\eta_{\boldsymbol{\mathcal{C}}}$ features a conjunct φ_{grid} whose canonical model, shown in Fig. 2, has the domain $\mathbb{N} \cup \{-1\} \times \mathbb{N}$. The signature of φ_{grid} consists of the two distinguished binary predicates b (blue) and r (red), together with the unary predicates: $a_{i,j}$ ($0 \leq i, j \leq 3$) and c_i ($0 \leq i \leq 3$). The formula φ_{grid} is a conjunction enforcing the following properties: (a) there exists an initial element satisfying c_0, and the unary predicates enforce a partition of the universe; (b) witness requirements for elements forming the leftmost column; (c) witness requirements for elements not on the leftmost column; (d) confluence. Properties (a)-(c) are typical formulas of $\mathcal{FL}^{2}$. The only conjuncts where counting is used are the confluence

conjuncts presented in Fig. 2. These conjuncts ensure that certain witnesses connected by the b and r relations must be identical. In this way we get a grid-like structure with short transitive paths that connect elements corresponding to both horizontal and vertical neighbours in a standard grid. One can also obtain finite models of φ_{grid} over a nearly toroidal grid structure $(\{-1\} \cup \mathbb{Z}_{4m}) \times \mathbb{Z}_{4m}$ $(m > 0)$ by identifying elements from columns 0 and $4m$ and from rows 0 and $4m$ of the canonical model.



Confluence conjuncts of φ_{grid}:

$$\bigwedge_{i\in\{1,3\}} \forall\,(c_i \to \exists_{[=1]}((b \vee r) \wedge a_{0,i+1}))$$

$$\bigwedge_{i\in\{0,2\}} \forall\,(c_i \to \exists_{[=1]}((b \vee r) \wedge a_{0,i}))$$

$$\bigwedge_{\substack{i\in\{0,1,2,3\}\\ j\in\{1,3\}}} \forall\,\big(a_{i,j} \to \exists_{[=1]}((b \vee r) \wedge a_{i+1,j+1}) \wedge \exists_{[=1]}((b \vee r) \wedge a_{i+1,j-1})\big).$$

Figure 2 Intended model of φ_{grid} (left) and the confluence conjuncts (right). The transitive relations b and r are depicted by blue and red arrows. Nodes with the coordinates $(-1, Y)$ satisfy the predicates $c_{Y \bmod 4}$; nodes with coordinates (X, Y) $(X \geq 0)$ satisfy the predicates $a_{X \bmod 4, Y \bmod 4}$. Addition and subtraction in indices of the confluence formulas are understood modulo 4.

The rest of the reduction is done in a standard fashion: using unary predicates representing tiles from $\mathcal{C}$ one adds to $\eta_{\mathcal{C}}$ conjuncts assigning tiles to elements of a model in such a way that the horizontal and vertical constraints are preserved. As a result one shows that: (i) from a periodic tiling of $\mathbb{N}^2$ a finite model of $\eta_{\mathcal{C}}$ can be built, and, (ii) from any model of $\eta_{\mathcal{C}}$ a tiling of $\mathbb{N}^2$ for $\mathcal{C}$ can be constructed. Hence we have

▶ **Theorem 15.** *The satisfiability problem and the finite satisfiability problem for* $\mathcal{FLC}^2$+2Tr *are both undecidable.*

We note that our proof of Theorem 15 is also valid when the two distinguished transitive relations are required to be *partial orders*. The same proof strategy does not work, however, if they are required to be *equivalence relations*. Nevertheless, it was shown in [26] that the satisfiability and finite satisfiability problems for the logic $\mathcal{C}^2$ with two equivalence relations are undecidable; and the formulas securing undecidability can easily be written as fluted formulas. The undecidability of the (finite) satisfiability problem for $\mathcal{FLC}^2$ with two equivalence relations then follows.

References

1 Bartosz Bednarczyk. Exploiting forwardness: Satisfiability and query-entailment in forward guarded fragment. In Wolfgang Faber, Gerhard Friedrich, Martin Gebser, and Michael Morak, editors, *Logics in Artificial Intelligence - 17th European Conference, JELIA 2021, Virtual Event, May 17-20, 2021, Proceedings*, volume 12678 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2021. doi:10.1007/978-3-030-75775-5_13.

2 Bartosz Bednarczyk, Emanuel Kieroński, and Piotr Witkowski. Completing the picture: Complexity of graded modal logics with converse. *Theory Pract. Log. Program.*, 21(4):493–520, 2021. doi:10.1017/S1471068421000065.

3 Michael Benedikt, Egor V. Kostylev, and Tony Tan. Two variable logic with ultimately periodic counting. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 112:1–112:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.112.

4 Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

5 Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Answering regular path queries in expressive description logics via alternating tree-automata. *Information and Computation*, 237:12–55, 2014. doi:10.1016/j.ic.2014.04.002.

6 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

7 Daniel Danielski and Emanuel Kieroński. Unary negation fragment with equivalence relations has the finite model property. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 285–294. ACM, 2018. doi:10.1145/3209108.3209205.

8 Daniel Danielski and Emanuel Kieroński. Finite satisfiability of unary negation fragment with transitivity. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 17:1–17:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.17.

9 Thomas Eiter, Magdalena Ortiz, and Mantas Simkus. Conjunctive query answering in the description logic SH using knots. *Journal of Computer and System Sciences*, 78(1):47–85, 2012. doi:10.1016/j.jcss.2011.02.012.

10 Birte Glimm, Ian Horrocks, and Ulrike Sattler. Unions of conjunctive queries in SHOQ. In Gerhard Brewka and Jérôme Lang, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*, pages 252–262. AAAI Press, 2008. URL: http://www.aaai.org/Library/KR/2008/kr08-025.php.

11 Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler. Conjunctive query answering for the description logic SHIQ. *J. Artif. Intell. Res.*, 31:157–204, 2008. doi:10.1613/jair.2372.

12 Tomasz Gogacz, Víctor Gutiérrez-Basulto, Yazmín Ibáñez-García, Jean Christoph Jung, and Filip Murlak. On finite and unrestricted query entailment beyond SQ with number restrictions on transitive roles. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1719–1725. ijcai.org, 2019. doi:10.24963/ijcai.2019/238.

13 Víctor Gutiérrez-Basulto, Yazmín Angélica Ibáñez-García, and Jean Christoph Jung. Number restrictions on transitive roles in description logics with nominals. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1121–1127. AAAI Press, 2017. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14357.

14 Víctor Gutiérrez-Basulto, Yazmín Angélica Ibáñez-García, and Jean Christoph Jung. Answering regular path queries over SQ ontologies. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1845–1852. AAAI Press, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16242.

15 Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000. doi:10.1093/jigpal/8.3.239.

16 Mark Kaminski and Gert Smolka. Terminating tableaux for $\mathcal{SOQ}$ with number restrictions on transitive roles. In Cristian S. Calude and Vladimiro Sassone, editors, *Theoretical Computer Science – 6th IFIP TC 1/WG 2.2 International Conference, TCS 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 213–228. Springer, 2010. doi:10.1007/978-3-642-15240-5_16.

17 Yevgeny Kazakov and Ian Pratt-Hartmann. A note on the complexity of the satisfiability problem for graded modal logics. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 407–416. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.17.

18 Yevgeny Kazakov, Ulrike Sattler, and Evgeny Zolin. How many legs do I have? Non-simple roles in number restrictions revisited. In Nachum Dershowitz and Andrei Voronkov, editors, *Proc. of the 14th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2007)*, volume 4790 of *Lecture Notes in Computer Science*, pages 303–317. Springer, 2007. doi:10.1007/978-3-540-75560-9_23.

19 Emanuel Kieroński. Results on the guarded fragment with equivalence or transitive relations. In C.-H. Luke Ong, editor, *Computer Science Logic, 19th International Workshop, CSL 2005, 14th Annual Conference of the EACSL, Oxford, UK, August 22-25, 2005, Proceedings*, volume 3634 of *Lecture Notes in Computer Science*, pages 309–324. Springer, 2005. doi:10.1007/11538363_22.

20 Emanuel Kieroński. On the complexity of the two-variable guarded fragment with transitive guards. *Inf. Comput.*, 204(11):1663–1703, 2006. doi:10.1016/j.ic.2006.08.001.

21 Emanuel Kieroński. Decidability issues for two-variable logics with several linear orders. In Marc Bezem, editor, *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway, Proceedings*, volume 12 of *LIPIcs*, pages 337–351. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.CSL.2011.337.

22 Emanuel Kieroński and Adam Malinowski. The triguarded fragment with transitivity. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 334–353. EasyChair, 2020. doi:10.29007/z359.

23 Emanuel Kieroński and Sebastian Rudolph. Finite model theory of the triguarded fragment and related logics. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470734.

24 Emanuel Kieroński and Lidia Tendera. Finite satisfiability of the two-variable guarded fragment with transitive guards and related variants. *ACM Trans. Comput. Log.*, 19(2):8:1–8:34, 2018. doi:10.1145/3174805.

25 Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. Comput.*, 6(3):467–480, 1977. doi:10.1137/0206033.

26 Ian Pratt-Hartmann. The two-variable fragment with counting and equivalence. *Mathematical Logic Quarterly*, 61(6):474–515, 2015. doi:10.1002/malq.201400102.

27 Ian Pratt-Hartmann. Fluted logic with counting. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 141:1–141:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.141.

28 Ian Pratt-Hartmann, Wiesław Szwast, and Lidia Tendera. Quine's fluted fragment is non-elementary. 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 39:1–39:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.39.

29 Ian Pratt-Hartmann, Wiesław Szwast, and Lidia Tendera. The fluted fragment revisited. *Journal of Symbolic Logic*, 84(3):1020–1048, 2019. doi:10.1017/jsl.2019.33.

30 Ian Pratt-Hartmann and Lidia Tendera. The fluted fragment with transitive relations. *Annals of Pure and Applied Logic*, 173(1):103042, 2022. doi:10.1016/j.apal.2021.103042.

31 William C. Purdy. Fluted formulas and the limits of decidability. *Journal of Symbolic Logic*, 61(2):608–620, 1996. doi:10.2307/2275678.

32 William C. Purdy. Complexity and nicety of fluted logic. *Studia Logica*, 71:177–198, 2002. doi:10.1023/A:1016596721799.

33 Sebastian Rudolph. Undecidability results for database-inspired reasoning problems in very expressive description logics. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, pages 247–257. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12787.

34 Wiesław Szwast and Lidia Tendera. The guarded fragment with transitive guards. *Ann. Pure Appl. Log.*, 128(1-3):227–276, 2004. doi:10.1016/j.apal.2004.01.003.

35 Stephan Tobies. Complexity results and practical algorithms for logics in knowledge representation, 2001. PhD Thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany.

36 Stephan Tobies. PSPACE reasoning for graded modal logics. *J. of Logic and Computation*, 11(1):85–106, 2001. doi:10.1093/logcom/11.1.85.

37 Evgeny Zolin. Undecidability of the transitive graded modal logic with converse. *J. Log. Comput.*, 27(5):1399–1420, 2017. doi:10.1093/logcom/exw026.

Parity Games of Bounded Tree-Depth

Konrad Staniszewski
University of Warsaw, Poland
IDEAS NCBR Sp. z o.o., Warsaw, Poland

Abstract

The exact complexity of solving parity games is a major open problem. Several authors have searched for efficient algorithms over specific classes of graphs. In particular, Obdržálek showed that for graphs of bounded tree-width or clique-width, the problem is in P, which was later improved by Ganardi, who showed that it is even in LOGCFL (with an additional assumption for clique-width case). Here we extend this line of research by showing that for graphs of bounded tree-depth the problem of solving parity games is in logspace uniform AC^0. We achieve this by first considering a parameter that we obtain from a modification of clique-width, which we call shallow clique-width. We subsequently provide a suitable reduction.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Parity Games, Circuits, Tree-Depth, Clique-Width

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.33

Related Version *Full Version*: `https://arxiv.org/abs/2211.02926` [28]

Funding *Konrad Staniszewski*: Work supported by the National Science Centre, Poland (grant no. 2021/41/B/ST6/03914).

Acknowledgements I want to thank the supervisor of my master's thesis Damian Niwiński and anonymous reviewers for valuable feedback.

1 Introduction

Parity games are two-player games played on directed graphs with ranked vertices $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V, rank : V \to \mathbb{N})$. Player X makes a move at vertices from V_{X} by moving a token along one of the outgoing edges. By making moves, players form a sequence of vertices called a play. A play may be finite or not, but it must be exhaustive. That is, whenever there is a possibility to make a move, then a move is made. Player E wins a play π if it is either finite and ends at a vertex from V_{O} or is infinite and the parity of the highest rank that occurs infinitely often is even, otherwise player O wins.

Parity games play an important role in system verification and synthesis [23]. However, the exact complexity of solving parity games is still an open problem. The fastest known algorithms work in time $n^{\mathcal{O}\left(\log\left(1+\frac{d}{\log(n)}\right)\right)}$ [22, 21], where n is the size of the graph and d is the number of different ranks. The problem was also tackled from the point of view of parameterized complexity. Obdržálek showed that in graphs with bounded tree-width [26] or clique-width [27] parity games can be solved in polynomial time. The result for tree-width was later improved to NC^2 by Fearnley and Schewe [13] and to LOGCFL by Ganardi [16]. The result for clique-width was improved by Ganardi to LOGCFL under an additional assumption that the k-expression is given [16]. It was also shown that parity games admit a polynomial-time algorithm on graphs with bounded entanglement [4], DAG-width [3], or Kelly-width [20] (for Kelly-width, decomposition is assumed to be a part of the input). For more about these classes see [17]. Calude, Jain, Khoussainov, Li, and Stephan in their breakthrough work about solving parity games in quasipolynomial time [5] showed that the problem is fixed-parameter tractable when parameterized by the number of ranks.

In this work, we show that solving parity games on graphs of bounded tree-depth is in logspace uniform AC^0 [1]. We start by introducing a parameter, which we call shallow clique-width, in Section 2. In Section 3, we introduce tools that are subsequently used in Section 4 to create an AC^0 algorithm for parity games on graphs of bounded shallow clique-width. In Section 5, we show how to efficiently reduce the problem of solving parity games on graphs of bounded tree-depth to solving parity games on graphs of bounded shallow clique-width. Our result improves over the ones of Obdržálek and Ganardi when considering graphs of bounded tree-depth. That is because logspace uniform AC^0 is a strict subclass of LOGCFL (parity is not in AC^0 [14, 1]), and graphs of bounded tree-depth have bounded tree-width. In addition to this, our intermediate result for shallow clique-width is an example of an AC^0 algorithm for solving parity games on a class of graphs that are not sparse (assuming an appropriate graph description is provided).

Related Work

The relation between uniform AC^0 and tree-depth was studied in [10, 11]. Elberfeld, Jakoby, and Tantau showed that MSO definable problems are in uniform AC^0 when considering structures whose Gaifman graphs have bounded tree-depth [11]. Elberfeld, Grohe, and Tantau showed that on graphs of bounded tree-depth FO, MSO, and GSO (guarded second-order logic) have the same expressive power [10]. The question of whether the winning regions of parity games are definable in MSO is unsolved for the case when the number of ranks is unbounded. Dawar and Grädel showed that they are not definable in μ-calculus (a fragment of MSO), but can be defined in GSO if we use a quasi-order over vertices (having a higher rank) along with a unary predicate OddRank [8]. However, this along with [10, 11] does not place parity games on graphs of bounded tree-depth in uniform AC^0 since the tree-depth of the Gaifman graphs of the structures considered in [8] is unbounded due to the aforementioned quasi-order.

We use a dynamic programming approach along the graph decomposition similar to the one of Obdržálek [27]. However, dealing with AC^0 requires more technical care. In particular, we need to provide definitions of operations that are computable in logspace uniform AC^0. Another difference is that we don't restrict our attention to t-strategies but to a potentially larger set of positional strategies. We also provide the efficient reduction from solving parity games on graphs of bounded tree-depth to solving parity games on graphs of bounded shallow clique-width.

2 Definitions and Notation

2.1 Basic Notation

We use $g = f[y \mapsto c]$ to define g that coincides with f except that $g(y) = c$. To define a partial function that coincides with f but is undefined on y we write $f[y \mapsto \text{undefined}]$. By $\text{Dom}(f)$ we denote the set of elements for which f is defined. Given a function $f : A \to B$ and finite or infinite sequence $s = a_0, a_1, \ldots$ of elements of A, we denote application of f to s by $f(s) = f(a_0), f(a_1), \ldots$.

[1] A problem is said to be in logspace uniform AC^0 if there is a constant k, a polynomial p, and a deterministic Turing machine that given an input size N, generates in space $\mathcal{O}(\log(N))$ the description of a circuit of size $\mathcal{O}(p(N))$ and depth at most k that solves the problem instances of size N. The circuit can be viewed as the directed graph with vertices of in-degree 0 denoted as input, out-degree 0 as output, and other vertices labeled with boolean operations ($\wedge$, $\vee$ or $\neg$; both $\wedge$ and $\vee$ can take an arbitrary number of arguments). Edges show computation flow. In such graphs, vertices are usually called gates. For a more detailed definition, see [2].

2.2 Arenas, Games, Colorings

An *arena* is a directed graph $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V)$ with vertices partitioned between players E and O. In a parity game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V, rank : V \to \mathbb{N})$, the behavior of player X is modeled by a partial function $\rho_{\mathrm{X}} : V^* \times V_{\mathrm{X}} \rightharpoonup V$ called *strategy*, which given the sequence of visited vertices and the vertex that the token is on, gives the next vertex to move the token to. Every strategy must be exhaustive and correct – whenever there is a possibility of player movement, then the function must be defined, and the token can only be moved along existing edges. A strategy ρ_{X} is *positional* if the result depends only on the vertex the token is on (i.e. for a vertex $v \in V_{\mathrm{X}}$ and a prefix of a play $\vec{\pi}$ we have $\rho_{\mathrm{X}}(\vec{\pi}, v) = \rho_{\mathrm{X}}(v)$). A play π is consistent with ρ_{X} if whenever it is the turn of player X, then the move is made according to ρ_{X}. A strategy ρ_{X} is winning for player X from a vertex v if all plays starting at v and consistent with ρ_{X} are winning for player X. It is known that parity games are globally positionally determined [12]. That is for $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank)$ there exists a partition of V into the winning regions W_{E} and W_{O} such that E has a positional strategy ρ_{E} that wins from every vertex in W_{E}, and O has a positional strategy ρ_{O} that wins from every vertex in W_{O}.

We define $\Pi_G(\rho_{\mathrm{X}}, v)$ as the set of plays in the arena G that start at the vertex v and are consistent with the strategy ρ_{X} for player X, whereas by $\vec{\Pi}_G(\rho_{\mathrm{X}}, v, w)$ we define their prefixes that end at w (i.e. $\vec{\Pi}_G(\rho_{\mathrm{X}}, v, w) = \{\pi[\dots i] : \pi[i] = w \wedge \pi \in \Pi_G(\rho_{\mathrm{X}}, v)\}$). To distinguish a play from its prefix, we denote plays using the symbol π and prefixes of plays using $\vec{\pi}$.

We say that $Color : V \to C_{\mathrm{E}} \uplus C_{\mathrm{O}}$ is a player-aware coloring of vertices from $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E)$ if vertices from V_X are colored using colors from C_{X}. We will sometimes write a parity game along with player-aware coloring of its arena and say that $G' = (V' = V'_{\mathrm{E}} \uplus V'_{\mathrm{O}}, E', rank', Color')$ is a subgame of $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank, Color)$ if and only if $V'_{\mathrm{E}} \subseteq V_{\mathrm{E}}$, $V'_{\mathrm{O}} \subseteq V_{\mathrm{O}}$, $E' \subseteq E$, $rank'$ is $rank$ with domain restricted to V' and $Color'$ is $Color$ with domain restricted to V'.

2.3 Shallow Clique-Width for Arenas

Here we define shallow clique-width for arenas, which can be viewed as clique-width [6, 7] with unions of arbitrary number of components (instead of only binary ones) and restricted term height.

▶ **Definition 1.** *A* tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ *consists of a rooted tree* T*, with all root-to-leaf paths having the same length, a set of colors* $C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}$*, a coloring* $Color$ *of leaves of* T*, and a function* D *that assigns to each internal node of* T *a subset of* $C \times C$*.*

For a tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ and a node l of T, by $\mathcal{TM}_l$ we mean the tree-model $(T_l, C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color', D')$ where T_l is the subtree of T rooted at l, $Color'$ is $Color$ with domain restricted to leaves of T_l, and D' is D with domain restricted to internal nodes of T_l.

▶ **Definition 2.** *We define an arena* $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V)$ *induced by* $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ *in the following way.*

- *If* T *consists of one node* v *and* $Color(v) \in C_{\mathrm{X}}$*, then it induces edgeless* G *with* $V_{\mathrm{X}} = \{v\}$*.*
- *Otherwise, let* l *be* T*'s root and let* $l_1, \dots l_m$ *be children of* l*. The arena induced by* $\mathcal{TM}$ *is created by taking union of arenas induced by* $\mathcal{TM}_{l_1}, \dots, \mathcal{TM}_{l_m}$*, and adding a directed edge from every vertex colored* c *to every vertex colored* c' *whenever* $\langle c, c' \rangle \in D(l)$*.*

For an example of a tree-model and the induced arena see Figure 1.

▶ **Definition 3.** *The* shallow clique-width *of an arena $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V)$ is the smallest k such that there exists a tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ with $|C| \leq k$ and T of height at most k such that the induced arena is G.*

One might observe that shallow clique-width is similar to the notion of shrub-depth [19, 18]. The main difference, however, is that for a shallow clique-width's tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ and an internal node l of T a pair $\langle c, c' \rangle \in D(l)$ affects all pairs $\langle v, v' \rangle$ of leaves in the subtree of T rooted at l such that $Color(v) = c$ and $Color(v') = c'$, whereas in shrub-depth it is additionally restricted to only those for which the lowest common ancestor in T is l.

It is worth mentioning that if a class of graphs has bounded shallow clique-width, then it has bounded clique-width and shrub-depth. The former observation follows from the definition of clique-width, the latter one from the fact that in the shallow clique-width's tree-model there are at most $2^{|C \times C|}$ sets $D(l)$ at each level of the tree-model. So at the expense of an increase in the number of colors, we can get rid of these sets and describe connections using only the distance to the lowest common ancestor and colors.

2.4 Tree-Depth

Tree-depth was originally defined for undirected graphs [24]. Here we adapt this definition to arenas by simply forgetting the orientation of edges.

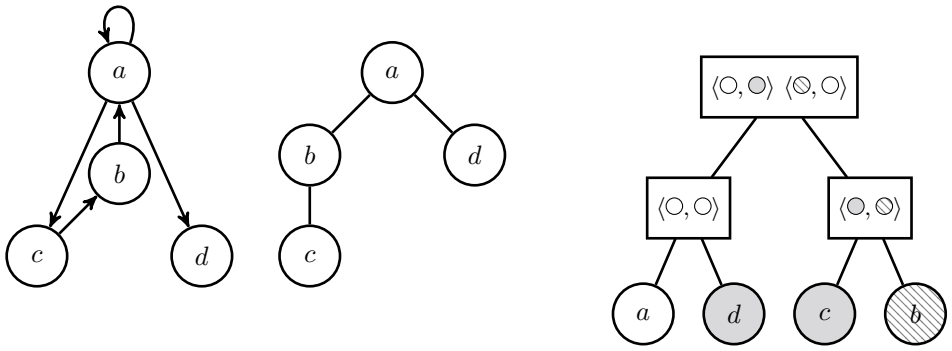
▶ **Definition 4.** *A forest of rooted trees $\mathcal{F}$ is an* elimination forest *of an arena $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V)$ if vertices of $\mathcal{F}$ are vertices of G and $\langle v, w \rangle \in E$ implies that v and w are in the same tree T in $\mathcal{F}$ and either v is an ancestor of w or w is an ancestor of v in T (see Figure 1). Tree-depth of an arena G is the smallest height of an elimination forest of G.*

▶ **Lemma 5** ([25, Chapter 6.2])**.** *If an arena G has an elimination forest of height k, then any simple path in G has length at most $2^{k+1} - 2$.*

The lemma above can be proven by induction on the height of the elimination forest.

One can think of graphs of bounded tree-depth as graphs of bounded tree-width [9, Chapter 12] with an additional constraint on the height of the tree-decomposition. More precisely, if a graph G has a tree-decomposition of height h where each bag contains at most k vertices, then it has tree-depth at most $k(h+1)$ [25, Chapter 6.4] (to obtain the elimination forest from the tree-decomposition, we leave each vertex of G in the bag that is closest to the root of the tree-decomposition and then replace each bag in the tree-decomposition with a path of its elements). What is more, if a graph has tree-depth k, then it admits a tree-decomposition of height bounded by $k+1$ where each bag contains at most $k+2$ vertices (to obtain the tree-decomposition, we take the elimination forest $\mathcal{F}$, make it a tree T by choosing a root r of an arbitrary tree in $\mathcal{F}$ and connecting other roots to r, and define bags for vertices of T as sets of vertices on vertex-r paths).

Shallow clique-width is more general than tree-depth. First, for an arena G of tree-depth k one can use $2(k+1)3^{k+1}$ colors to create a tree-model of height $k+2$ that induces G. This is because we can take some elimination forest $\mathcal{F}$ of G of height k and color each vertex according to its player-membership in G, and depth and connections to ancestors in $\mathcal{F}$ (for each vertex v and its ancestor w we have that either there is a directed edge from v to w, to v from w, or v and w are not connected in G). On the other hand, arenas where each vertex has a directed edge to every vertex have small shallow clique-width, but large tree-depth.



(a) Arena **(b)** Elimination forest **(c)** Tree-model

Figure 1 Arena (a), an elimination forest for the arena (b) and a tree-model that induces the arena (c) (all vertices belong to one player).

The major advantage of tree-depth over shallow clique-width is that given an arena G of tree-depth at most k, we know how to efficiently find an elimination forest of G of height at most k (details in Section 5.4). It is also worth noting that despite the simplicity of tree-depth, some problems are $W[1]$-*hard* when parameterized by it [15, Lemma 2, Proposition 5].

As mentioned before, in this work, we take a detour by first proving that solving parity games played on arenas of bounded shallow clique-width is in logspace uniform AC^0 (assuming an appropriate graph description is provided) and then providing a reduction. We decided to do so as thinking about vertices of the same color as the ones that will be treated similarly turned out to be conceptually simpler from our perspective. What is more, such an approach gave us a logspace uniform family of AC^0-circuits for a class of graphs that are not sparse.

3 Tools

Now we proceed with tools for constructing a logspace uniform family of circuits for parity games played on arenas with bounded shallow clique-width. The approach presented here is similar to the one used in [27] for creating a polynomial-time algorithm for parity games played on arenas of bounded clique-width. Here we also operate on some succinct information characterizing strategies that we call enforcements. Enforcements are an adaptation of Obdržálek's borders [26] from tree-width to shallow clique-width. The main idea behind borders is to store for a separating set S, a strategy ρ_{E}, and a vertex v information about the worst arrivals at vertices of S that opponent can achieve when starting from v and playing against ρ_{E}, whereas in enforcements we are interested in knowing what the worst arrivals at vertices of a specific color can be. Our adaptation of borders is analogous to the one considered in [27].

We additionally extend and modify this approach to obtain a logspace uniform family of constant-depth circuits.

3.1 Parity Games With Stop

▶ **Definition 6.** *We define a* stop parity game *as a parity game where player* E *has an additional move called* STOP *at his positions. Executing this move ends the game in a draw.*

▶ **Observation 7.** *In an arena $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V)$ with vertices ranked by $rank : V \to \mathbb{N}$, winning regions for player* E *in both stop and normal parity games are equal. What is more if a strategy ρ_{E} is winning in either of the games, then it is winning in the other one, as ρ_{E} cannot be forced to use STOP.*

▶ **Definition 8.** *We say that a strategy ρ_{E} is* safe from *v in a stop parity game G, if all plays that start at v and are consistent with ρ_{E} are not losing for player* E.

Stop parity games will turn out to be useful for solving parity games on arenas that can be extended.

3.2 Enforcements

▶ **Definition 9.** *Let* $\mathrm{elem}(s)$ *denote the set of elements that appear in the sequence s. For a game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank)$, we define a function $mr_G(\vec{\pi}) = \max(\mathrm{elem}(rank(\vec{\pi})))$. That is for a prefix of a play, return the maximum rank of its vertex.*

▶ **Definition 10.** *We define a total order $\prec$ on natural numbers as $p \prec q \Leftrightarrow (-1)^p p < (-1)^q q$ and $p \preceq q \Leftrightarrow (p \prec q \vee p = q)$. That is $\ldots \prec 5 \prec 3 \prec 1 \prec 0 \prec 2 \prec 4 \prec 6 \prec \ldots$.*

▶ **Definition 11.** *For a stop parity game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank)$, a strategy ρ_{E} and $v \in V$ we define a partial function*

$$\mathrm{venf}^G_{\rho_{\mathrm{E}},v}(w) = \begin{cases} \min_{\preceq}\{mr_G(\vec{\pi}) : \vec{\pi} \in \vec{\Pi}_G(\rho_{\mathrm{E}}, v, w) \wedge \rho_{\mathrm{E}}(\vec{\pi}) = \mathit{STOP}\} & \text{if } w \in V_{\mathrm{E}}. \\ \min_{\preceq}\{mr_G(\vec{\pi}) : \vec{\pi} \in \vec{\Pi}_G(\rho_{\mathrm{E}}, v, w)\} & \text{if } w \in V_{\mathrm{O}} \end{cases}$$

Where $\min_{\preceq}(A)$ is a partial function that for a finite subset of natural numbers A gives the smallest element of A according to $\preceq$ and is undefined for $A = \emptyset$.

Intuitively for $w \in V_{\mathrm{O}}$ the operation above returns the worst possible arrival at w that player O can force when playing against the strategy ρ_{E} and for $w \in V_{\mathrm{E}}$ the worst arrival at w such that ρ_{E} will decide to stop the game.

▶ **Definition 12.** *We naturally adapt the definition above to games $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank)$ with player-aware coloring $Color : V \to C_{\mathrm{E}} \uplus C_{\mathrm{O}}$:*

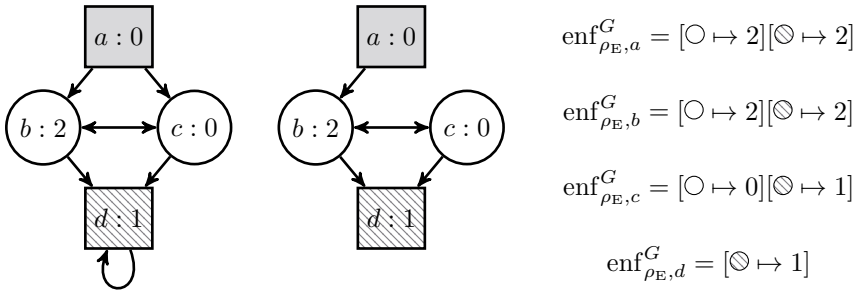
$$\mathrm{enf}^G_{\rho_{\mathrm{E}},v}(c) = \min_{\preceq}\{\mathrm{venf}^G_{\rho_{\mathrm{E}},v}(w) : w \in \mathrm{Dom}(\mathrm{venf}^G_{\rho_{\mathrm{E}},v}) \wedge Color(w) = c\}$$

An *enforcement* is any partial function from C to $\{0, \ldots, \max(rank(V))\}$. From now on by *enforcement from v* of a strategy ρ_{E} in G we will mean $\mathrm{enf}^G_{\rho_{\mathrm{E}},v}$. Note that this is always an enforcement and that ρ_{E} can have different enforcements from different vertices (see Figure 2).

▶ **Observation 13.** *Enforcement $\mathrm{enf}^G_{\rho_{\mathrm{E}},v}$ is defined for some color of player* E *(i.e. color c such that $c \in C_{\mathrm{E}}$) if and only if there is a finite play π consistent with ρ_{E} that starts at v such that $\rho_{\mathrm{E}}(\pi) = \mathit{STOP}$.*

▶ **Definition 14.** *We define a partial order $\sqsubseteq$ on enforcements as follows:*
$P \sqsubseteq Q \Leftrightarrow \mathrm{Dom}(P) \subseteq \mathrm{Dom}(Q) \wedge \forall a \in \mathrm{Dom}(P).Q(a) \preceq P(a)$
$P \sqsubset Q \Leftrightarrow P \sqsubseteq Q \wedge P \neq Q$.

Intuitively $P \sqsubset Q$ means that Q describes worse arrival scenarios from player E perspective than P.



(a) Parity game. **(b)** Parity game restricted to ρ_E.

Figure 2 A parity game G (a), G with the moves of player E restricted to the positional strategy $\rho_E = [\langle \overrightarrow{\pi}, a\rangle \mapsto b][\langle \overrightarrow{\pi}, d\rangle \mapsto \mathit{STOP}]$ along with the enforcements of ρ_E from various vertices (b). Square vertices belong to player E, round ones to O. The letters denote vertices, whereas the numbers inside vertices denote their ranks. Background shading denotes vertex color (vertices b and c have the same color, whereas colors of a, b and d are pairwise distinct).

Operations on Enforcements

Clearly, modifications of a strategy ρ_E can result in modifications of enforcements of this strategy. What is more, some of these modifications can be done directly on the enforcements without knowing the original strategy. In the further parts of this work, we will be interested in modifications that adapt strategies of player E from an arena G to an arena G', that was made from G by addition of directed edges between vertices of specified colors. It will usually suffice to remember enforcements of a strategy instead of the strategy itself and to perform respective operations on enforcements. Below we introduce some useful operations on enforcements.

The first operation corresponds to a modification of a strategy ρ_E to the strategy ρ'_E that behaves like ρ_E, but each time ρ_E decides to STOP at some vertex of a specified color c ρ'_E uses the option to move to the vertex from which the game was started. Moreover, we want to ensure that ρ'_E will never close a cycle with the highest rank being odd by doing so.

▶ **Definition 15.** *For $G = (V = V_E \uplus V_O, E \subseteq V \times V, rank : V \to \mathbb{N}, Color : V \to C_E \uplus C_O)$, enforcement P and color c we define*

$$\mathrm{loop}(P,c) = \begin{cases} P & \text{if } c \in C_E \wedge c \notin \mathrm{Dom}(P) \\ P[c \mapsto \mathit{undefined}] & \text{if } c \in C_E \wedge P(c) \text{ is even} \end{cases}$$

Please note that $\mathrm{loop}(P,c)$ is undefined if $c \in C_O$ or $P(c)$ is odd. Let P be the enforcement of ρ_E from some vertex v. Let c be any element of C_E and assume that from each vertex of color c there is an edge to v. Now, the first case in the definition of $\mathrm{loop}(P,c)$ corresponds to the situation when ρ_E will never stop at a vertex of color c when starting the game from v. Whereas the second one to the situation when ρ_E can be forced to stop at a vertex of color c, but the prefix of the play that will lead here will always have even max rank.

▶ **Definition 16.** *For $G = (V = V_E \uplus V_O, E \subseteq V \times V, rank : V \to \mathbb{N}, Color : V \to C_E \uplus C_O)$, $r \in \mathbb{N}$, color c and enforcements P, Q, we define two partial functions*

$$\mathrm{lift}(Q,r)(c) = \max(Q(c), r) \text{ if } c \in \mathrm{Dom}(Q)$$
$$\widetilde{\min}(P,Q)(c) = \min_{\preceq}(P(c), Q(c))$$

In this writing, we allow one of $\min_{\preceq}(P(c), Q(c))$ *arguments to be undefined and define* $\min_{\preceq}(P(c), Q(c))$ *to be the defined one (or undefined when both are undefined). The* $\max(Q(c), r)$ *in the definition of* $\mathrm{lift}(Q, r)(c)$ *is taken using the standard order on* $\mathbb{N}$.

Another operation is for composing strategies in a way that corresponds to playing one strategy and switching to another when either reaching vertices of specified color $c \in C_{\mathrm{O}}$ or stopping at vertices of specified color $c \in C_{\mathrm{E}}$.

▶ **Definition 17.** *For* $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V, rank : V \to \mathbb{N}, Color : V \to C_{\mathrm{E}} \uplus C_{\mathrm{O}})$, *color* c *and enforcements* P, Q, *we define*

$$\mathrm{merge}(P, c, Q) = \begin{cases} \widetilde{\min}(P, \mathrm{lift}(Q, P(c))) & \text{if } c \in \mathrm{Dom}(P) \wedge c \in C_{\mathrm{O}} \\ \widetilde{\min}(P[c \mapsto \textit{undefined}], \mathrm{lift}(Q, P(c))) & \text{if } c \in \mathrm{Dom}(P) \wedge c \in C_{\mathrm{E}} \\ P & \text{otherwise} \end{cases}$$

Intuitively, taking $\mathrm{lift}(Q, P(c))$ in the formula above corresponds to extending plays that were used to calculate P with the ones being used to calculate Q. More detailed explanation is the following.

▶ **Lemma 18.** *Properties of* $\mathrm{merge}(P, m, Q)$ *and* $\mathrm{loop}(P, m)$*:*

1. *Let* P, P', Q, Q' *be enforcements such that* $P' \sqsubseteq P$ *and* $Q' \sqsubseteq Q$, *then*

$$\mathrm{merge}(P', c, Q') \sqsubseteq \mathrm{merge}(P, c, Q)$$

2. *Let* c *be a color of player* O. *Let* ρ_{E}, ρ'_{E}, *be such that* ρ_{E} *is safe from* v, *whereas* ρ'_{E} *is safe from every vertex of color* c *in* G. *Let* ρ''_{E} *be like* ρ_{E}, *but when* ρ_{E} *reaches a vertex of color* c, ρ''_{E} *forgets about the past and switches permanently to* ρ'_{E} *behavior. Then* ρ''_{E} *is safe from* v *in* G *and*

$$\mathrm{enf}^G_{\rho''_{\mathrm{E}}, v} \sqsubseteq \mathrm{merge}(\mathrm{enf}^G_{\rho_{\mathrm{E}}, v}, c, \widetilde{\min}(\mathrm{enf}^G_{\rho'_{\mathrm{E}}, w_1}, \ldots, \mathrm{enf}^G_{\rho'_{\mathrm{E}}, w_m}))$$

   where $w_1, \ldots, w_m$ *are all vertices of color* c *and* $\widetilde{\min}(A, B, C) = \widetilde{\min}(\widetilde{\min}(A, B), C)$.
3. *Let* ρ_{E}, ρ'_{E}, *be strategies that are safe from* v *and* w *in* G *respectively. Let* c *be a color of player* E *such that from every vertex of color* c *there is an edge to* w, *and let* ρ''_{E} *be like* ρ_{E}, *but when* ρ_{E} *decides to stop at some vertex of color* c, ρ''_{E} *moves to* w *instead, forgets about the past and switches permanently to* ρ'_{E} *behavior. Then* ρ''_{E} *is safe from* v *in* G *and*

$$\mathrm{enf}^G_{\rho''_{\mathrm{E}}, v} = \mathrm{merge}(\mathrm{enf}^G_{\rho_{\mathrm{E}}, v}, c, \mathrm{enf}^G_{\rho'_{\mathrm{E}}, w})$$

4. *Let* ρ_{E} *be a strategy that is safe from* v *in* G. *Let* c *be a color of player* E *such that from every vertex of color* c *there is an edge to* v, *and let* ρ'_{E} *be like* ρ_{E}, *but each time* ρ_{E} *decides to stop at some vertex of color* c, ρ'_{E} *moves to* v, *forgets about the past and resumes as* ρ'_{E}. *Then if* $\mathrm{loop}(\mathrm{enf}^G_{\rho_{\mathrm{E}}, v}, c)$ *is defined, we have that* ρ'_{E} *is safe from* v *in* G *and*

$$\mathrm{enf}^G_{\rho'_{\mathrm{E}}, v} = \mathrm{loop}(\mathrm{enf}^G_{\rho_{\mathrm{E}}, v}, c)$$

The lemma above can be proven by simple but tedious case analysis. The key step in the proof is to show that $p \preceq p' \wedge q \preceq q'$ implies $\max(p, q) \preceq \max(p', q')$, what can be done by considering cases on the parity of p' and q'.

Enforcements and Strategies

For a set of enforcements $\mathcal{E}$ and a vertex v of a game G we consider the following properties.

1. (*soundness*) If $P \in \mathcal{E}$, then there exists a (possibly non-positional) strategy ρ_{E} that is safe from v in G such that $\mathrm{enf}^{G}_{\rho_{\mathrm{E}},v} \sqsubseteq P$.
2. (*completeness*) For each positional strategy ρ_{E} that is safe from v in G we have that $\mathrm{enf}^{G}_{\rho_{\mathrm{E}},v} \in \mathcal{E}$.
3. (*up-closure*) If $P \in \mathcal{E}$ and $P \sqsubseteq Q$ then $Q \in \mathcal{E}$.

We will say that $\mathcal{E}$ is *valid* for v if it is sound, complete and up-closed for v.

From now on we will assume that for a stop parity game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E, rank)$ we always have $rank(V) \subseteq \{0, \ldots, 2|rank(V)| - 1\}$, and denote $2|rank(V)|$ by d, as otherwise we can sort elements of $rank(V)$ and put them in the desired range.

▶ **Lemma 19.** *Given a set $\mathcal{E}$ of enforcements in G that is valid for v, we have that player* E *wins the parity game on G's arena from v if and only if $\mathcal{E}$ contains an enforcement that is undefined for every color of player* E.

Proof. If player E wins from v in the parity game played on G's arena, then there exists a positional strategy ρ_{E} that wins the parity game starting at v, and the enforcement generated by ρ_{E} from v in G is undefined for every color of player E (see Observation 13). What is more, since $\mathcal{E}$ is valid for v, from Observation 7 we have that $\mathrm{enf}^{G}_{\rho_{\mathrm{E}},v} \in \mathcal{E}$. On the other hand, if $\mathcal{E}$ contains an enforcement P that is undefined for every color of player E, then since $\mathcal{E}$ is valid for v, there exists a strategy ρ_{E} that is safe from v such that $\mathrm{enf}^{G}_{\rho_{\mathrm{E}},v} \sqsubseteq P$, what by Observation 13 means that ρ_{E} never uses STOP. ◀

In the following parts, we will aim to calculate for each vertex of a game a valid set of enforcements.

4 Circuit Family for Shallow Clique-Width

Now we will show how for $n, d > 1$, and $k > 0$ create in space $\mathcal{O}(\log(n) + \log(d)k)$ (and therefore time $n^{\mathcal{O}(1)}d^{\mathcal{O}(k)}$) the circuit $\mathbb{SCW}_{\langle n,d,k\rangle}$ of depth $\mathcal{O}(k^3)$, that given the specific representation of a parity game of shallow clique-width at most k, computes the partition into winning regions.

4.1 Overview

Circuit $\mathbb{SCW}_{\langle n,d,k\rangle}$ will be a combination of auxiliary circuits called gadgets. The circuit will input parity game's G arena as a tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ (details about the input are presented in Section 4.2), and it will process $\mathcal{TM}$ bottom-up.

The first layers of the circuit will calculate valid sets of enforcements for one-vertex arenas consisting of leaves of T. To do so gadgets defined in Section 4.3.1 will be used. The following layers of the circuit will correspond to computing valid sets of enforcements for larger and larger arenas induced by larger and larger subtrees of the given tree-model. For the last layers, we will use Lemma 19 to get the partition into winning regions.

To be more precise for an internal node l of T the circuit will consider a subgame G_l played on the arena induced by $\mathcal{TM}_l$. For each vertex of such G_l the circuit will calculate a valid set of enforcements using valid sets of enforcements calculated for subgames played on arenas induced by children of l. This calculation will be divided into several steps. First, a disjoint union of arenas induced by children of l will be taken, and then connections described

by $D(l)$ will be added one by one. For a pair $\langle s,t\rangle \in D(l)$ cases on whether s is a color of player E or O will be considered. Let G' be the subgame of G_l just before processing of $\langle s,t\rangle$ and G'' be G' after adding directed edges from all vertices of color s to all of color t in G'.

For the case when s is a color of player E, we will observe that a positional strategy ρ_{E} that is safe from a vertex v in G'' can be modified to a strategy ρ'_{E} such that ρ'_{E} is also safe from v, $\mathrm{enf}^{G''}_{\rho'_{\mathrm{E}},v} \sqsubseteq \mathrm{enf}^{G''}_{\rho_{\mathrm{E}},v}$ and ρ'_{E} behaves similarly to ρ_{E} but when ρ_{E} decides to move along a newly added edge, then ρ'_{E} chooses to move to some fixed vertex w. Given such ρ'_{E} we will be able to factor it into two strategies that are safe in G' from v and w respectively.

For the case when s is a color of player O, we will observe that a positional strategy of player E that is safe from v in G'' can be factored into a positional strategy that is safe from v in G' and a positional strategy that is safe from all vertices of color t in G'.

These observations will allow us to calculate valid sets of enforcements for vertices of G'' using valid sets of enforcements calculated for G'. These updates will be performed using gadgets defined in Section 4.3.3. The whole procedure of adding connections between vertices of specified colors will be handled by the gadget defined in Section 4.3.5. The details about connections between gadgets and therefore the construction of the whole circuit are presented in Section 4.4.

It is worth noting that the approach presented in this section fails for parity games of bounded shrub-depth. In short, that is because it makes use of the fact that in shallow clique-width, we specify connections using colors, whereas in shrub-depth, we additionally condition connections on the lowest common ancestor in the tree-model. To be more precise, when constructing an arena bottom-up from shallow clique-width's tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ during the processing of an internal vertex l of T for $\langle s,t\rangle \in D(l)$, we have that there is an edge from each leaf of T_l of color s to each leaf of T_l of color t. In shrub-depth, this is additionally conditioned on whether l is the lowest common ancestor of the vertices to be connected. This means that in shrub-depth a vertex is distinguished not only by its color but also by the information which subtree of T_l has the vertex as a leaf. Simple attempts to mitigate that either increase the depth of the circuit to be logarithmic or the number of possible enforcements to be exponential in the number of vertices.

4.2 Circuit Input

The circuit $\mathbb{SCW}_{\langle n,d,k\rangle}$ inputs an n-vertex game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V, rank : V \to \mathbb{N}, Color : V \to C_{\mathrm{E}} \uplus C_{\mathrm{O}})$ as $rank$ and a tree-model $\mathcal{TM} = (T, C = C_{\mathrm{E}} \uplus C_{\mathrm{O}}, Color, D)$ such that $rank(V) \subseteq \{0, \dots d-1\}$, $d \leq 2n$, $|C| = k$, the height of T is k and $\mathcal{TM}$ induces the arena of G. The whole input is encoded as follows.

- For each vertex $v \in V$: k bits that one-hot encode its color (denoted as $COLOR_v$, one bit for one color, only the bit denoting the true color of the vertex is going to be true), and d bits that one-hot encode $rank(v)$ (denoted as $RANK_v$).
- k bits for color-to-player mapping: i'th bit is set to true if i'th color belongs to player E (denoted as $ECOLOR$).
- Description of T viewed as a subgraph of the graph consisting of $k+1$ layers, where each layer consists of n vertices, and each of them is connected to all vertices from the layer below (see Figure 3). So, for each $l \in \{1, \dots, k\}$ and $i \in \{1, \dots, n\}$:
  - n bits describing the connections to the layer below. (denoted as $CHILD_{l,i}$, for $l = 1$ it describes connections to layer 0, which is formed of vertices from V)
  - k^2 bits describing the result of D for a given internal node – one bit for each pair of colors (denoted as $DB_{l,i}$).

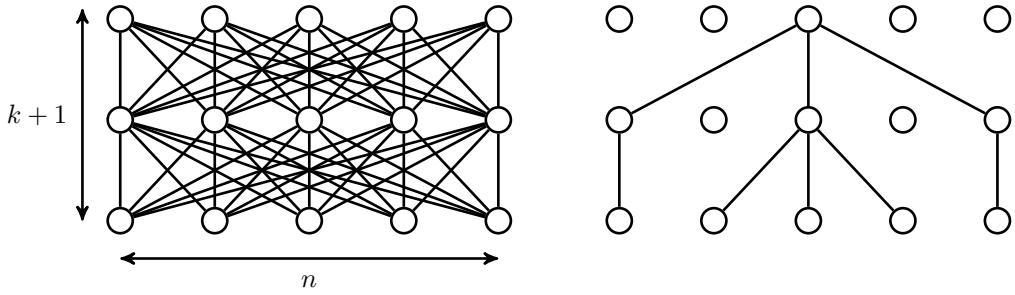


Figure 3 Tree embedding graph (left) and embedding of some tree (right).

.

We will call a distinguished collection of bits (which can be input bits and gate results) a bit pack. To denote a specific bit of a bit pack we will use square brackets notation. For example $COLOR_v[c]$ denotes the bit that is true if and only if vertex v has color c.

For validating input data, one can easily compute in space $\mathcal{O}(\log(n) + \log(k))$ parts of the circuit that have constant-depth and validate one-hot encodings. To validate T, one can compute in space $\mathcal{O}(\log(n) + \log(k))$ constant depth parts for validating that: each node has at most one parent, each node from layer 0 has exactly one parent, each node that has at least one child and is not from the last layer has a parent, each node that has a parent and is not from the layer 0 has at least one child, there is exactly one node with children in the last layer.

4.3 Gadgets

Here we define gadgets (auxiliary circuits) that will be used to construct the whole circuit.

4.3.1 Initialization

Here we define a constant-depth gadget that for a vertex v of G calculates a valid set of enforcements for v in an edgeless subgame of G that contains only one vertex v. We define the gadget so that it outputs empty sets of enforcements for other vertices of G and marks the vertex v in its output.

▶ **Definition 20.** INIT_v

$$A^{\text{out}}[w] = (w = v)$$
$$ENF_w^{\text{out}}[P] = (w = v) \wedge \bigvee_{\langle c,r\rangle : P \text{ is in up-closure of } \{[c \mapsto r]\}} COLOR_w[c] \wedge RANK_w[r]$$

Note that for every P the set $\{\langle c, r\rangle : P \text{ is in upward closure of } \{[c \mapsto r]\}\}$ can be enumerated in space $\mathcal{O}(\log(d)k)$. Note also that $\{P : P \text{ is in upward closure of } \{[c \mapsto r]\}\}$ is a valid enforcement set for a vertex v in one vertex game G where v has rank r and color c.

4.3.2 Combination Gadget

We define a constant-depth gadget that given m packs of input bits, each of length l (i'th bit of j'th pack denoted as $IN_j[i]$), and m bits that encode the choice of the bit pack (j'th bit denoted as $COND[j]$), outputs the result of the or operation on chosen bit packs.

▶ **Definition 21.** COMBINE^{m,l}

$$OUT[i] = \bigvee_{1 \le j \le m} IN_j[i] \wedge COND[j]$$

4.3.3 Update Gadget

Here we define a constant depth gadget that given for each vertex v of a subgame G' of G a valid set of enforcements outputs a valid set of enforcements for each vertex of the game G'' created from G' by adding directed edges from vertices of color s to those of color t.

Let $G' = (V' = V'_{\mathrm{E}} \uplus V'_{\mathrm{O}}, E', rank', Color')$ be any subgame of G. For a pair of colors $\langle s, t\rangle$ we define a gadget $\mathbb{U}^{\langle s,t\rangle}$ that inputs one bit $ECOLOR[s]$, and for each vertex v of G one bit encoding whether $v \in V'$ (denoted as $A^{\mathrm{in}}[v]$), $COLOR_v$, and $(d+1)^k$ bits denoted as ENF^{in}_v. For a vertex v of G', ENF^{in}_v will encode a set of enforcements that is valid for v in G' (one bit for one partial function from C to $\{0, \ldots, d-1\}$; bit for enforcement P denoted as $ENF^{\mathrm{in}}_v[P]$). We define $\mathbb{U}^{\langle s,t\rangle}$'s output to be $(d+1)^k$ bits for each vertex v of G (denoted as ENF^{out}_v). For a vertex $v \in V'$ those bits will encode enforcement set that is valid for v in the game G'' created from G' by adding connections from vertices colored s to ones colored t.

We consider two cases for a pair of colors $\langle s, t\rangle$. The first one is when color s belongs to player E (i.e. $ECOLOR[s]$ is true) and the other is when it belongs to player O. For the first case, we use gadget $\mathrm{U}^{s,t}_{\mathrm{E}}$, whereas for the second one $\mathrm{U}^{s,t}_{\mathrm{O}}$, both of which are defined below.

▶ **Definition 22.** $\mathrm{U}^{s,t}_{\mathrm{E}}$

$$MOVE[P] = \bigvee_{w} A^{\mathrm{in}}[w] \wedge COLOR_w[t] \wedge ENF^{\mathrm{in}}_w[P]$$

$$LOOPED[P] = s \notin \mathrm{Dom}(P) \wedge \bigvee_{2r \in \{0,\ldots,d-1\}} MOVE[P[s \mapsto 2r]]$$

$$OPT[P] = MOVE[P] \vee LOOPED[P]$$

$$ENF^{\mathrm{out}}_v[P] = A^{\mathrm{in}}[v] \wedge \left(\bigvee_{S \sqsubseteq P} ENF^{\mathrm{in}}_v[S] \vee \bigvee_{\langle Q,R\rangle : \mathrm{merge}(Q,s,R) \sqsubseteq P} ENF^{\mathrm{in}}_v[Q] \wedge OPT[R] \right)$$

Where $\mathrm{merge}(Q, s, R)$ *is calculated as if* s *was the color of player* E.

The intuition behind $\mathrm{U}^{s,t}_{\mathrm{E}}$ is that we should be able to modify each positional strategy ρ_{E} that is safe from v in game G'' to a strategy ρ'_{E} that behaves similarly to ρ_{E} but when ρ_{E} decides to make a move along a new edge, then ρ'_{E} always chooses to move to a fixed vertex w of color t. What is more, we should be able to make this modification in a way that $\mathrm{enf}^{G''}_{\rho'_{\mathrm{E}},v} \sqsubseteq \mathrm{enf}^{G''}_{\rho_{\mathrm{E}},v}$ and ρ'_{E} is also safe from v. In $ENF^{\mathrm{out}}_v[P]$ we check for the existence of such ρ'_{E} with $\mathrm{enf}^{G''}_{\rho'_{\mathrm{E}},v} \sqsubseteq P$. We do it by splitting ρ'_{E} into two parts: part till the new move (with enforcement $\sqsubseteq Q$) and after (with enforcement $\sqsubseteq R$). Note that ρ'_{E} defined as above will either make a move along a new edge at most once ($MOVE$) or an unbounded number of times ($LOOPED$).

▶ **Definition 23.** $\mathrm{U}^{s,t}_{\mathrm{O}}$

First, for each pair of enforcements Q, P, *and a vertex* v *such that* $Q \sqsubseteq P$*:*

$$\mathrm{SAFE}^{Q,P}_v = \neg A^{\mathrm{in}}[v] \vee \neg COLOR_v[t] \vee \bigvee_{R : \mathrm{eon}(R,s) \wedge \mathrm{merge}(Q,s,R) \sqsubseteq P} ENF^{\mathrm{in}}_v[R]$$

$$\mathrm{SAFE}^{Q,P} = \bigwedge_{w} \mathrm{SAFE}^{Q,P}_w$$

Where $\mathrm{eon}(R, s)$ *is* true *if and only if* $R(s)$ *is either even or undefined, and* $\mathrm{merge}(Q, s, R)$ *is calculated as if* s *was the color of player* O.

$$ENF_v^{\mathrm{out}}[P] = A^{\mathrm{in}}[v] \wedge \bigvee_{Q \sqsubseteq P} ENF_v^{\mathrm{in}}[Q] \wedge (s \notin \mathrm{Dom}(Q) \vee \mathrm{SAFE}^{Q,P})$$

The intuition behind $\mathrm{U}_{\mathrm{O}}^{s,t}$ is that each safe positional strategy in game G'' should factor to a strategy up to reaching a vertex of color s and a bunch of strategies after the move of player O. The part $\mathrm{SAFE}_v^{Q,P}$ corresponds to checking whether given a safe strategy with enforcement $\sqsubseteq Q$ in game G' we can make it respond to a move of player O to a vertex v in a way that maintains the safety of the strategy and makes its enforcement non-worse than P.

We merge auxiliary gadgets using $\mathrm{COMBINE}^{2,n(d+1)^k}$ with input packs being the outputs of $\mathrm{U}_{\mathrm{E}}^{s,t}$ and $\mathrm{U}_{\mathrm{O}}^{s,t}$ respectively and $COND[1] = ECOLOR[s]$, $COND[2] = \neg ECOLOR[s]$ ($\mathrm{U}_{\mathrm{E}}^{s,t}$ was created to handle the case when s is the color of player E and $\mathrm{U}_{\mathrm{O}}^{s,t}$ for the other one, here we use $\mathrm{COMBINE}^{2,n(d+1)^k}$ to choose between their results).

Note that by transitivity of $\sqsubseteq$, we have that sets ENF_v^{out} produced by $\mathrm{U}_{\mathrm{E}}^{s,t}$ and $\mathrm{U}_{\mathrm{O}}^{s,t}$ are up-closed. Note also that $\mathbb{U}^{\langle s,t \rangle}$ has constant depth and can be computed in space $\mathcal{O}(\log(n) + \log(d)k)$. This is because $\mathcal{O}(\log(n))$ bits suffice to remember a constant number of vertices. What is more, as each enforcement is a partial function from the set of k colors to the set of d ranks, so to encode an enforcement $\mathcal{O}(\log(d)k)$ bits suffice, and we consider at most four enforcements simultaneously.

▶ **Lemma 24.** *Given that* $\mathbb{U}^{\langle s,t \rangle}$ *inputs a valid set of enforcements for each vertex* v *of a subgame* G' *of game* G *then it outputs a valid set of enforcements for each vertex* v *of a game* G'' *created from* G' *by addition of directed edge from every vertex of color* s *to every vertex of color* t.

A simple case analysis suffices to prove this lemma when s belongs to player O. When s belongs to player E one can separately prove soundness (again by simple case analysis) and completeness following the presented intuition. These proofs are rather long and tedious and can be found in the full version [28].

4.3.4 Conditional Update Gadget

For each pair of colors $\langle s, t \rangle$ we define $\mathbb{U}_{cond}^{\langle s,t \rangle}$ that inputs the bits that $\mathbb{U}^{\langle s,t \rangle}$ inputs with one additional bit that is true if and only if the update defined by $\mathbb{U}^{\langle s,t \rangle}$ should be performed (i.e. whether enforcements should be updated or copied). We define $\mathbb{U}_{cond}^{\langle s,t \rangle}$ to be a simple combination of $\mathbb{U}^{\langle s,t \rangle}$ and $\mathrm{COMBINE}^{2,n(d+1)^k}$.

4.3.5 Processing Gadget

Here we define a gadget of depth $\mathcal{O}(k^2)$ that stacks operations of conditional update gadgets.

Let $p_1, \ldots, p_{k^2}$ be pairs of colors in some fixed order. We define the gadget $\mathbb{P}$ as a sequence of gadgets $\mathbb{U}_{cond}^{p_1}, \ldots \mathbb{U}_{cond}^{p_{k^2}}$ where ENF_v^{in} part of the input for gadget $\mathbb{U}_{cond}^{p_i}$ comes from the output of $\mathbb{U}_{cond}^{p_{i-1}}$ for $i > 1$ and rest of the input of $\mathbb{U}_{cond}^{p_j}$ for $j > 0$ comes from the input of $\mathbb{P}$. In particular for $\mathbb{U}_{cond}^{p_j}$ the bit that tells whether to perform the update comes from the pack of k^2 bits (denoted as DB^{in}) that $\mathbb{P}$ inputs aside from $ECOLOR$ and $COLOR_v$, ENF_v^{in}, $A^{\mathrm{in}}[v]$ for each v. We define the output of $\mathbb{P}$ to be the output of $\mathbb{U}_{cond}^{p_{k^2}}$ and $A^{\mathrm{out}}[v] = A^{\mathrm{in}}[v]$ for each v.

Note that $\mathbb{P}$ defined above has depth $\mathcal{O}(k^2)$ and can be computed in space $\mathcal{O}(\log(n) + \log(d)k)$, as each of the conditional update gadgets can be computed in this space, and we can use $\mathcal{O}(\log(k))$ bits to encode for which $\mathbb{U}^{p_j}_{cond}$ we are performing the computation.

Let l be an internal node of T from the given tree-model and let $l_1, \ldots, l_m$ be children of l. Gadget $\mathbb{P}$ will be later used to calculate valid enforcement sets for the subgame of G played on an arena induced by $\mathcal{TM}_l$ from valid enforcement sets for the subgame of G played on an union of arenas induced by $\mathcal{TM}_{l_1}, \ldots, \mathcal{TM}_{l_m}$.

4.4 Circuit Construction

The construction of the circuit will proceed in layers, with each layer output being n big packs of bits, each containing ENF^{out}_v and $A^{\text{out}}[v]$ for each vertex v, so $n(n((d+1)^k+1))$ bits in total. The output of each layer will be treated as the input for the next one. The output of l'th layer will be denoted as $LPACK_l$. To start the construction off and create layer 0, we use INIT_v gadget for each vertex v. To create parts for layer $1 \leq l \leq k$ we need to use $\mathbb{P}$ for each node $\langle l, i\rangle$. We do so by setting $\mathbb{P}$'s DB^{in} bits to $DB_{l,i}$, whereas ENF^{in}_v and $A^{\text{in}}[v]$ bits using $\text{COMBINE}^{n,n((d+1)^k+1)}$ with input being n big bit packs from $LPACK_{l-1}$ and $COND[j] = CHILD_{l,i}[j]$ (here $\text{COMBINE}^{n,n((d+1)^k+1)}$ is used to take a disjoint union of arenas). The output of $\mathbb{P}$ here constitutes to $LPACK_l$.

To produce partition of V into winning regions we first gather calculated results using $\text{COMBINE}^{n,n((d+1)^k+1)}$ gadget with input being set to n big bit packs from $LPACK_k$ and $COND[j] = \text{true}$, and then create for each vertex v gadget that checks whether there is an enforcement P such that $ENF^{\text{in}}_v[P]$ is true and for each $c \in \text{Dom}(P)$ $ECOLOR[c]$ is false.

▶ **Theorem 25.** *There exists a Turing machine that given $n, d > 1$, and $k > 0$ computes in space $\mathcal{O}(\log(n) + \log(d)k)$ circuit $\mathbb{SCW}_{\langle n,d,k\rangle}$ of depth $\mathcal{O}(k^3)$, that inputs parity game $G = (V = V_{\text{E}} \uplus V_{\text{O}}, E, rank, Color)$ and tree-model $\mathcal{TM} = (T, C = C_{\text{E}} \uplus C_{\text{O}}, Color, D)$ such that $rank(V) \subseteq \{0, \ldots, d-1\}$, $d \leq 2n$, $|C| = k$, T has height exactly k and $\mathcal{TM}$ induces G's arena, and outputs the partition of G's vertices into winning regions of players* E *and* O.

Proof. The complexity follows from the observations made during the construction. Correctness follows from Lemma 24 and simple induction on the size of the circuit. ◀

5 Circuit Family for Tree-Depth

Now we will show how for $n, d > 1$, and $k \geq 0$ compute in space $\mathcal{O}(\log(n) + \log(d)k)$ circuit $\mathbb{TD}_{\langle n,d,k\rangle}$ of depth being at most $\mathcal{O}(k^3)$, that given an n-vertex parity game with all ranks being less that d (we also require $d \leq 2n$) and elimination forest of height at most k, outputs the winner for each vertex. We achieve this by creating circuit $\mathbb{RED}_{\langle n,d,k\rangle}$ that changes input representation to the one accepted by $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$, and then getting the winners from the output of $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$ using circuit $\mathbb{GET}_{\langle n,d,k\rangle}$. Note that along with the construction of the circuit $\mathbb{RED}_{\langle n,d,k\rangle}$ we show a more efficient reduction that does not have an exponential blow-up in k (in contrast to the simple one presented in Section 2.4). Such improvement is not necessary for the main result of this paper, but it makes the construction more computationally tractable. In the end, we show how to get rid of the requirement for the elimination forest.

5.1 Circuit Input

Circuit $\mathbb{RED}_{\langle n,d,k\rangle}$ inputs a game $G = (V = V_{\mathrm{E}} \uplus V_{\mathrm{O}}, E \subseteq V \times V, rank : V \to \mathbb{N})$ and elimination forest $\mathcal{F}$ as the following sequence of bits.

- For each vertex $v \in V$:
  - d bits that one-hot encode $rank(v)$ (denoted as $RANK_v$),
  - one bit that is true if and only if v is a vertex of player E (denoted as $EB[v]$),
  - $k{+}1$ bits that one-hot encode the depth of v in $\mathcal{F}$ (denoted as $DEPTH_v$; bit $DEPTH_v[x]$ is set to true if v has depth x; root has depth 0),
  - $n-1$ bits that encode the parent of v in $\mathcal{F}$ (denoted as $PARENT_v$; $PARENT_v[w]$ is true if w is a parent of v in $\mathcal{F}$; each non-root vertex has exactly one parent).
- For each pair of vertices $v, w \in V$, a bit that is set to true if and only if there is a directed edge from v to w (denoted as $EDG[v, w]$).

Checking whether $\mathcal{F}$ is a valid forest can be done similarly as in Section 4.2. For ancestor-descendant relationship one can compute in space $\mathcal{O}(\log(n) + \log(k))$ a gadget of depth $\mathcal{O}(\log(k))$ that at level i checks for every $v, w \in V$ whether there is a simple path from v to w that goes towards the root and has length at most 2^i.

5.2 Reduction Circuit

First of all, we do not aim to create exactly the same game as input for $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$, but a modified game from which we could deduce winning regions in the original one. The main idea here is to split each vertex in a number of vertices such that when we merge them back, then we get the original arena. Let $V = \{v_1, \ldots, v_n\}$ be all vertices of G, with each vertex $v \in V$ we associate $2n$ copies of v with the same player ownership: $v \to v^{\circ}_{v_1}, \ldots, v^{\circ}_{v_n}, v^{\rightarrow}_{v_1}, \ldots, v^{\rightarrow}_{v_n}$. The first n of these copies are denoted as v°, whereas the last n as $v^{\rightarrow}$. Each of them corresponds to one additional vertex from the input of $\mathbb{RED}_{\langle n,d,k\rangle}$ (the copy among the first n copies of v that corresponds to w is denoted as v°_w, whereas the copy among the last n copies that corresponds to w is denoted as $v^{\rightarrow}_w$). Intuitively for a vertex $v \in V$, copies from v° and $v^{\rightarrow}$ will be used to distribute the original connections from and to v. More precisely, vertices from v° will allow to choose a connection from v, whereas vertices from $v^{\rightarrow}$ to execute it, and if there will be a connection from v to w in G, then we will create a connection from some $v^{\rightarrow}_{v'}$ to some $w^{\circ}_{w'}$.

We set the rank of copies from $v^{\rightarrow}$ to $rank(v) + 2$, whereas the rank of copies from v° to 1 if v belongs to player E, and to 0 if v belongs to player O (as v° will be used to choose a connection we don't want a player to put off the decision indefinitely). Let x be the depth of v in $\mathcal{F}$. We color copies v° using color $\langle \circ, EB[v], x\rangle$, whereas copies from $v^{\rightarrow}$ using $\langle \rightarrow, EB[v], x\rangle$. The copies created here are leaves of the tree-model that $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$ will take as input. Note that for all operations presented above we can compute in space $\mathcal{O}(\log(n) + \log(k))$ gadget of constant depth, as we need $\mathcal{O}(\log(n))$ bits to represent a copy of a vertex and rank and $\mathcal{O}(\log(k))$ to represent a color.

Before we go any further, we introduce some gadgets. For simplicity, we will assume that vertex is its own ancestor.

- $\mathrm{CHILDLESS}_v$ – that is true if and only if v has no children in $\mathcal{F}$.
- ANEQ^w_v – that is true if and only if w is an ancestor of v in $\mathcal{F}$. Note that it can be done by a gadget with $\mathcal{O}(\log(k))$ layers computable in space $\mathcal{O}(\log(n) + \log(k))$ that in layer i considers paths from vertices towards root of length at most 2^i.
- $\mathrm{ANEQC}^{x \to x'}_v$ – that is true if and only there are two ancestor w and w' of v of depth x and x' respectively such that there is an an edge from w to w'.

Now we will describe how we compute the part of $\mathbb{RED}_{\langle n,d,k\rangle}$ that outputs the rest of the tree-model. The main thing that we will do here is that for each vertex v that is a leaf in the given elimination forest we will gather copies of ancestors that correspond to v, add connections using the copies and join the copies along the way creating a tree-model similar in shape to the given elimination forest. The computation will consist of $k+1$ major steps, and $3k+3$ simple steps. Major steps will create layers of the tree-model (see Section 4.2; layers are numbered starting from the layer of leaves, which has number 0), whereas simple steps will extend the height of the tree-model to match the specification for $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$. As in the description of the tree-model for $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$ we use graph where number of nodes in every layer is the same and in our case equals number of copies of vertices, so we will use copies of vertices to number nodes from a specific layer. The tree-model that we will produce here may not use all of the leaves (there may be vertices at the bottom that will be left isolated), we get around it by relaxing the correctness of $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$ input so that it will allow vertices from layer 0 without parents (note that they will be ignored in the further parts of the circuit). As the steps presented below are rather technical we encourage to refer to Figure 4.

First the major steps, for $l=1$, for each vertex v of the input game that is childless in $\mathcal{F}$, we gather and initialize copies of ancestors of v corresponding to v. That is for each $v,w\in V$, $p\in\{\text{true},\text{false}\}$ and $x\in\{0,\dots,k\}$ we set

$$\begin{aligned} \mathit{CHILD}_{l,v^\circ_v}[w^\circ_v] &= \mathrm{ANEQ}^w_v \wedge \mathrm{CHILDLESS}_v \\ \mathit{CHILD}_{l,v^\circ_v}[w^\rightarrow_v] &= \mathrm{ANEQ}^w_v \wedge \mathrm{CHILDLESS}_v \\ \mathit{DB}_{l,v^\circ_v}[\langle\langle\circ,p,x\rangle,\langle\rightarrow,p,x\rangle\rangle] &= \mathrm{CHILDLESS}_v \end{aligned}$$

Then we add all connections between gathered copies, that is for $v\in V$, $p,p'\in\{\text{true},\text{false}\}$ and $x,x'\in\{0,\dots,k\}$ we set

$$\mathit{DB}_{l,v^\circ_v}[\langle\langle\rightarrow,p,x\rangle,\langle\circ,p',x'\rangle\rangle] = \mathrm{CHILDLESS}_v \wedge \mathrm{ANEQC}^{x\to x'}_v$$

Now let $1<l\leq k+1$. Here we create parts that drag low depth vertices up in the tree-model and merge gathered copies. That is for $p,p'\in\{\text{true},\text{false}\}$, $x\in\{0,\dots,k-l+1\}$, $w\neq v$ we set:

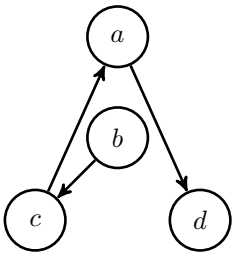
$$\begin{aligned} \mathit{CHILD}_{l,v^\circ_v}[v^\circ_v] &= \mathrm{CHILDLESS}_v \wedge \bigvee_{t\in\{0,\dots,k-l+1\}} \mathit{DEPTH}_v[t] \\ \mathit{CHILD}_{l,v^\circ_v}[w^\circ_w] &= \mathit{DEPTH}_v[k-l+1]\wedge\mathit{PARENT}_w[v] \\ \mathit{DB}_{l,v^\circ_v}[\langle\langle\circ,p,k-l+1\rangle,\langle\circ,p,k-l+1\rangle\rangle] &= \mathit{DEPTH}_v[k-l+1] \end{aligned}$$

For simple steps, we extend the height of the tree-model to match specification for $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$. Let v be some fixed vertex in V and w any vertex from V, and let $k+3\leq l\leq 4k+4$, we set $\mathit{CHILD}_{k+2,v^\circ_v}[w^\circ_w]$ to $\mathit{DEPTH}_w[0]$ whereas $\mathit{CHILD}_{l,v^\circ_v}[v^\circ_v]$ to true.

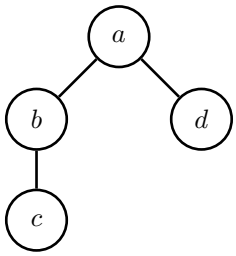
In both major and simple steps, we set unmentioned bits to false.

▶ **Proposition 26.** *The depth of $\mathbb{RED}_{\langle n,d,k\rangle}$ when constructed as above is $\mathcal{O}(\log(k))$, what is more the construction can be done in space $\mathcal{O}(\log(n)+\log(k))$.*

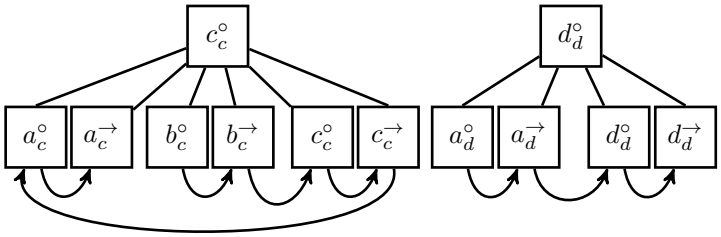
Proof. Observe that $\mathit{CHILD}_{l,v^\circ_v}$ and $\mathit{DB}_{l,v^\circ_v}$ bits depend only on bits describing depth and ancestors. What is more, $\mathcal{O}(\log(n)+\log(k))$ bits suffice to describe which step l we are performing and for which bit of either $\mathit{CHILD}_{l,v^\circ_v}$ or $\mathit{DB}_{l,v^\circ_v}$ we are creating a gate. ◀



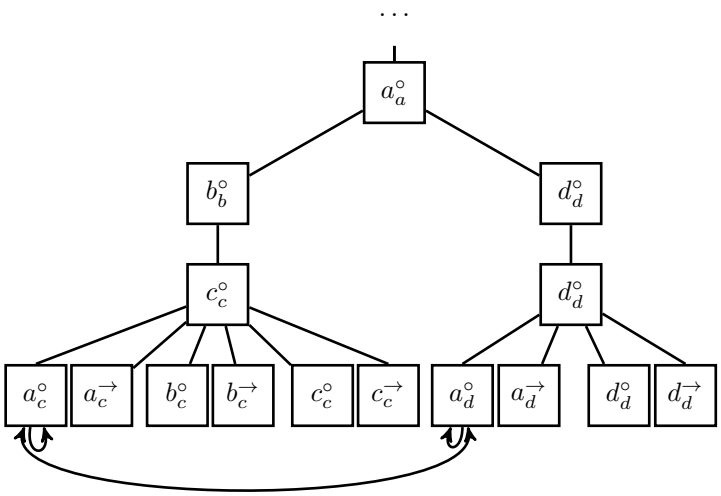
(a) Arena.



(b) Elimination forest.



(c) Layers 0 and 1 from tree-model and connections added at major step 1 (bottom).



(d) Layers $0, 1, 2, 3$ from tree-model and merge performed at major step 3 (bottom).

Figure 4 Tree-model produced for a simple arena (a) with elimination forest (b).

5.3 Output Circuit

Now we will proceed with the definition of $\mathbb{GET}_{\langle n,d,k\rangle}$. To get the winning region of E we need to check for each vertex $v \in V_{\mathrm{E}}$ whether there is a copy v_w° such that E wins from v_w° and for each $v \in V_{\mathrm{O}}$ whether E wins from all copies of v. It suffices as note that we have constructed input for $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$ such that when we contract all v° and $v^\rightarrow$ into one vertex of rank $rank(v)$ then we get the original game.

$$OUT^{\mathbb{GET}}[v] = (EB[v] \wedge \bigvee_{w \in V} OUT[v_w^\circ]) \vee (\neg EB[v] \wedge \bigwedge_{w \in V} OUT[v_w^\circ])$$

where OUT denotes the output bits of $\mathbb{SCW}_{\langle 2n^2,d+2,4k+4\rangle}$. Note that $\mathbb{GET}_{\langle n,d,k\rangle}$ can be computed in space $\mathcal{O}(\log(n))$ as $\mathcal{O}(\log(n))$ bits suffice to handle a constant number of vertices.

▶ **Theorem 27.** *There exists a Turing machine that given $n, d > 1$, and $k \geq 0$ computes in space $\mathcal{O}(\log(n) + \log(d)k)$ circuit $\mathbb{TD}_{\langle n,d,k\rangle}$ of depth $\mathcal{O}(k^3)$, that inputs n-vertex parity game G along with elimination forest of G's arena, and outputs the partition of G's vertices into winning regions of players* E *and* O*.*

Proof. Follows from Proposition 26 and Theorem 25. ◀

5.4 Computing Elimination Forest

The following is an easy consequence of the result from [11, Theorem 1].

▶ **Lemma 28.** *For any $k \geq 0$ there exists a logspace uniform family of AC^0-circuits that inputs a graph and checks whether its tree-depth is k.*

Now observe that we can get an elimination forest for an arena of tree-depth at most k by a circuit that forgets about the orientation of edges and performs k steps. At step $i \in \{1, \ldots, k\}$ it considers a graph of tree-depth at most $k - i + 1$ and checks which vertices should be removed to make it a graph of tree-depth at most $k - i$. It does so by taking for each vertex v its connected component, removing v from it and checking the tree-depth of the component with v removed using a circuit from Lemma 28. In case there are many candidates the circuit picks the first one. From Lemma 28 and Lemma 5, for fixed k this circuit can have constant depth and be computed in logarithmic space (to compute connected components it suffices to check for paths of length at most $2^{k+1} - 2$ what can be done similarly as in ANEQ_v^w). So from the construction above and Theorem 27 we have the following.

▶ **Corollary 29.** *For any $k \geq 0$ there exists a logspace uniform family of AC^0-circuits that solves parity games played on arenas of tree-depth at most k.*

6 Conclusions and Further Work

We have shown that solving parity games played on arenas of bounded tree-depth or shallow clique-width is in logspace uniform AC^0, assuming that the suitable tree-model is provided for the latter case. Speaking of uniformity, one can assign appropriate binary identifiers to gates such that for any k, the problem: "Given circuit parameters n, d in binary and binary identifiers of two gates A, B decide whether in $\mathbb{TD}_{\langle n,d,k\rangle}$ both A and B are present, and the output of B constitutes to the input of A" can be solved by a deterministic Turing machine that works in time polylogarithmic in n and uses space logarithmic in n. However, going lower seems to require more technical care. A natural next step in research would be to decide whether one can obtain a similar result for the bounded shrub-depth case. However, as we observed in Section 4.1, such step seems to require a different approach.

References

1 Miklós Ajtai. $\sum^1_1$-formulae on finite structures. *Ann. Pure Appl. Log.*, 24(1):1–48, 1983. doi:10.1016/0168-0072(83)90038-6.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009. URL: http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264.

3 Dietmar Berwanger, Anuj Dawar, Paul Hunter, and Stephan Kreutzer. Dag-width and parity games. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 524–536. Springer, 2006. doi:10.1007/11672142_43.

4 Dietmar Berwanger and Erich Grädel. Entanglement – A measure for the complexity of directed graphs with applications to logic and games. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Montevideo, Uruguay, March 14-18, 2005, Proceedings*, volume 3452 of *Lecture Notes in Computer Science*, pages 209–223. Springer, 2004. doi:10.1007/978-3-540-32275-7_15.

5 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263. ACM, 2017. doi:10.1145/3055399.3055409.

6 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993. doi:10.1016/0022-0000(93)90004-G.

7 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

8 Anuj Dawar and Erich Grädel. The descriptive complexity of parity games. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, volume 5213 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2008. doi:10.1007/978-3-540-87531-4_26.

9 Reinhard Diestel. *Graph Theory*. Springer Publishing Company, Incorporated, 5th edition, 2017. doi:10.1007/978-3-662-53622-3.

10 Michael Elberfeld, Martin Grohe, and Till Tantau. Where first-order and monadic second-order logic coincide. *ACM Trans. Comput. Log.*, 17(4):25, 2016. doi:10.1145/2946799.

11 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th – March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 66–77. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.66.

12 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185392.

13 John Fearnley and Sven Schewe. Time and parallelizability results for parity games with bounded treewidth. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 189–200. Springer, 2012. doi:10.1007/978-3-642-31585-5_20.

14 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.

15 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Z. Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation – 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8_15.

16 Moses Ganardi. Parity games of bounded tree- and clique-width. In Andrew M. Pitts, editor, *Foundations of Software Science and Computation Structures – 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 390–404. Springer, 2015. doi:10.1007/978-3-662-46678-0_25.

17 Robert Ganian, Petr Hlinený, Joachim Kneis, Alexander Langer, Jan Obdrzálek, and Peter Rossmanith. Digraph width measures in parameterized algorithmics. *Discret. Appl. Math.*, 168:88–107, 2014. doi:10.1016/j.dam.2013.10.038.

18 Robert Ganian, Petr Hlinený, Jaroslav Nesetril, Jan Obdrzálek, and Patrice Ossona de Mendez. Shrub-depth: Capturing height of dense graphs. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:7)2019.

19 Robert Ganian, Petr Hlinený, Jaroslav Nesetril, Jan Obdrzálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast MSO1. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 – 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2012. doi:10.1007/978-3-642-32589-2_38.

20 Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theor. Comput. Sci.*, 399(3):206–219, 2008. doi:10.1016/j.tcs.2008.02.038.

21 Marcin Jurdzinski and Ranko Lazic. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–9. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005092.

22 Karoliina Lehtinen, Pawel Parys, Sven Schewe, and Dominik Wojtczak. A recursive approach to solving parity games in quasipolynomial time. *Log. Methods Comput. Sci.*, 18(1), 2022. doi:10.46298/lmcs-18(1:8)2022.

23 Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. Practical synthesis of reactive systems from ltl specifications via parity games. *Acta Informatica*, 57(1–2):3–36, November 2019. doi:10.1007/s00236-019-00349-3.

24 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

25 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

26 Jan Obdrzálek. Fast mu-calculus model checking when tree-width is bounded. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 80–92. Springer, 2003. doi:10.1007/978-3-540-45069-6_7.

27 Jan Obdrzálek. Clique-width and parity games. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2007. doi:10.1007/978-3-540-74915-8_8.

28 Konrad Staniszewski. Parity games of bounded tree-depth, 2022. doi:10.48550/arXiv.2211.02926.

Tower-Complete Problems in Contraction-Free Substructural Logics

Hiromi Tanaka
Keio University, Tokyo, Japan

Abstract

We investigate the non-elementary computational complexity of a family of substructural logics without contraction. With the aid of the technique pioneered by Lazić and Schmitz (2015), we show that the deducibility problem for full Lambek calculus with exchange and weakening ($\mathbf{FL_{ew}}$) is not in Elementary (i.e., the class of decision problems that can be decided in time bounded by an elementary recursive function), but is in PR (i.e., the class of decision problems that can be decided in time bounded by a primitive recursive function). More precisely, we show that this problem is complete for Tower, which is a non-elementary complexity class forming a part of the fast-growing complexity hierarchy introduced by Schmitz (2016). The same complexity result holds even for deducibility in BCK-logic, i.e., the implicational fragment of $\mathbf{FL_{ew}}$. We furthermore show the Tower-completeness of the provability problem for elementary affine logic, which was proved to be decidable by Dal Lago and Martini (2004).

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Complexity theory and logic; Theory of computation → Proof theory

Keywords and phrases substructural logic, linear logic, full Lambek calculus, BCK-logic, computational complexity, provability, deducibility

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.34

Related Version *Full Version*: https://arxiv.org/abs/2201.02496

Funding This work was partially supported by JST, CREST Grant Number JPMJCR2114, Japan.

Acknowledgements I would like to thank Koji Mineshima for helpful comments on the introduction section. I am also grateful to the anonymous reviewers for useful suggestions, which improve some proofs and deepen the discussion in Section 7.

1 Introduction

The term "substructural logic" [14, 31, 34] is an umbrella term for a family of logics that limit the use of some of the structural rules. Substructural logics encompass a wide range of non-classical logics (e.g., intuitionistic, classical, relevance, paraconsistent and multi-valued logics), and thus are discussed in several distinct areas close to mathematical logic such as philosophy, linguistics and computer science. In any of those research fields, one of the major topics is to settle the computational complexity of the *provability problem* for a logic, i.e., the problem of whether a given formula is provable in the logic. There are many seminal papers concerning this subject; see, e.g., [3, 21, 26, 28, 33, 36, 37, 41].

It is no surprise that a more general problem can be considered for a given logic. The *deducibility problem* for the logic asks for a given finite set Φ of formulas and a given formula A whether A is provable in the logic augmented with Φ as a set of non-logical axioms. In the setting of classical and intuitionistic logic, the notion of deducibility is reduced to provability via the deduction theorem. As a result, the deducibility problem for intuitionistic (resp. classical) propositional logic is complete for Pspace (resp. coNP) as with the provability problem. On the other hand, since most of substructural logics do not admit the deduction theorem, there is no guarantee that these two problems are inter-reducible to each other. For

this reason, it is important to distinguish them in the framework of substructural logic. In fact, some substructural logics have a critical "gap" between the complexity of provability and the complexity of deducibility. The so-called *Lambek calculus* is a striking example of such logics. Buszkowski [6] showed that its deducibility is undecidable, but later on Pentus [33] proved that its provability is NP-complete.

Main contribution

This paper aims at clarifying the non-elementary computational complexity of deducibility in contraction-free substructural logics. So far, such a topic has not been sufficiently explored while some earlier papers investigated a non-primitive recursive complexity of weakening-free substructural logics (i.e., relevance logics); see [26, 36, 41].

Full Lambek calculus with exchange and weakening ($\mathbf{FL_{ew}}$), i.e., intuitionistic logic without the rule of contraction, is one of the most basic contraction-free logics. The deducibility problem for this logic is known to be decidable, thanks to the finite embeddability property of FL_{ew}-algebras shown by Blok and van Alten [4, 5]. However, its exact complexity remained open. Hence the following natural questions arise:

- *Is there a primitive recursive algorithm – i.e., one whose runtime is bounded by a primitive recursive function – for the deducibility problem for* $\mathbf{FL_{ew}}$*?*
- *If so, is there an elementary recursive algorithm – i.e., one whose runtime is bounded by a tower of exponentials of fixed height – for the problem?*

We answer "yes" to the first question, but provide a negative answer to the second question. To be precise, we show that the problem is actually complete for the class Tower (Corollary 23). This class forms a part of the *fast-growing complexity hierarchy* introduced by Schmitz [35], and roughly speaking, is located between Elementary (i.e., the class of problems decidable in elementary time) and PR (i.e., the class of problems that can be solved in time bounded by a primitive recursive function). As a consequence, it turns out that the aforementioned "gap" also lies between provability and deducibility in $\mathbf{FL_{ew}}$; the provability problem for $\mathbf{FL_{ew}}$ is Pspace-complete, cf. [21].

We stress that the same holds even when almost all the logical connectives are removed from $\mathbf{FL_{ew}}$. We also prove that the deducibility problem for *BCK-logic* [22, 32], i.e., the implicational fragment of $\mathbf{FL_{ew}}$, is Tower-complete (Corollary 23). This is in sharp contrast to the NP-completeness of provability in BCK-logic (Corollary 3).

Proof overview

To show the Tower-membership of deducibility in $\mathbf{FL_{ew}}$, we prove that there are reductions:

(1) from deducibility in $\mathbf{FL_{ew}}$ to provability in a variant of intuitionistic affine logic (denoted by $\mathbf{ILZW'}$),

(2) from the provability problem for $\mathbf{ILZW'}$ to the lossy reachability problem for *alternating branching vector addition systems with states* (ABVASS, for short).

The first reduction is quite similar to the one used in the famed proof of the undecidability of propositional linear logic by Lincoln, Mitchell, Scedrov and Shankar [28]. The second reduction is substantially inspired by Lazić and Schmitz [26]. Due to the Tower-completeness of lossy reachability in ABVASS, shown in [26], we obtain the membership in Tower of deducibility in $\mathbf{FL_{ew}}$.

In order to show the Tower-hardness, we introduce the notion of !-*prenex implicational sequent*. It is a slight modification of !-prenex sequents which Terui introduced in his PhD thesis [38]. We prove the Tower-hardness of a restricted version of the provability problem

for intuitionistic affine logic, which asks whether a given !-prenex implicational sequent is provable in intuitionistic affine logic. We obtain the desired result by showing that this problem can be reduced into deducibility in $\mathbf{FL_{ew}}$.

Provability (type inhabitation) in elementary affine logic

As a by-product resulting from our methods, we provide the precise complexity of provability (not of deducibility) in *propositional elementary affine logic* [1]. Its name comes from the fact that it characterizes elementary recursive computation in the paradigm of proofs-as-programs; see also [11, 17]. Although this logic is seemingly just an extension of BCK-logic (and $\mathbf{FL_{ew}}$) by a sort of modal storage operator, it is exploited for a variety of purposes, e.g., to characterize the class P and the exponential time hierarchy [2], to formulate a consistent naive set theory with a rich computational power [39].

In most situations, elementary affine logic is treated as a type system rather than a purely logical system. Accordingly, as with many other type systems, some decision problems can be considered, i.e., typability, type checking and type inhabitation (provability). For instance, Coppola and Martini [8] showed the decidability of typability in the $\{\multimap, !\}$-fragment of intuitionistic elementary affine logic.

On the other hand, of particular interest to us is the provability problem for elementary affine logic. Dal Lago and Martini [10] showed that provability in a classical variant of elementary affine logic is decidable. However, there are no known upper and lower bounds for that problem. We refine and extend the existing decidability result by showing the TOWER-completeness of some variants of elementary affine logic (Section 6.2). It should be noted that such a non-elementary aspect of elementary affine logic does not conflict with its elementary character that comes from the proofs-as-programs correspondence.

Organization of this paper

In the next section, we review various contraction-free logics in a step-by-step manner, and define a useful translation from classical affine logic into intuitionistic affine logic. A large portion of Sections 3 and 4 is taken from [26, Section 3]. In Section 3, we summarize some basic notions involved in ABVASSs. Section 4 is devoted to a short discussion about the existing complexity results which are crucial in proving the main claims in Sections 5 and 6. We prove the main results in Sections 5 and 6. In Section 7, we conclude the paper with some remarks on the complexity status of other substructural logics.

Proofs omitted due to space limitations appear in the full version of this paper (`https://arxiv.org/abs/2201.02496`).

2 Contraction-free substructural logics

2.1 Sequent calculi for contraction-free substructural logics

For convenience, we start with the formal definition of a sequent calculus for *intuitionistic affine logic with bottom*, denoted by **ILZW**. It is merely the extension of Troelstra's **ILZ** by the rule of left-weakening, cf. [26, 40]. The *language* $\mathcal{L}$ of **ILZW** contains logical connectives $\&, \oplus, \otimes, \multimap$ of arity 2, $!$ of arity 1, and $\mathbf{1}, \top, \bot, \mathbf{0}$ of arity 0. We fix a countable set of propositional variables $V = \{p, q, r, \ldots\}$. An *intuitionistic $\mathcal{L}$-formula* is built from propositional variables using connectives in $\mathcal{L}$. For brevity, parentheses in formulas are omitted when confusion is unlikely. Throughout this paper, metavariables $A, B, C, \ldots$ range over formulas and $\Gamma, \Delta, \Sigma, \ldots$ over finite multisets of formulas. By abuse of notation, we

$$\frac{}{A \vdash A}\ (\text{Init}) \qquad \frac{}{\vdash \mathbf{1}}\ (\mathbf{1}\text{R}) \qquad \frac{}{\bot \vdash}\ (\bot\text{L})$$

$$\frac{}{\Gamma \vdash \top}\ (\top\text{R}) \qquad \frac{}{\mathbf{0}, \Gamma \vdash \Pi}\ (\mathbf{0}\text{L}) \qquad \frac{\Gamma \vdash A \qquad A, \Delta \vdash \Pi}{\Gamma, \Delta \vdash \Pi}\ (\text{Cut})$$

$$\frac{\Gamma \vdash \Pi}{\mathbf{1}, \Gamma \vdash \Pi}\ (\mathbf{1}\text{L}) \qquad \frac{\Gamma \vdash}{\Gamma \vdash \bot}\ (\bot\text{R}) \qquad \frac{\Gamma \vdash A \qquad B, \Delta \vdash \Pi}{A \multimap B, \Gamma, \Delta \vdash \Pi}\ (\multimap\text{L})$$

$$\frac{A, \Gamma \vdash B}{\Gamma \vdash A \multimap B}\ (\multimap\text{R}) \qquad \frac{A, B, \Gamma \vdash \Pi}{A \otimes B, \Gamma \vdash \Pi}\ (\otimes\text{L}) \qquad \frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ (\otimes\text{R})$$

$$\frac{A, \Gamma \vdash \Pi}{A \,\&\, B, \Gamma \vdash \Pi}\ (\&\text{L1}) \qquad \frac{B, \Gamma \vdash \Pi}{A \,\&\, B, \Gamma \vdash \Pi}\ (\&\text{L2}) \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\ (\&\text{R})$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\ (\oplus\text{R1}) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}\ (\oplus\text{R2}) \qquad \frac{A, \Gamma \vdash \Pi \qquad B, \Gamma \vdash \Pi}{A \oplus B, \Gamma \vdash \Pi}\ (\oplus\text{L})$$

$$\frac{A, \Gamma \vdash \Pi}{!A, \Gamma \vdash \Pi}\ (!\text{D}) \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A}\ (!\text{P}) \qquad \frac{\Gamma \vdash \Pi}{A, \Gamma \vdash \Pi}\ (\text{W}) \qquad \frac{!A, !A, \Gamma \vdash \Pi}{!A, \Gamma \vdash \Pi}\ (!\text{C})$$

Figure 1 Inference rules of **ILZW**; A, B range over intuitionistic $\mathcal{L}$-formulas and Γ, Δ range over finite multisets of intuitionistic $\mathcal{L}$-formulas, and Π ranges over stoups.

simply write A for the singleton of a formula A. The multiset sum of Γ and Δ is denoted by Γ, Δ. We write $!\Gamma$ for the multiset obtained by prefixing each formula in Γ with exactly one $!$. An *intuitionistic $\mathcal{L}$-sequent* is an expression of the form $\Gamma \vdash \Pi$, where Γ is a finite multiset of intuitionistic $\mathcal{L}$-formulas, and Π is a stoup, i.e., either an intuitionistic $\mathcal{L}$-formula or the empty multiset ε. We always denote an intuitionistic $\mathcal{L}$-sequent of the form $\varepsilon \vdash \Pi$ (resp. $\Gamma \vdash \varepsilon$) by $\vdash \Pi$ (resp. $\Gamma \vdash$). The sequent calculus for **ILZW** consists of the inference rules depicted in Figure 1. A *proof* of a sequent $\Gamma \vdash \Pi$ in **ILZW** is defined in the usual manner. We furthermore define another variant of intuitionistic affine logic by adding the following *right-weakening* rule (W') to **ILZW**:

$$\frac{\Gamma \vdash}{\Gamma \vdash A}\ (\text{W'})$$

The resulting system is denoted by $\mathbf{ILZW}'$.

Let $\mathcal{K}$ be a non-empty subset of $\mathcal{L}$. An *intuitionistic $\mathcal{K}$-formula* is an intuitionistic formula containing only logical connectives from $\mathcal{K}$. An *intuitionistic $\mathcal{K}$-sequent* is an intuitionistic sequent consisting only of intuitionistic $\mathcal{K}$-formulas. The *$\mathcal{K}$-fragment* of **ILZW** (resp. $\mathbf{ILZW}'$) is the sequent calculus obtained from **ILZW** (resp. $\mathbf{ILZW}'$) by dropping all the inference rules concerning connectives not in $\mathcal{K}$.

Each of the logical systems within the scope of this paper is a fragment of **ILZW** or $\mathbf{ILZW}'$. We list them below:

- *Full Lambek calculus with exchange and left-weakening* ($\mathbf{FL_{ei}}$) is the $\{\otimes, \multimap, \&, \oplus, \mathbf{1}, \bot\}$-fragment of **ILZW**.
- *Full Lambek calculus with exchange and weakening* ($\mathbf{FL_{ew}}$) is the $\{\otimes, \multimap, \&, \oplus, \mathbf{1}, \bot\}$-fragment of $\mathbf{ILZW}'$.
- The *positive fragment* ($\mathbf{FL_{ei}^{+}}$) of $\mathbf{FL_{ei}}$ is the $\{\otimes, \multimap, \&, \oplus, \mathbf{1}\}$-fragment of **ILZW**.
- *BCK logic* (**BCK**) is the implicational fragment (i.e., $\{\multimap\}$-fragment) of **ILZW**.

Unfortunately, our notation is considerably different from the notation widely employed in the substructural logic community; we refer the reader to [14, Table 2.1] for the notational correspondence between linear logic and substructural logic.

$$\frac{}{\vdash A, A^{\perp}}\ (\text{Init}) \qquad \frac{}{\vdash \mathbf{1}}\ (\mathbf{1}) \qquad \frac{}{\vdash \Gamma, \top}\ (\top) \qquad \frac{\vdash \Gamma, A \qquad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta}\ (\text{Cut})$$

$$\frac{\vdash \Gamma}{\vdash \Gamma, \perp}\ (\perp) \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\ (\otimes) \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\ (\parr)$$

$$\frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\ (\&) \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\ (\oplus 1) \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\ (\oplus 2)$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\ (?) \qquad \frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\ (!) \qquad \frac{\vdash \Gamma}{\vdash \Gamma, A}\ (\text{W}) \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\ (?\text{C})$$

Figure 2 Inference rules of **LLW**; metavariables A, B range over classical $\mathcal{L}_C$-formulas, and Γ, Δ over finite multisets of $\mathcal{L}_C$-formulas.

We next formulate right-hand sided sequent calculi for *classical affine logic* (**LLW**), cf. [16, 24]. For our purpose here, we again employ the countable set V of propositional variables, and introduce their duals $V^{\perp} = \{p^{\perp}, q^{\perp}, r^{\perp}, \ldots\}$. Elements in $V \cup V^{\perp}$ are often referred to as *literals*. The language $\mathcal{L}_C$ consists of binary operation symbols $\&, \oplus, \otimes, \parr$, and constants $\mathbf{1}, \top, \perp, \mathbf{0}$, and unary operation symbols $!, ?$. Given a sublanguage $\mathcal{K}$ of $\mathcal{L}_C$, *Classical $\mathcal{K}$-formulas* are built up from literals using operation symbols in $\mathcal{K}$. For each classical $\mathcal{L}_C$-formula A, we inductively define the formula $A^{\perp}$ by the de Morgan duality; $(p^{\perp})^{\perp} = p, (A \,\&\, B)^{\perp} = A^{\perp} \oplus B^{\perp}, (A \oplus B)^{\perp} = A^{\perp} \,\&\, B^{\perp}, (A \otimes B)^{\perp} = A^{\perp} \parr B^{\perp}, (A \parr B)^{\perp} = A^{\perp} \otimes B^{\perp}, (!A)^{\perp} = ?A^{\perp}, (?A)^{\perp} = !A^{\perp}, \mathbf{1}^{\perp} = \perp, \perp^{\perp} = \mathbf{1}, \top^{\perp} = \mathbf{0}, \mathbf{0}^{\perp} = \top$. It is easy to see that $A = A^{\perp\perp}$ for any classical $\mathcal{L}_C$-formula A. A *classical $\mathcal{K}$-sequent* is an expression of the form $\vdash \Gamma$, where Γ is a finite multiset of classical $\mathcal{K}$-formulas. The inference rules of **LLW** are presented in Figure 2.

As with the intuitionistic sequent systems discussed earlier, various fragments of **LLW** can be defined in the usual manner. Among such fragments, of importance to us is the $\{\otimes, \parr, \&, \oplus, \mathbf{1}, \perp\}$-fragment of **LLW**, called *involutive full Lambek calculus with exchange and weakening* ($\mathbf{InFL_{ew}}$).

In [17] Girard proposed a logic which captures elementary recursive computation, called *elementary linear logic* (**ELL**). We review here some affine variants of **ELL**, following [2, 8, 10]. *Intuitionistic elementary affine logic with bottom* (**IEZW**) is obtained from **ILZW** by dropping the rules of (!D) and (!P) and by adding the following functorial promotion rule:

$$\frac{\Gamma \vdash A}{!\Gamma \vdash !A}\ (!\text{F})$$

Similarly, *classical elementary affine logic* (**ELLW**) is obtained from **LLW** by deleting the rules of (?) and (!) and by adding the following rule:

$$\frac{\vdash \Gamma, A}{\vdash ?\Gamma, !A}\ (\text{F})$$

It is easy to see that **IEZW** (resp. **ELLW**) is a subsystem of **ILZW** (resp. **LLW**), i.e., every sequent that is provable in **IEZW** (resp. **ELLW**) is provable in **ILZW** (resp. **LLW**). Henceforce, we write $\mathcal{L}^{+}$ for the language $\mathcal{L} \setminus \{\perp\}$. The notation **ILLW** (resp. **IELW**) is used to denote the $\mathcal{L}^{+}$-fragment of **ILZW** (resp. **IEZW**). Every sequent in such $\perp$-free logical systems is of the form $\Gamma \vdash A$.

Let **L** be one of the sequent calculi described so far and Φ a set of formulas in **L**. We write $\mathbf{L}[\Phi]$ for the sequent calculus obtained from **L** by adding $\vdash B$ as an initial sequent for every $B \in \Phi$. In this paper, we consider the following two types of decision problems for **L**.

▶ **Problem** (Provability in $\mathbf{L}$).
Instance: *A formula F in $\mathbf{L}$.*
Question: *Is the sequent $\vdash F$ provable in $\mathbf{L}$?*

▶ **Problem** (Deducibility in $\mathbf{L}$).
Instance: *A finite set $\Phi \cup \{F\}$ of formulas in $\mathbf{L}$.*
Question: *Is the sequent $\vdash F$ provable in $\mathbf{L}[\Phi]$?*

Our argument depends heavily on the following cut-elimination theorem, as we will see in the remaining sections:

▶ **Theorem 1** (cf. [10, 24, 30]). *The sequent calculi for* **ILLW**, **ILZW**, **ILZW′**, **LLW**, **IELW**, **IEZW**, *and* **ELLW** *all enjoy cut-elimination.*

As far as we know, for instance, the cut-elimination for **ILZW′** has not been settled. The reader can however show this without great difficulty, using a proof-theoretic or algebraic manner; see also [28, 29, 40] for technical details on cut-elimination in linear logic. We thus omit the proof in this paper.

Of course, the cut-elimination theorem also holds for various fragments of the systems stated in Theorem 1, e.g., **BCK**, $\mathbf{FL_{ei}}$, the $\{\multimap, !\}$-fragment of **ILZW**.

2.2 Translation from classical affine logic to intuitionistic affine logic

In this subsection, we present an efficient (i.e., polynomial-time) translation from **LLW** (resp. **ELLW**) into **ILLW** (resp. **IELW**). It is a modification of Laurent's *parametric negative translation* from classical linear logic to intuitionistic linear logic; see [25, Definition 2.2].

Let us fix an intuitionistic $\mathcal{L}$-formula F. Given a classical $\mathcal{L}_C$-formula A, we inductively define the intuitionistic $\mathcal{L}$-formula $A^{[F]}$ as follows:

$$
\begin{aligned}
p^{[F]} &:= \neg_F p & (p^{\perp})^{[F]} &:= p \\
\mathbf{1}^{[F]} &:= \neg_F \mathbf{1} & \perp^{[F]} &:= \mathbf{1} \\
\top^{[F]} &:= \mathbf{0} & \mathbf{0}^{[F]} &:= \neg_F \mathbf{0} \\
(B \otimes C)^{[F]} &:= \neg_F B^{[F]} \multimap C^{[F]} & (B \parr C)^{[F]} &:= \neg_F (B^{[F]} \multimap \neg_F C^{[F]}) \\
(B \mathbin{\&} C)^{[F]} &:= B^{[F]} \oplus C^{[F]} & (B \oplus C)^{[F]} &:= \neg_F (\neg_F B^{[F]} \oplus \neg_F C^{[F]}) \\
(!B)^{[F]} &:= \neg_F !\neg_F B^{[F]} & (?B)^{[F]} &:= !B^{[F]}
\end{aligned}
$$

where $\neg_F A$ is an abbreviation for $A \multimap F$. We have the following theorem. The proof can be found in the full version.

▶ **Theorem 2.** *Let $\vdash \Gamma$ be a classical $\mathcal{L}_C$-sequent and x a fresh propositional variable not occurring in Γ.*
(1) *$\vdash \Gamma$ is provable in* **LLW** *if and only if $\Gamma^{[x]} \vdash x$ is provable in* **ILLW**.
(2) *$\vdash \Gamma$ is provable in* **ELLW** *if and only if $\Gamma^{[x]} \vdash x$ is provable in* **IELW**.

This translation is convenient to show the complexity of the contraction-free logics that we deal with, e.g., the NP-completeness of the provability problem for **BCK**.

▶ **Corollary 3.** *The provability problem for* **BCK** *is NP-complete.*

Proof. Membership in NP is an immediate consequence of cut elimination for **BCK**. The proof is based on that of [28, Lemma 5.3]. In any cut-free proof in the system, the only applicable rules are (Init), ($\multimap$L), ($\multimap$R) and (W). Thus each subformula occurring in the

endsequent is analyzed at most once in such a proof-tree. This means that, the size of a cut-free proof in the system is polynomially bounded in the size of the endsequent. Hence the problem is in NP.[1]

For the hardness, we construct a polynomial-time reduction from provability in the $\{\otimes, \parr\}$-fragment of **LLW** (i.e., the constant-free fragment of multiplicative classical affine logic) into provability in **BCK**. The NP-completeness of the former is shown by Lincoln-Mitchell-Scedrov-Shankar [28], and Kanovich [23]. Let A be a classical $\{\otimes, \parr\}$-formula and x a fresh variable not in A. Our goal is to show that $\vdash A$ is provable in the $\{\otimes, \parr\}$-fragment of **LLW** if and only if $\vdash \neg_x A^{[x]}$ is provable in **BCK**. As a consequence of cut elimination for **LLW**, we can easily show that **LLW** is conservative over its $\{\otimes, \parr\}$-fragment. That is, $\vdash A$ is provable in the $\{\otimes, \parr\}$-fragment of **LLW** if and only if $\vdash A$ is provable in **LLW**. By Theorem 2, $\vdash A$ is provable in **LLW** if and only if $A^{[x]} \vdash x$ is provable in **ILLW**. Here $A^{[x]} \vdash x$ is an intuitionistic $\{\multimap\}$-sequent. Again, by the cut elimination theorem for **ILLW**, we know that **ILLW** is conservative over **BCK**; hence $A^{[x]} \vdash x$ is provable in **ILLW** if and only if $A^{[x]} \vdash x$ is provable in **BCK**. By the invertibility of ($\multimap$R), $A^{[x]} \vdash x$ is provable in **BCK** if and only if $\vdash \neg_x A^{[x]}$ is provable in **BCK**; thus we conclude that $\vdash A$ is provable in the $\{\otimes, \parr\}$-fragment of **LLW** if and only if $\vdash \neg_x A^{[x]}$ is provable in **BCK**. ◀

In particular, the translation $(\_)^{[\bot]}$ is a sort of standard negative translation from **LLW** (resp. **ELLW**) to **ILZW** (resp. **IEZW**). One can also show the following:

▶ **Theorem 4.** *Let $\vdash \Gamma$ be a classical $\mathcal{L}_C$-sequent. $\vdash \Gamma$ is provable in* **LLW** *(resp.* **ELLW***) if and only if $\Gamma^{[\bot]} \vdash$ is provable in* **ILZW** *(resp.* **IEZW***).*

3 Alternating branching VASS

The whole content of this section is taken from [26, Section 3]. Let d be in $\mathbb{N}$. The symbols $\bar{\mathsf{v}}_1, \bar{\mathsf{v}}_2, \ldots$ are used to denote d-dimensional vectors. In particular, we write $\bar{\mathsf{e}}_i$ for the *i-th unit vector* in $\mathbb{N}^d$ (i.e., the vector with a one in the i-th coordinate and zeros elsewhere), and $\bar{\mathsf{0}}$ for the vector whose every coordinate is zero.

An *alternating branching vector addition system with states and full zero tests* ($\mathrm{ABVASS}_{\bar{\mathsf{0}}}$, for short) is a structure of the form $\mathcal{A} = \langle Q, d, T_u, T_s, T_f, T_z \rangle$ where:

- Q is a finite set,
- d is in $\mathbb{N}$,
- T_u is a finite subset of $Q \times \mathbb{Z}^d \times Q$,
- T_s and T_f are subsets of Q^3, and
- T_z is a subset of Q^2.

We call Q a *state space*, d a *dimension*, and T_u (resp. T_s, T_f, T_z) the set of *unary* (resp. *split*, *fork*, *full zero test*) rules of $\mathcal{A}$. For readability, we always write $q \xrightarrow{\bar{\mathsf{u}}} q'$ for $(q, \bar{\mathsf{u}}, q') \in T_u$, $q \to q_1 \wedge q_2$ for $(q, q_1, q_2) \in T_f$, $q \to q_1 + q_2$ for $(q, q_1, q_2) \in T_s$, and $q \xrightarrow{\stackrel{?}{=}\bar{\mathsf{0}}} q'$ for $(q, q') \in T_z$. A *configuration* of $\mathcal{A}$ is an element of $Q \times \mathbb{N}^d$.

Given an $\mathrm{ABVASS}_{\bar{\mathsf{0}}}$ $\mathcal{A} = \langle Q, d, T_u, T_s, T_f, T_z \rangle$, the operational semantics for $\mathcal{A}$ is given by a deduction system over configurations in $Q \times \mathbb{N}^d$. It consists of the deduction rules depicted in Figure 3. Given a subset Q_ℓ of Q, a $Q_\ell \times \{\bar{\mathsf{0}}\}$-*leaf-covering deduction tree* in $\mathcal{A}$ is a finite tree labeled by configurations, where leaves are all in $Q_\ell \times \{\bar{\mathsf{0}}\}$ and each other node is obtained

[1] This was already pointed out in [20, p. 71].

$$\frac{q, \bar{\mathsf{v}}}{q', \bar{\mathsf{v}} + \bar{\mathsf{u}}}\ (q \xrightarrow{\bar{\mathsf{u}}} q') \qquad \frac{q, \bar{0}}{q', \bar{0}}\ (q \xrightarrow{\stackrel{?}{=}\bar{0}} q')$$

$$\frac{q, \bar{\mathsf{v}}}{q_1, \bar{\mathsf{v}} \qquad q_2, \bar{\mathsf{v}}}\ (q \to q_1 \wedge q_2) \qquad \frac{q, \bar{\mathsf{v}}_1 + \bar{\mathsf{v}}_2}{q_1, \bar{\mathsf{v}}_1 \qquad q_2, \bar{\mathsf{v}}_2}\ (q \to q_1 + q_2)$$

Figure 3 The deduction rules of an ABVASS$_{\bar{0}}$ $\mathcal{A} = \langle Q, d, T_u, T_s, T_f, T_z \rangle$, where $q \xrightarrow{\stackrel{?}{=}\bar{0}} q' \in T_z$, $q \xrightarrow{\bar{\mathsf{u}}} q' \in T_u$, $q \to q_1 + q_2 \in T_s$, and $q \to q_1 \wedge q_2 \in T_f$. The symbol $+$ stands for componentwise addition and $\bar{\mathsf{v}} + \bar{\mathsf{u}}$ must be in $\mathbb{N}^d$.

from its children by applying one of the deduction rules derived from $T_u \cup T_s \cup T_f \cup T_z$. A deduction tree $\mathcal{T}$ whose root configuration is $q, \bar{\mathsf{v}}$ is denoted by the following figure:

$$\begin{array}{c} q, \bar{\mathsf{v}} \\ \mathcal{T} \end{array}$$

We write $\mathcal{A}, Q_\ell \triangleright q, \bar{\mathsf{v}}$ if there exists a $Q_\ell \times \{\bar{0}\}$-leaf-covering deduction tree whose root configuration is $q, \bar{\mathsf{v}}$.

In addition, we also give an account of another semantics for ABVASS$_{\bar{0}}$s. Given an ABVASS$_{\bar{0}}$ $\mathcal{A} = \langle Q, d, T_u, T_s, T_f, T_z \rangle$, the *lossy semantics* for $\mathcal{A}$ is given by the aforementioned deduction system of $\mathcal{A}$ augmented with the following additional deduction rules:

$$\frac{q, \bar{\mathsf{v}} + \bar{\mathsf{e}}_i}{q, \bar{\mathsf{v}}}\ (\mathsf{loss})$$

for every $q \in Q$ and every $i \in \{1, \ldots, d\}$. In the natural way, we define the notion of $Q_\ell \times \{\bar{0}\}$-*leaf-covering lossy deduction tree* in $\mathcal{A}$. We write $\mathcal{A}, Q_\ell \triangleright_\ell q, \bar{\mathsf{v}}$ if there exists a $Q_\ell \times \{\bar{0}\}$-leaf-covering lossy deduction tree with root $q, \bar{\mathsf{v}}$.

4 Some Tower-complete problems

Following the terminology of [26, 35], we define

$$\textsc{Tower} := \bigcup_{f \in \mathsf{FELEMENTARY}} \textsc{DTime}(2^{\cdot^{\cdot^{\cdot^{2}}}} \}^{f(n) \text{ times}})$$

where FELEMENTARY denotes the set of elementary functions. This class of problems is closed under elementary many-one reductions (and elementary Turing reductions), i.e., for any two languages X and Y, if there is an elementary reduction from X to Y and Y is in Tower, then X is in Tower. The notion of *Tower-completeness* is defined with respect to elementary reductions in the usual manner. For an elaborate discussion on the fast-growing complexity hierarchy, we refer the reader to [35].

For later use, we summarize here some Tower-complete problems. Given an ABVASS$_{\bar{0}}$ $\mathcal{A} = \langle Q, d, T_u, T_s, T_f, T_z \rangle$, $Q_\ell \subseteq Q$, and $q_r \in Q$, the *reachability problem* (resp. *lossy reachability problem*) asks whether it holds that $\mathcal{A}, Q_\ell \triangleright q_r, \bar{0}$ (resp. $\mathcal{A}, Q_\ell \triangleright_\ell q_r, \bar{0}$).

▶ **Theorem 5** (Lazić and Schmitz [26], Theorem 3.6). *The lossy reachability problem for ABVASS$_{\bar{0}}$s is Tower-complete.*

In contrast, the reachability problem is undecidable for ABVASS$_{\bar{0}}$s. In fact, the same holds even for alternating VASSs, which are ABVASS$_{\bar{0}}$s with only unary rules and fork rules; see [28, Section 3.4] and [26, Section 3.3.1] for details.

Intriguingly, the lossy reachability problem is complete for TOWER even when a restricted version of $\mathrm{ABVASS}_{\bar{0}}$s is considered. An *ordinary ABVASS*$_{\bar{0}}$ is an $\mathrm{ABVASS}_{\bar{0}}$ $\mathcal{A} = \langle Q, d, T_u, T_s, T_f, T_z \rangle$ where for every $q \xrightarrow{\bar{\mathsf{u}}} q' \in T_u$ it holds that either $\bar{\mathsf{u}} = \bar{\mathsf{e}}_i$ or $\bar{\mathsf{u}} = -\bar{\mathsf{e}}_i$ for some $1 \leq i \leq d$. A *branching vector addition system with states* (BVASS, for short) is an $\mathrm{ABVASS}_{\bar{0}}$ $\mathcal{A} = \langle Q, d, T_u, T_s, T_f, T_z \rangle$ where T_f and T_z are both empty. The main results in the remaining sections rely crucially on the following two complexity results:

▶ **Theorem 6** (Lazić and Schmitz [26], Lemma 3.5 and Theorem 3.6)**.** *The lossy reachability problem for ordinary BVASSs is* TOWER*-complete.*

▶ **Theorem 7** (Lazić and Schmitz [26]; Fact 4.2, Corollary 5.4 and Corollary 6.3)**.** *The provability problems for* **ILZW** *and* **LLW** *are* TOWER*-complete.*

5 Membership in Tower of contraction-free logics

This section consists of two parts. We first show that provability in elementary affine logic is in TOWER (Section 5.1). Secondly, we also show the TOWER-membership of deducibility in $\mathbf{FL_{ew}}$ and related logical systems (Section 5.2).

5.1 Tower upper bound for provability in elementary affine logic

We show that there exists an elementary reduction from the provability problem for **IEZW** to the lossy reachablity problem for $\mathrm{ABVASS}_{\bar{0}}$s. Our reduction is a slightly modified version of the (polynomial-space) reduction, given in [26, Section 4.1.2], from provability in **ILZW** to lossy reachability in $\mathrm{ABVASS}_{\bar{0}}$.

Let F be an intuitionistic $\mathcal{L}$-formula. Let S be the set of subformulas of F, $S_!$ the set of formulas in S of the form $!B$, and $\bullet$ a fresh symbol not in S. For $\Pi \in S \cup \{\varepsilon\}$, we define $\Pi^\dagger = A$ if $\Pi = A$, and $\Pi^\dagger = \bullet$ otherwise. A multiset over S is just a map from S to $\mathbb{N}$, i.e., an element of $\mathbb{N}^S$. Given a multiset m over S, $m(B)$ denotes the multiplicity of a formula B in m.

Fix an enumeration $F_1, \ldots, F_d$ of all the formulas in S. A multiset m over S can be expressed as $F_1^{m(F_1)}, \ldots, F_d^{m(F_d)}$. For each multiset m over S, we write $\bar{\mathsf{v}}_m$ for the vector $\langle m(F_1), \ldots, m(F_d) \rangle$ in $\mathbb{N}^d$. In particular, we write $\bar{\mathsf{e}}_B$ for the vector $\bar{\mathsf{v}}_B$ corresponding to a formula B in S. Note that $\bar{\mathsf{v}}_m = \bar{\mathsf{0}}$ if m is the empty multiset. We write $\sigma(m)$ for the *support* of a multiset m, i.e., $\{B \in S \mid m(B) > 0\}$. We then construct an $\mathrm{ABVASS}_{\bar{0}}$ $\mathcal{A}_F^E$ as below:

- The dimension of $\mathcal{A}_F^E$ is d $(= |S|)$.
- The state space of $\mathcal{A}_F^E$ contains $\mathcal{P}(S_!) \times (S \cup \{\bullet\})$, a distinguished leaf state q_ℓ, and several intermediate states which are needed for defining the rules for $(\multimap\mathsf{L})$, $(\&\mathsf{L})$, $(\oplus\mathsf{L})$, $(\otimes\mathsf{L})$, $(\mathbf{0}\mathsf{L})$, $(\top\mathsf{R})$ and (func) in $\mathcal{A}_F^E$ (cf. Figure 4).
- The rules and intermediate states of $\mathcal{A}_F^E$ are defined as in Figure 4.

The construction of $\mathcal{A}_F^E$ is quite similar to that of the $\mathrm{ABVASS}_{\bar{0}}$ $\mathcal{A}_F^I$ defined in [26, Section 4.1.2]. We stress that $\mathcal{A}_F^E$ has the rules for (func) instead of the rules for $(!\mathsf{D})$ and $(!\mathsf{P})$ (see Figure 5 in Section 5.2), whereas $\mathcal{A}_F^I$ has the rules for $(!\mathsf{D})$ and $(!\mathsf{P})$ instead of the rules for (func). Notice that $\mathcal{A}_F^E$ does not have the rules corresponding to the inference rule of (W) in **ILZW**. The left-weakening rule is implemented by loss rules derived from the lossy semantics for $\mathcal{A}_F^E$.

Let $\Theta, \Gamma \vdash \Pi$ be an intuitionistic $\mathcal{L}$-sequent such that Θ is a multiset of formulas in $S_!$, Γ is a multiset of formulas in $S \setminus S_!$, and Π is in $S \cup \{\varepsilon\}$. It is translated as the configuration $\sigma(\Theta), \Pi^\dagger, \bar{\mathsf{v}}_\Gamma$ in $\mathcal{P}(S_!) \times (S \cup \{\bullet\}) \times \mathbb{N}^d$. We have the key theorem of this subsection; see the full version for a detailed proof.

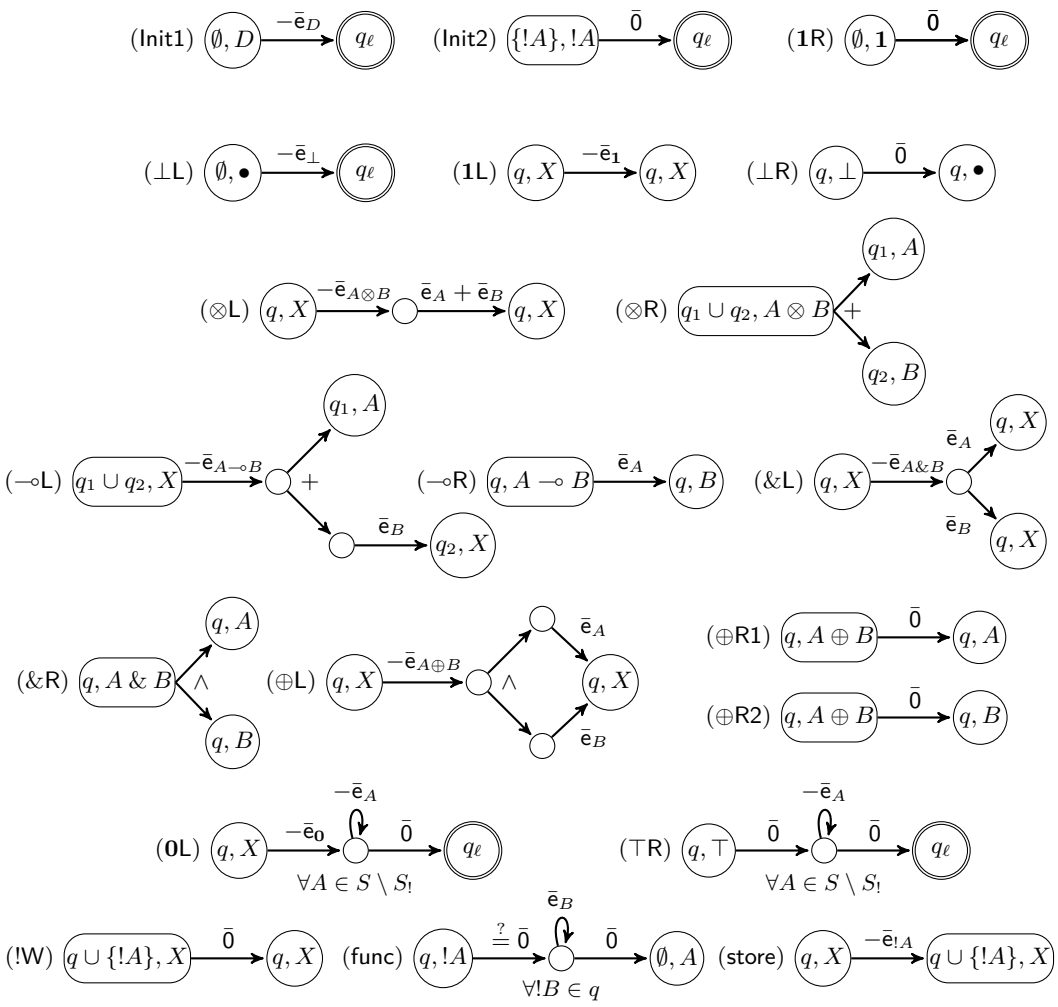


Figure 4 Rules and intermediate states of $\mathcal{A}_F^E$. All formulas range over S, q, q_1, q_2 range over $\mathcal{P}(S_!)$, D ranges over $S \setminus S_!$, and X ranges over $S \cup \{\bullet\}$. Small circles stand for intermediate states.

▶ **Theorem 8.** *Let* Π *be in* $S \cup \{\varepsilon\}$, Θ *a multiset of formulas in* $S_!$, Γ *a multiset of formulas in* $S \setminus S_!$. $\Theta, \Gamma \vdash \Pi$ *is provable in* **IEZW** *if and only if* $\mathcal{A}_F^E, \{q_\ell\} \rhd_\ell \sigma(\Theta), \Pi^\dagger, \bar{\mathsf{v}}_\Gamma$.

In particular, Theorem 8 guarantees that for any intuitionistic $\mathcal{L}$-formula F, F is provable in **IEZW** if and only if $\mathcal{A}_F^E, \{q_\ell\} \rhd_\ell \emptyset, F, \bar{0}$. By Theorems 5 and 4:

▶ **Corollary 9.** *The provability problems for* **IEZW** *and* **ELLW** *are in* Tower.

We stress that the Tower upper bound also holds for fragments of **IEZW** and **ELLW**, e.g., the $\{\multimap, !\}$-fragment of **IEZW**.

5.2 Tower upper bound for deducibility in FLew and related systems

We first describe the notion of !-*prenex sequent* which originates in [38, Section 2]. Let $\mathcal{K}$ be a language such that $\bot \in \mathcal{K} \subseteq \mathcal{L}$. A !-*prenex* $\mathcal{K}$-*sequent* is an intuitionistic sequent of the form $!\Gamma, \Delta \vdash \Pi$, where Γ and Δ are finite multisets of intuitionistic $\mathcal{K}$-formulas, and Π is an intuitionistic $\mathcal{K}$-formula or the empty multiset. Similarly, let $\mathcal{K}$ be a sublanguage of $\mathcal{L}_C$. A

?*-prenex* $\mathcal{K}$*-sequent* is a right-hand sided sequent of the form $\vdash ?\Gamma, \Delta$, where Γ and Δ are finite multisets of classical $\mathcal{K}$-formulas. We write Γ^n for the multiset sum of n copies of Γ for each $n \geq 0$.

▶ **Lemma 10.** *Let* $!\Gamma, \Delta \vdash \Pi$ *be a* !*-prenex* $\{\otimes, \multimap, \&, \oplus, \mathbf{1}, \bot\}$*-sequent.*
(1) *If* $!\Gamma, \Delta \vdash \Pi$ *is provable in* **ILZW***, then* $\Gamma^n, \Delta \vdash \Pi$ *is provable in* $\mathbf{FL_{ei}}$ *for some* $n \geq 0$.
(2) *If* $!\Gamma, \Delta \vdash \Pi$ *is provable in* $\mathbf{ILZW}'$*, then* $\Gamma^n, \Delta \vdash \Pi$ *is provable in* $\mathbf{FL_{ew}}$ *for some* $n \geq 0$.

Proof. The proof of Statement (1) proceeds by induction on the size of the cut-free proof of $!\Gamma, \Delta \vdash \Pi$ in **ILZW**. We perform a case analysis, depending on which inference rule is applied last.

We consider only the case of $(\multimap\mathrm{L})$. If $!\Gamma, !\Sigma, A \multimap B, \Delta, \Xi \vdash \Pi$ is obtained from $!\Gamma, \Delta \vdash A$ and $!\Sigma, B, \Xi \vdash \Pi$ by an application of $(\multimap\mathrm{L})$, then by the induction hypothesis, $\Gamma^{n'}, \Delta \vdash A$ is provable in $\mathbf{FL_{ei}}$ for some n', and $\Sigma^{n''}, B, \Xi \vdash \Pi$ is provable in $\mathbf{FL_{ei}}$ for some n''. Applying $(\multimap\mathrm{L})$ we obtain a proof of $\Gamma^{n'}, \Sigma^{n''}, A \multimap B, \Delta, \Xi \vdash \Pi$ in $\mathbf{FL_{ei}}$. Note that n' is not always equal to n''. However, we may unify them using the rule of (W); thus $(\Gamma, \Sigma)^{n'+n''}, A \multimap B, \Delta, \Xi \vdash \Pi$ is provable in $\mathbf{FL_{ei}}$. The remaining cases are similar.

One can show Statement (2) similarly. ◀

Similarly to the above theorem, one can also show the following:

▶ **Lemma 11.** *Let* $\vdash ?\Gamma, \Delta$ *be a* ?*-prenex* $\{\otimes, \parr, \&, \oplus, \mathbf{1}, \bot\}$*-sequent. If* $\vdash ?\Gamma, \Delta$ *is provable in* **LLW***, then* $\vdash \Gamma^n, \Delta$ *is provable in* $\mathbf{InFL_{ew}}$ *for some* $n \geq 0$.

Lemmas 10 and 11 are affine analogues of [38, Proposition 2.6]. Interestingly, a similar idea is found in the proof of the *local deduction theorem* for $\mathbf{FL_{ew}}$; see [15, Corollary 2.15]. The following lemma, which is inspired by [28, Lemmas 3.2 and 3.3], provides a very simple reduction from $\mathbf{FL_{ei}}$ deducibility (resp. $\mathbf{FL_{ew}}$ deducibility) into **ILZW** provability (resp. $\mathbf{ILZW}'$ provability).

▶ **Lemma 12.** *Let* Φ *be a finite set of intuitionistic* $\{\otimes, \multimap, \&, \oplus, \mathbf{1}, \bot\}$*-formulas and* $\Gamma \vdash \Pi$ *an intuitionistic* $\{\otimes, \multimap, \&, \oplus, \mathbf{1}, \bot\}$*-sequent.*
(1) $\Gamma \vdash \Pi$ *is provable in* $\mathbf{FL_{ei}}[\Phi]$ *if and only if* $!\Phi, \Gamma \vdash \Pi$ *is provable in* **ILZW**.
(2) $\Gamma \vdash \Pi$ *is provable in* $\mathbf{FL_{ew}}[\Phi]$ *if and only if* $!\Phi, \Gamma \vdash \Pi$ *is provable in* $\mathbf{ILZW}'$.

Proof. We only prove the first statement, the proof of the second one being similar. For the *if* part, let us suppose that $!\Phi, \Gamma \vdash \Pi$ is provable in **ILZW**. Since $!\Phi, \Gamma \vdash \Pi$ is a !-prenex $\{\otimes, \multimap, \&, \oplus, \mathbf{1}, \bot\}$-sequent, $\Phi^n, \Gamma \vdash \Pi$ is provable in $\mathbf{FL_{ei}}$ for some n by Lemma 10. Since $\vdash B$ is an initial sequent of $\mathbf{FL_{ei}}[\Phi]$ for each $B \in \Phi$, we can construct a proof of $\Gamma \vdash \Pi$ in $\mathbf{FL_{ei}}[\Phi]$ by several applications of (Cut). The *only-if* part follows by induction on the height of the proof of $\Gamma \vdash \Pi$ in $\mathbf{FL_{ei}}[\Phi]$. ◀

In the same way as before, the deducibility problems for $\mathbf{FL}^{+}_{\mathbf{ei}}$ and **BCK** are also reduced to the provability problem for **ILZW**. Furthermore, one can show that there is also a straightforward reduction from $\mathbf{InFL_{ew}}$ deducibility to **LLW** provability, using Lemma 11:

▶ **Lemma 13.** *Let* Φ *be a finite set of classical* $\{\otimes, \parr, \&, \oplus, \mathbf{1}, \bot\}$*-formulas and* $\vdash \Gamma$ *a classical* $\{\otimes, \parr, \&, \oplus, \mathbf{1}, \bot\}$*-sequent.* $\vdash \Gamma$ *is provable in* $\mathbf{InFL_{ew}}[\Phi]$ *if and only if* $\vdash ?\Phi^{\bot}, \Gamma$ *is provable in* **LLW**.

Recall that the provability problems for **ILZW** and **LLW** are both in Tower by Theorem 7. We obtain:

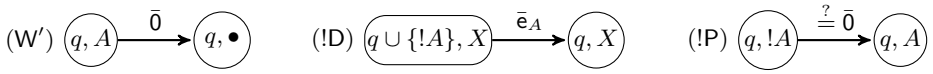


Figure 5 The ABVASS$_{\bar{0}}$ rules for (W′), (!D) and (!P).

▶ **Corollary 14.** *The following decision problems are in* Tower.
- *the deducibility problem for* **BCK**,
- *the deducibility problem for* $\mathbf{FL}^{+}_{\mathbf{ei}}$,
- *the deducibility problem for* $\mathbf{FL}_{\mathbf{ei}}$,
- *the deducibility problem for* $\mathbf{InFL}_{\mathbf{ew}}$.

At the end of this section, we show the membership in Tower of the deducibility problem for $\mathbf{FL}_{\mathbf{ew}}$. It suffices by Lemma 12 to show that the provability problem for $\mathbf{ILZW}'$ is in Tower.

Let F be an instance of the provability problem for $\mathbf{ILZW}'$. As before, S denotes the set of subformulas of F, $S_!$ the set of formulas in S of the form $!B$, and $\bullet$ a distinguished symbol. We then construct an ABVASS$_{\bar{0}}$ $\mathcal{A}^{I'}_F$ of dimension $|S|$, by modifying the construction of $\mathcal{A}^{E}_F$ given in the previous subsection. The state space of $\mathcal{A}^{I'}_F$ contains $\mathcal{P}(S_!) \times (S \cup \{\bullet\})$, a distinguished leaf state q_ℓ, and intermediate states that are needed for defining the rules for ($\multimap$L), (&L), ($\oplus$L), ($\otimes$L), (**0**L), and ($\top$R), cf. Figure 4. The rules of $\mathcal{A}^{I'}_F$ are (Init1), (Init2), (store), (**1**L), (**1**R), ($\bot$L), ($\bot$R), ($\multimap$L), ($\multimap$R), ($\otimes$L), ($\otimes$R), (&L), (&R), ($\oplus$L), ($\oplus$R1), ($\oplus$R2), (**0**L), ($\top$R), (!W), (W′), (!D) and (!P), all of which except for (W′), (!D) and (!P), are depicted in Figure 4. The rules for (W′), (!D) and (!P) are defined as in Figure 5. Clearly, these three types of ABVASS$_{\bar{0}}$ rules correspond to the inference rules of (W'), (!D) and (!P) in $\mathbf{ILZW}'$, respectively. Note that $\mathcal{A}^{I'}_F$ is not equipped with the rule for (func). We can show that $\mathcal{A}^{I'}_F$ simulates the proof search of F in $\mathbf{ILZW}'$ with the lossy semantics; see the full version for a proof.

▶ **Theorem 15.** *Let* Π *be in* $S \cup \{\varepsilon\}$, Θ *a multiset of formulas in* $S_!$, Γ *a multiset of formulas in* $S \setminus S_!$. $\Theta, \Gamma \vdash \Pi$ *is provable in* $\mathbf{ILZW}'$ *if and only if* $\mathcal{A}^{I'}_F, \{q_\ell\} \rhd_\ell \sigma(\Theta), \Pi^\dagger, \bar{\mathsf{v}}_\Gamma$.

Specifically, $\mathcal{A}^{I'}_F, \{q_\ell\} \rhd_\ell \emptyset, F, \bar{0}$ if and only if $\vdash F$ is provable in $\mathbf{ILZW}'$; thus Lemma 12 and Theorem 5 provide:

▶ **Corollary 16.** *The provability problem for* $\mathbf{ILZW}'$ *is in* Tower.

▶ **Corollary 17.** *The deducibility problem for* $\mathbf{FL}_{\mathbf{ew}}$ *is in* Tower.

6 Tower-hardness of contraction-free logics

For our purposes, we recall from [26, Section 4.2] a log-space reduction from the lossy reachability problem for ordinary BVASSs to the provability problem of ?-prenex $\{\otimes, ⅋\}$-sequents in **LLW**.

Let $(\mathcal{B}, Q_\ell, q_r)$ be an instance of the lossy reachability problem for ordinary BVASSs, where $\mathcal{B} = \langle Q, d, T_u, T_s, \emptyset, \emptyset\rangle$. We fix a set $Q \cup \{e_i \mid 1 \le i \le d\}$ of propositional variables. Given $(q, \bar{\mathsf{v}}) \in Q \times \mathbb{N}^d$, we define $\theta(q, \bar{\mathsf{v}}) = q^\bot, (e^\bot_1)^{\bar{\mathsf{v}}(1)}, \ldots, (e^\bot_d)^{\bar{\mathsf{v}}(d)}$, where $\bar{\mathsf{v}}(i)$ stands for the i-th coordinate of $\bar{\mathsf{v}}$. We write T for the set of the three types of non-logical axioms, each of which is derived from $T_u \cup T_s$ as follows:
- $\vdash q^\bot, q_1 \otimes e_i$ for $q \xrightarrow{\bar{\mathsf{e}}_i} q_1 \in T_u$,
- $\vdash q^\bot, e^\bot_i, q_1$ for $q \xrightarrow{-\bar{\mathsf{e}}_i} q_1 \in T_u$,

- $\vdash q^\perp, q_1 \parr q_2$ for $q \to q_1 + q_2 \in T_s$.

Each sequent in T is of the form $\vdash q_1^\perp, \dots, q_n^\perp, C$, where $q_1^\perp, \dots, q_n^\perp$ are negative literals and C is a classical $\{\otimes, \parr\}$-formula. For any sequent $t = \vdash q_1^\perp, \dots, q_n^\perp, C$ in T, we define $\ulcorner t \urcorner = q_1 \otimes \cdots \otimes q_n \otimes C^\perp$. Given a finite set $T = \{t_1, \dots, t_n\}$ of sequents, $\ulcorner T \urcorner$ denotes the multiset $\ulcorner t_1 \urcorner, \dots, \ulcorner t_n \urcorner$. It then holds that, for any $(q, \bar{\mathsf{v}}) \in Q \times \mathbb{N}^d$, $\mathcal{B}, Q_\ell \rhd_\ell q, \bar{\mathsf{v}}$ if and only if $\vdash ?\ulcorner T \urcorner, ?Q_\ell, \theta(q, \bar{\mathsf{v}})$ is provable in **LLW**. In particular, the following holds:

▶ **Theorem 18** (Lazić and Schmitz [26], Section 4.2.3). *$\mathcal{B}, Q_\ell \rhd_\ell q_r, \bar{0}$ if and only if $\vdash ?\ulcorner T \urcorner, ?Q_\ell, q_r^\perp$ is provable in* **LLW**.

The key observation here is that $\vdash ?\ulcorner T \urcorner, ?Q_\ell, q_r^\perp$ forms a ?-prenex $\{\otimes, \parr\}$-sequent. Thus in conjunction with Theorem 6, we obtain:

▶ **Corollary 19** (Lazić and Schmitz [26], Section 4.2.3). *The problem of determining if a given ?-prenex $\{\otimes, \parr\}$-sequent is provable in* **LLW** *is* TOWER*-hard.*

6.1 Tower-hardness of deducibility in FLew and related systems

A *!-prenex $\{\multimap\}$-sequent* is an intuitionistic $\{\multimap, !\}$-sequent of the form $!\Gamma, \Delta \vdash C$ where the only connective occurring in Γ, Δ, C is $\multimap$. We prove:

▶ **Theorem 20.** *The problem of determining if a given !-prenex $\{\multimap\}$-sequent is provable in the $\{\multimap, !\}$-fragment of* **ILLW** *is* TOWER*-hard.*

Proof. In view of Corollary 19, we reduce from the problem of whether a given ?-prenex $\{\otimes, \parr\}$-sequent is provable in **LLW**. Let $\vdash ?\Gamma, \Delta$ be a ?-prenex $\{\otimes, \parr\}$-sequent and x a new propositional variable not occurring in $?\Gamma, \Delta$. Theorem 2 guarantees that $\vdash ?\Gamma, \Delta$ is provable in **LLW** if and only if $(?\Gamma, \Delta)^{[x]} \vdash x$ is provable in **ILLW**. Clearly, the latter sequent forms a !-prenex $\{\multimap\}$-sequent. Due to the fact that **ILLW** admits cut-elimination, **ILLW** is a conservative extension of its $\{\multimap, !\}$-fragment; hence $(?\Gamma, \Delta)^{[x]} \vdash x$ is provable in **ILLW** if and only if it is provable in the $\{\multimap, !\}$-fragment of **ILLW**. We conclude that $\vdash ?\Gamma, \Delta$ is provable in **LLW** if and only if $(?\Gamma, \Delta)^{[x]} \vdash x$ is provable in the $\{\multimap, !\}$-fragment of **ILLW**. Hence the problem is hard for TOWER. ◀

We furthermore show the following lemma:

▶ **Lemma 21.** *Let $!\Gamma, \Delta \vdash A$ be a !-prenex $\{\multimap\}$-sequent. The following statements are mutually equivalent:*

(1) *$!\Gamma, \Delta \vdash A$ is provable in the $\{\multimap, !\}$-fragment of* **ILLW**,
(2) *$\Delta \vdash A$ is provable in* $\mathbf{BCK}[\sigma(\Gamma)]$,
(3) *$\Delta \vdash A$ is provable in* $\mathbf{FL}_{\mathbf{ei}}^{+}[\sigma(\Gamma)]$,
(4) *$\Delta \vdash A$ is provable in* $\mathbf{FL}_{\mathbf{ei}}[\sigma(\Gamma)]$,
(5) *$\Delta \vdash A$ is provable in* $\mathbf{FL}_{\mathbf{ew}}[\sigma(\Gamma)]$.

Here $\sigma(\Gamma)$ stands for the support of Γ, i.e., the set of formulas that are contained in Γ at least once.

Proof. For starters, observe that the following claim holds:

(a) Let $!\Xi, \Sigma \vdash C$ be a !-prenex $\{\multimap\}$-sequent. If $!\Xi, \Sigma \vdash C$ is provable in the $\{\multimap, !\}$-fragment of **ILLW**, then $\Xi^n, \Sigma \vdash C$ is provable in **BCK** for some n.

Similarly to Lemma 10, this is shown by induction on the size of cut-free proofs in the $\{\multimap, !\}$-fragment of **ILLW**. To show that Statement (1) implies Statement (2), suppose that $!\Gamma, \Delta \vdash A$ is provable in the $\{\multimap, !\}$-fragment of **ILLW**. By Claim (a), $\Gamma^n, \Delta \vdash A$ is provable

in **BCK** for some n. For each formula B in Γ, B is also in $\sigma(\Gamma)$. Hence we obtain a proof of $\Delta \vdash A$ in $\mathbf{BCK}[\sigma(\Gamma)]$, applying (Cut) several times. The implications $(2 \Rightarrow 3)$, $(3 \Rightarrow 4)$ and $(4 \Rightarrow 5)$ trivially hold.

We now embark on the proof of the remaining implication $(5 \Rightarrow 1)$. By straightforward induction on the length of derivations, we can show:

(b) Let Ξ be a finite multiset of formulas in $\mathbf{FL_{ew}}$ and $\Sigma \vdash \Pi$ a sequent of $\mathbf{FL_{ew}}$. If $\Sigma \vdash \Pi$ is provable in $\mathbf{FL_{ew}}[\sigma(\Xi)]$, then $!\Xi, \Sigma \vdash \Pi$ is provable in $\mathbf{ILZW'}$.

Let us assume that $\Delta \vdash A$ is provable in $\mathbf{FL_{ew}}[\sigma(\Gamma)]$. By Claim (b), $!\Gamma, \Delta \vdash A$ is provable in $\mathbf{ILZW'}$. By the cut elimination theorem for $\mathbf{ILZW'}$, we can easily check that $\mathbf{ILZW'}$ is conservative over the $\{\multimap, !\}$-fragment of **ILLW**. Hence $!\Gamma, \Delta \vdash A$ is provable in the $\{\multimap, !\}$-fragment of **ILLW**. ◀

The above lemma provides a polynomial time reduction from the provability problem of !-prenex $\{\multimap\}$-sequents in the $\{\multimap, !\}$-fragment of **ILLW** to deducibility in **BCK**, $\mathbf{FL^{+}_{ei}}$, $\mathbf{FL_{ei}}$, and $\mathbf{FL_{ew}}$; hence by Corollary 20 we obtain the Tower-hardness of deducibility in these systems. In the same way as before, we can show:

▶ **Lemma 22.** *Let $\vdash ?\Gamma, \Delta$ be a ?-prenex $\{\otimes, \parr, \&, \oplus, \mathbf{1}, \bot\}$-sequent. $\vdash ?\Gamma, \Delta$ is provable in* **LLW** *if and only if $\vdash \Delta$ is provable in $\mathbf{InFL_{ew}}[\sigma(\Gamma^{\bot})]$.*

By Corollary 19 and Lemma 22, the deducibility problem for $\mathbf{InFL_{ew}}$ is also hard for Tower. The deducibility problems for **BCK**, $\mathbf{FL^{+}_{ei}}$, $\mathbf{FL_{ei}}$, $\mathbf{FL_{ew}}$, and $\mathbf{InFL_{ew}}$ are all in Tower by Corollaries 14 and 17. We thus conclude:

▶ **Corollary 23.** *Each of the following decision problems is complete for Tower:*

- *the deducibility problem for* **BCK**,
- *the deducibility problem for* $\mathbf{FL^{+}_{ei}}$,
- *the deducibility problem for* $\mathbf{FL_{ei}}$,
- *the deducibility problem for* $\mathbf{FL_{ew}}$,
- *the deducibility problem for* $\mathbf{InFL_{ew}}$.

6.2 Tower-hardness of provability in elementary affine logic

▶ **Lemma 24.** *Let Γ be a finite multiset of classical $\{\otimes, \parr, \&, \oplus, \mathbf{1}, \bot\}$-formulas and A a classical $\{\otimes, \parr, \&, \oplus, \mathbf{1}, \bot\}$-formula. $\vdash ?\Gamma, A$ is provable in* **LLW** *if and only if $\vdash ?\Gamma, !A$ is provable in* **ELLW**.

Proof. Suppose that $\vdash ?\Gamma, A$ is provable in **LLW**. By Lemma 11, $\vdash \Gamma^n, A$ is provable in $\mathbf{InFL_{ew}}$ for some n. Obviously, $\vdash \Gamma^n, A$ is provable in **ELLW** for some n. Using the rule of (F) and structural rules, we obtain a proof of $\vdash ?\Gamma, !A$ in **ELLW**. For the other direction, assume that $\vdash ?\Gamma, !A$ is provable in **ELLW**. It is provable in **LLW** since every sequent that is provable in **ELLW** is provable in **LLW**. Recall that $\vdash ?A^{\bot}, A$ is provable in **LLW** whereas it is not provable in **ELLW**. We therefore obtain a proof of $\vdash ?\Gamma, A$ in **LLW** by a single application of (Cut). ◀

It follows from Theorem 18 and Lemma 24 that $\mathcal{B}, Q_\ell \rhd_\ell q_r, \bar{0}$ if and only if $\vdash ?\ulcorner T \urcorner, ?Q_\ell, !q_r^{\bot}$ is provable in **ELLW**. Here the sequent $\vdash ?\ulcorner T \urcorner, ?Q_\ell, !q_r^{\bot}$ is built from literals using only connectives in $\{\otimes, \parr, !, ?\}$. We therefore obtain:

▶ **Corollary 25.** *The problem of determining if a given classical $\{\otimes, \parr, !, ?\}$-sequent is provable in* **ELLW** *is Tower-hard, and hence so is the provability problem for* **ELLW**.

Combining this with Corollary 9, we obtain:

▶ **Corollary 26.** *The provability problem for* **ELLW** *is* TOWER*-complete.*

Clearly, provability in the multiplicative-exponential fragment of **ELLW** is also complete for TOWER. The same holds for a very small fragment of **IELW**:

▶ **Theorem 27.** *The provability problem for the* $\{\multimap, !\}$*-fragment of* **IELW** *is* TOWER*-complete.*

Proof. Similar to the proof of Theorem 20. The problem is clearly in TOWER due to Corollary 9. Let $\vdash \Gamma$ be a classical $\{\otimes, \parr, !, ?\}$-sequent and x a fresh variable not occurring in this sequent. By Theorem 2, $\vdash \Gamma$ is provable in **ELLW** if and only if $\Gamma^{[x]} \vdash x$ is provable in **IELW**. By cut elimination for **IELW**, we know that **IELW** is a conservative extension of its $\{\multimap, !\}$-fragment. Thus $\Gamma^{[x]} \vdash x$ is provable in **IELW** if and only if it is provable in the $\{\multimap, !\}$-fragment of **IELW**. Consequently, $\vdash \Gamma$ is provable in **ELLW** if and only if $\Gamma^{[x]} \vdash x$ is provable in the $\{\multimap, !\}$-fragment of **IELW**. Hence there is a polynomial time reduction from the problem of whether a given classical $\{\otimes, \parr, !, ?\}$-sequent is provable in **ELLW** to provability in the $\{\multimap, !\}$-fragment of **IELW**. By Corollary 25, provability in the $\{\multimap, !\}$-fragment of **IELW** is hard for TOWER. ◀

7 Concluding remarks

We have shown the TOWER-completeness of deducibility in some contraction-free substructural logics without modal operators. We hope that our work sheds new light on computational aspects of fuzzy logic. This is because $\mathbf{FL_{ew}}$ forms a theoretical basis for a wide range of fuzzy logics. In fact, one can construct from $\mathbf{FL_{ew}}$ various fuzzy logics (such as monoidal t-norm based logic, basic logic, weak nilpotent minimum logic, Łukasiewicz logic and product logic) by adding some new axioms, cf. [13, 18]. Remarkably, the latter four of the above examples are shown to be coNP-complete with respect to both provability and deducibility; see [19] for a detailed survey. Together with those facts, our result suggests that there is a critical difference between $\mathbf{FL_{ew}}$ and its fuzzy extensions with respect to computational complexity.

We summarize the known complexity results in Tables 1 and 2. We adopt the notation of combinatory logic in Table 1:

- **BCI** = the implicational fragment of intuitionistic linear logic,
- **BCIW** = the extension of **BCI** by the rule of contraction,
- **BCK** = the extension of **BCI** by the rule of weakening.

BCIW is usually denoted by $\mathbf{R}_{\to}$ in the relevance logic community. $\mathbf{IL}_{\to}$ stands for the extension of **BCI** by weakening and contraction. It is nothing but the implicational fragment of intuitionistic propositional logic (**IL**). In Table 2, $\mathbf{FL_e}$ (resp. $\mathbf{FL_{ec}}$) denotes the extension of **BCI** (resp. **BCIW**) by the connectives $\otimes$, $\&$, $\oplus$, $\mathbf{1}$, and $\bot$.

Below we comment on some results not covered in the main sections.

Ackermannian complexity and deducibility in BCI. In [35] Schmitz also defined a non-primitive recursive complexity class by:

$$\text{ACKERMANN} := \bigcup_{f \in \mathsf{FPR}} \text{DTIME}(Ack(f(n)))$$

Table 1 Complexity results of extensions of **BCI**.

	Provability	Deducibility
BCI	NP-complete [7]	open (Ackermann-hard, cf. Corollary 31)
BCIW	2-ExpTime-complete [36]	decidable (2-ExpTime-hard, cf. [36])
BCK	NP-complete (Corollary 3)	Tower-complete (Corollary 23)
$\mathbf{IL}_{\to}$	Pspace-complete [37]	Pspace-complete [37]

Table 2 Complexity results of extensions of $\mathbf{FL_e}$.

	Provability	Deducibility
$\mathbf{FL_e}$	Pspace-complete [28]	undecidable [28]
$\mathbf{FL_{ec}}$	Ackermann-complete, cf. [26, 41]	Ackermann-complete, cf. [26, 41]
$\mathbf{FL_{ei}}$	Pspace-complete, cf. [21]	Tower-complete (Corollary 23)
$\mathbf{FL_{ew}}$	Pspace-complete, cf. [21]	Tower-complete (Corollary 23)
$\mathbf{FL_{ewc}}$ $(= \mathbf{IL})$	Pspace-complete [37]	Pspace-complete [37]

where *Ack* is the Ackermannian function and FPR denotes the set of primitive recursive functions. The class contains the class PR of primitive recursive problems, and is closed under primitive recursive reductions. As in [26, 35], we define the notion of *Ackermann-hardness* using primitive recursive reductions.

The decidability of deducibility in **BCI** is a long-standing open problem. However, one reviewer pointed out that the Ackermann-hardness of deducibility in **BCI** follows from the recent breakthrough by Czerwiński and Orlikowski [9], and Leroux [27], who independently proved that the reachability problem for VASSs is Ackermann-hard.

Let us sketch the proof of the Ackermannian lower bound for deducibility in **BCI**. Clearly, by the aforementioned result in [9, 27], reachability in BVASSs is also hard for Ackermann. As is the case of lossy reachability for ABVASS_0s, the BVASS-reachability problem is reduced to reachability in ordinary BVASSs (cf. Section 4); see [26, Lemma 3.5] for details. Furthermore, reachability in ordinary BVASSs can be efficiently encoded to the problem of whether classical linear logic (**LL**), i.e., **LLW** without the rule of (W), proves a given ?-prenex $\{\otimes, ⅋\}$-sequent; see [26, Section 4.2.2]. To sum up, the following holds:

▶ **Theorem 28** ([9, 26, 27]). *The problem of determining if a given ?-prenex $\{\otimes, ⅋\}$-sequent is provable in* **LL** *is Ackermann-hard.*

The translation in Section 2.2 also holds between **LL** and **ILL** (i.e., **ILLW** without the unrestricted left-weakening); see the full version for a sketch of a proof.

▶ **Theorem 29.** *Let $\vdash \Gamma$ be a classical $\mathcal{L}_C$-sequent and x a fresh propositional variable not occurring in Γ. $\vdash \Gamma$ is provable in* **LL** *if and only if $\Gamma^{[x]} \vdash x$ is provable in* **ILL**.

Using this, we show the intuitionistic version of Theorem 28.[2]

▶ **Corollary 30** ([9, 12, 27]). *The problem of determining if a given !-prenex $\{\multimap\}$-sequent is provable in* **ILL** *is Ackermann-hard.*

[2] In [12] de Groote, Guillaume and Salvati provided a reduction from BVASS-reachability to provability of !-prenex $\{\multimap\}$-sequents in **ILL**. Thus one can also show Corollaries 30 and 31, using the results in [9, 12, 27].

Proof. By Theorem 28, it suffices to show that there is a reduction from the provability of ?-prenex $\{\otimes, \parr\}$-sequents in **LL** to the provability of !-prenex $\{\multimap\}$-sequents in **ILL**. The proof is essentially the same as that of Theorem 20, but we use Theorem 29 instead of Theorem 2. Let $\vdash ?\Gamma, \Delta$ be a ?-prenex $\{\otimes, \parr\}$-sequent and x a fresh propositional variable not in the sequent. By Theorem 29, $\vdash ?\Gamma, \Delta$ is provable in **LL** if and only if $(?\Gamma, \Delta)^{[x]} \vdash x$ is provable in **ILL**. Obviously, the latter sequent is a !-prenex implicational sequent. ◀

▶ **Corollary 31.** *The deducibility problem for* **BCI** *is* Ackermann*-hard.*

Proof. In view of Corollary 30, our goal is to show that: for any !-prenex $\{\multimap\}$-sequent $!\Gamma, \Delta \vdash C$, $\Delta \vdash C$ is provable in $\mathbf{BCI}[\sigma(\Gamma)]$ if and only if $!\Gamma, \Delta \vdash C$ is provable in **ILL**.

The proof of the *if* part goes by induction on the length of the cut-free proof of $!\Gamma, \Delta \vdash C$ in **ILL**. We only analyze the case where the rule of (!D) is applied last. If $!\Gamma, !A, \Delta \vdash C$ is obtained from $!\Gamma, A, \Delta \vdash C$ by an application of (!D), then by the induction hypothesis, $A, \Delta \vdash C$ is provable in $\mathbf{ILL}[\sigma(\Gamma)]$; thus it is also provable in $\mathbf{ILL}[\sigma(\Gamma, A)]$. Since $\vdash A$ is a non-logical axiom of $\mathbf{ILL}[\sigma(\Gamma, A)]$, we may construct a derivation of $\Delta \vdash C$ in $\mathbf{ILL}[\sigma(\Gamma, A)]$ by using (Cut). The remaining cases are similar.

The converse direction is shown by induction on the length of the derivation for $\Delta \vdash C$ in $\mathbf{BCI}[\sigma(\Gamma)]$. All cases are straightforward. ◀

Tower-hardness of light affine logic. We have also shown the Tower-completeness of provability in elementary affine logic. On the other hand, one can define another subsystem of affine logic, called *intuitionistic light affine logic* (**ILAL**) [1, 38], which characterizes polynomial time functions. We obtain it from the $\{\multimap, !\}$-fragment of **ILLW** by dropping the rules for (!D) and (!P), and by adding a new unary connective "§" and the following inference rules:

$$\frac{E \vdash A}{!E \vdash !A}\ (!) \qquad \frac{\Gamma, \Delta \vdash A}{!\Gamma, \S\Delta \vdash \S A}\ (\S)$$

where E is a formula or the empty multiset. It is known that provability in **ILAL** is decidable because Terui proved that it has the finite model property; see [38, Corollary 7.45]. We show that there is no elementary recursive algorithm for solving the provability problem for **ILAL**:

▶ **Lemma 32.** *Let* $!\Gamma, \Delta \vdash A$ *be a* !*-prenex* $\{\multimap\}$*-sequent.* $!\Gamma, \Delta \vdash A$ *is provable in the* $\{\multimap, !\}$*-fragment of* **ILLW** *if and only if* $!\Gamma, \S\Delta \vdash \S A$ *is provable in* **ILAL***.*

Proof. Our proof is inspired by the proof of the undecidability of provability in light linear logic by Terui; see [38, Section 2.3.2]. For any intuitionistic $\{\multimap, !, \S\}$-formula A, we inductively define the intuitionistic $\{\multimap, !\}$-formula A^- by $p^- = p$, $(B \multimap C)^- = B^- \multimap C^-$, $(!B)^- = !B^-$, and $(\S B)^- = B^-$. It is easy to see that if $\Sigma \vdash C$ is provable in **ILAL** then $\Sigma^- \vdash C^-$ is provable in the $\{\multimap, !\}$-fragment of **ILLW**, for any sequent $\Sigma \vdash C$ of **ILAL**. This is checked by induction on the size of proofs. Thus the *if* direction holds. The *only-if* direction follows from Claim (a) used in the proof of Lemma 21, the rule of (§), and the structural rules. ◀

By Theorem 20, the provability problem for **ILAL** is hard for Tower. We strongly believe that this problem is in Tower.

References

1 Andrea Asperti. Light affine logic. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, pages 300–308, 1998. doi:10.1109/LICS.1998.705666.

2 Patrick Baillot. On the expressivity of elementary linear logic: characterizing Ptime and an exponential time hierarchy. *Information and Computation*, 241:3–31, 2015. doi:10.1016/j.ic.2014.10.005.

3 A.R. Balasubramanian, Timo Lang, and Revantha Ramanayake. Decidability and complexity in weakening and contraction hypersequent substructural logics. In *Proceedings of the 36th Annual IEEE Symposium on Logic in Computer Science*, pages 1–13, 2021. doi:10.1109/LICS52264.2021.9470733.

4 Willem J. Blok and Clint J. van Alten. The finite embeddability property for residuated lattices, pocrims and BCK-algebras. *Algebra Universalis*, 48:253–271, 2002. doi:10.1007/s000120200000.

5 Willem J. Blok and Clint J. van Alten. On the finite embeddability property for residuated ordered groupoids. *Transactions of the American Mathematical Society*, 357(10):4141–4157, 2005. doi:10.1090/S0002-9947-04-03654-2.

6 Wojciech Buszkowski. Some decision problems in the theory of syntactic categories. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 28(33–38):539–548, 1982. doi:10.1002/malq.19820283308.

7 Wojciech Buszkowski. On the complexity of some substructural logics. *Reports on Mathematical Logic*, 43:5–24, 2008.

8 Paolo Coppola and Simone Martini. Optimizing optimal reduction: a type inference algorithm for elementary affine logic. *ACM Transactions on Computational Logic*, 7(2):219–260, 2006. doi:10.1145/1131313.1131315.

9 Wojciech Czerwiński and Łukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science*, pages 1229–1240, 2021. doi:10.1109/FOCS52979.2021.00120.

10 Ugo Dal Lago and Simone Martini. Phase semantics and decidability of elementary affine logic. *Theoretical Computer Science*, 318(3):409–433, 2004. doi:10.1016/j.tcs.2004.02.037.

11 Vincent Danos and Jean-Baptiste Joinet. Linear logic and elementary time. *Information and Computation*, 183(1):123–137, 2003. doi:10.1016/S0890-5401(03)00010-5.

12 Philippe de Groote, Bruno Guillaume, and Sylvain Salvati. Vector addition tree automata. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, pages 64–73, 2004. doi:10.1109/LICS.2004.1319601.

13 Francesc Esteva and Lluís Godo. Monoidal t-norm based logic: towards a logic for left-continuous t-norms. *Fuzzy Sets and Systems*, 124(3):271–288, 2001. doi:10.1016/S0165-0114(01)00098-7.

14 Nikolaos Galatos, Peter Jipsen, Tomasz Kowalski, and Hiroakira Ono. *Residuated Lattices: An Algebraic Glimpse at Substructural Logics*, volume 151 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2007.

15 Nikolaos Galatos and Hiroakira Ono. Algebraization, parametrized local deduction theorem and interpolation for substructural logics over FL. *Studia Logica*, 83:279–308, 2006. doi:10.1007/s11225-006-8305-5.

16 Jean-Yves Girard. Linear logic: its syntax and semantics. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Note Series*, pages 1–42. Cambridge University Press, 1995. doi:10.1017/CBO9780511629150.002.

17 Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998. doi:10.1006/inco.1998.2700.

18 Petr Hájek. *Metamathematics of Fuzzy Logic*, volume 4 of *Trends in Logic*. Springer, 1998. doi:10.1007/978-94-011-5300-3.

19 Zuzana Haniková. Computational complexity of propositional fuzzy logics. In Petr Cintula, Petr Hájek, and Carles Noguera, editors, *Handbook of Mathematical Fuzzy Logic (Volume 2)*, volume 38 of *Mathematical Logic and Foundations*, pages 793–851. College Publications, 2011.

20 Zuzana Haniková. Complexity of some language fragments of fuzzy logics. *Soft Computing*, 21:69–77, 2017. doi:10.1007/s00500-016-2346-0.

21 Rostislav Horčík and Kazushige Terui. Disjunction property and complexity of substructural logics. *Theoretical Computer Science*, 412(31):3992–4006, 2011. doi:10.1016/j.tcs.2011.04.004.

22 Kiyoshi Iséki and Shôtarô Tanaka. An introduction to theory of BCK-algebras. *Mathematica Japonica*, 23:1–26, 1978.

23 Max I. Kanovich. The complexity of Horn fragments of linear logic. *Annals of Pure and Applied Logic*, 69(2–3):195–241, 1994. doi:10.1016/0168-0072(94)90085-X.

24 Yves Lafont. The finite model property for various fragments of linear logic. *Journal of Symbolic Logic*, 62(4):1202–1208, 1997. doi:10.2307/2275637.

25 Olivier Laurent. Around classical and intuitionistic linear logics. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 629–638, 2018. doi:10.1145/3209108.3209132.

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

27 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science*, pages 1241–1252, 2021. doi:10.1109/FOCS52979.2021.00121.

28 Patrick Lincoln, John Mitchell, Andre Scedrov, and Natarajan Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56(1–3):239–311, 1992. doi:10.1016/0168-0072(92)90075-B.

29 Mitsuhiro 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. doi:10.1016/S0304-3975(02)00024-5.

30 Mitsuhiro Okada and Kazushige Terui. The finite model property for various fragments of intuitionistic linear logic. *Journal of Symbolic Logic*, 64(2):790–802, 1999. doi:10.2307/2586501.

31 Hiroakira Ono. Substructural logics and residuated lattices — an introduction. In Vincent F. Hendricks and Jacek Malinowski, editors, *Trends in Logic: 50 Years of Studia Logica*, volume 21 of *Trends in Logic*, pages 193–228, 2003. doi:10.1007/978-94-017-3598-8_8.

32 Hiroakira Ono and Yuichi Komori. Logics without the contraction rule. *Journal of Symbolic Logic*, 50(1):169–201, 1985. doi:10.2307/2273798.

33 Mati Pentus. Lambek calculus is NP-complete. *Theoreitcal Computer Science*, 357(1–3):186–201, 2006. doi:10.1016/j.tcs.2006.03.018.

34 Greg Restall. *An Introduction to Substructural Logics*. Routledge, 2000.

35 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Transactions on Computation Theory*, 8(1):1–36, 2016. doi:10.1145/2858784.

36 Sylvain Schmitz. Implicational relevance logic is 2-EXPTIME-complete. *Journal of Symbolic Logic*, 81(2):641–661, 2016. doi:10.1017/jsl.2015.7.

37 Richard Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoreical Computer Science*, 9(1):67–72, 1979. doi:10.1016/0304-3975(79)90006-9.

38 Kazushige Terui. *Light Logic and Polynomial Time Computation*. PhD thesis, Keio University, 2002.

39 Kazushige Terui. Light affine set theory: a naive set theory of polynomial time. *Studia Logica*, 77:9–40, 2004. doi:10.1023/B:STUD.0000034183.33333.6f.

40 Anne Sjerp Troelstra. *Lectures on Linear logic*, volume 29 of *CSLI Lecture Notes*. Center for the Study of Language and Information, 1992.

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

Dynamic Complexity of Regular Languages: Big Changes, Small Work

Felix Tschirbs
Ruhr-Universität Bochum, Germany

Nils Vortmeier
Ruhr-Universität Bochum, Germany

Thomas Zeume
Ruhr-Universität Bochum, Germany

Abstract

Whether a changing string is member of a certain regular language can be maintained in the DynFO framework of Patnaik and Immerman: after changing the symbol at one position of the string, a first-order update formula can express – using additionally stored information – whether the resulting string is in the regular language.

We extend this and further known results by considering changes of many positions at once. We also investigate to which degree the obtained update formulas imply work-efficient parallel dynamic algorithms.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Logic and databases

Keywords and phrases dynamic descriptive complexity, regular languages, batch changes, work

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.35

Acknowledgements We are grateful to Jonas Schmidt and Thomas Schwentick for insightful discussions.

1 Introduction

Which queries allow for efficient, parallel updates of their results under changes to their input? Dynamic descriptive complexity theory is a fundamental framework for addressing this question, which was proposed independently by Patnaik and Immerman [17] as well as by Dong and Su [8] in the early nineties. In their frameworks, dynamic update programs formalized via first-order formulas are used for updating both query results and helpful auxiliary data after changes to an input structure. Queries maintainable in this fashion are said to be in the class DynFO and, due to classical correspondences [25], can also be maintained in constant time by parallel random-access machines with polynomially many processors.

The class DynFO is surprisingly powerful. It is known, for instance, that all context-free languages [10, Theorem 4.1] as well as the reachability query [4, Theorem 1] can be maintained by first-order update programs, and that all properties expressible in monadic second-order logic can be maintained assuming that the input structure retains bounded treewidth [6, Theorem 6.1]. All these results have been stated solely for changes that modify a single tuple. Also, although these results imply that updates can be performed in constant parallel time, they usually do not consider the necessary work, that is, the total number of necessary computation steps, to perform such an update. A DynFO program that uses first-order formulas of quantifier-depth ℓ to update k-ary auxiliary relations over a domain of size n essentially requires work $n^{\ell+k}$ if it is evaluated naïvely. In fact, many DynFO programs require at least quadratic work, some updates for important queries as graph reachability have a polynomial work bound with much higher degree.

Towards practical applications, therefore two additional aspects need to be taken into account: (1) How to deal with batch changes of non-constant size?, and (2) How much work is required for updates after changes?

Several results for maintaining queries under batch changes have been obtained recently. Reachability is in DynFO under changes of $\mathcal{O}(\frac{\log n}{\log \log n})$ many edges [7, Theorem 1], and even under changes of size $\mathcal{O}(\log^c n)$, for any $c \in \mathbb{N}$, for undirected graphs or for insertions only [5, Theorems 6 and 7]. For declaratively specified batch changes, it is known that undirected reachability can be maintained under first-order defined insertions and acyclic reachability under insertions defined by quantifier-free formulas [22, Theorems 4.2 and 4.5]. Context-free languages are maintainable under Σ_1-defined changes [21, Theorem 9].

For work-efficient dynamic first-order programs, not much has been done so far. A framework for studying work-efficient DynFO is introduced in [18]. There it is shown that under changes of single positions it is possible to maintain regular languages with $\mathcal{O}(n^\epsilon)$ work for all $\epsilon > 0$, and that star-free languages and certain context-free (non-regular) languages can be maintained with polylogarithmic work.

The focus of this paper is to study these two aspects for the membership problem for regular languages, and in particular to explore how to deal with batch changes with as little work as possible. Our main contributions are:

- Regular (string) languages and regular tree languages can be maintained under changes of polylogarithmic size. For regular string languages, only work $\mathcal{O}(n^\epsilon)$ is necessary, for all $\epsilon > 0$.
- Star-free (string) languages can be maintained under changes of polylogarithmic size with polylogarithmic work. No significant improvement is possible with respect to the class of languages.

The main proof tool for the upper bounds, already used in [5], is to exploit the power of first-order formulas on small substructures. Intuitively, on small substructures, first-order quantifiers are as powerful as (restricted) second-order quantifiers. Exploiting this yields that first-order formulas are as powerful as LOGCFL on substructures of polylogarithmic size – a fact previously observed in the context of constant-depth circuits of size 2^{n^ϵ} in [1, cf. Lemma 8.1]. For maintaining queries under changes of polylogarithmic size, this can now be used by performing LOGCFL computations on structures defined on the elements affected by the change. We also investigate the expressive power of second-order quantification over relations of bounded size.

We infer the lower bounds from classical lower bounds from circuit complexity.

Parts of these results have been included in the PhD thesis of the second author [26].

Outline. After recalling essential notions from descriptive dynamic complexity theory in Section 2, we discuss how the expressivity of first-order formulas on small substructures can be exploited for dealing with large changes to a structure in Sections 3 and 6. We then introduce our techniques by establishing that regular languages can be maintained under changes of polylogarithmic size in Section 4, and discuss how to generalize these techniques to dynamic programs that use little work in Section 5.

2 The dynamic setting

We introduce the main notions of the dynamic complexity framework, following [23, 18]. We slightly adapt the framework to take into account both batch changes and the required amount of work.

Preliminaries. A schema σ is a set of relation symbols and function symbols, each with a corresponding arity. A relational structure $\mathcal{S}$ over some schema σ consists of a finite domain D, a relation $R^{\mathcal{S}} \subseteq D^k$ for each k-ary relation symbol $R \in \sigma$ and a function $f^{\mathcal{S}} \colon D^k \to D$ for every k-ary function symbol $f \in \sigma$. All structures considered in this work are equipped with a linear order $\leq$ on their domain D, so we assume that D is an initial segment $\{0, \dots, n-1\}$ of the natural numbers.

Words over an alphabet Σ are represented by structures over the schema that includes $\leq$ and a relation R_σ for every $\sigma \in \Sigma$: domain elements represent the positions of the word, and $i \in R_\sigma$ means that position i carries the symbol σ. A position i may be included in at most one relation R_σ; if it is not included in any such relation we say that it is *empty* and labeled with ϵ. Labeled trees have an additional relation E such that $(u, v) \in E$ holds exactly if v is the parent of u.

First-order logic FO is defined in the usual way, we refer to [15] for details. We allow if-then-else constructs in terms: if t_1 and t_2 are terms and φ is a formula, then $\mathrm{ITE}(\varphi, t_1, t_2)$ is a term. It evaluates to the valuation of t_1 if φ is satisfied, and to the valuation of t_2 otherwise. Throughout this work we allow first-order formulas to access the linear order $\leq$ on the structures and relations $+$ and $\times$ that encode addition and multiplication on the domain D, and write $\mathsf{FO}(\leq, +, \times)$ to make this explicit.

Circuits and complexity classes. First-order logic with the numeric relations $\leq, +, \times$ is equivalent to (first-order uniform) AC^0, the class of problems that are decided by uniform families of constant-depth circuits with polynomially many "and"-, "or"- and "not"-gates, where the former two may have unbounded fan-in. The complexity class LOGCFL contains all problems that can be reduced in logarithmic space to a context-free language. It includes NL and is included in AC^1.

Dynamic problems. In our setting, a dynamic problem is characterized by (1) the schema σ_{in} of an input structure, and (2) the set Δ of allowed change and query operations.

An example is the dynamic membership problem for some language L:

- The input structure is a word $w = w_0 \cdots w_{n-1}$ of some length n over the alphabet Σ of L.
- Change operations have the form $\textsc{set}_\sigma(P)$, for $\sigma \in \Sigma \cup \{\epsilon\}$. A concrete such change has as parameter a set P of positions and is applied by setting the symbols for all positions $i \in P$ to σ.
- The operation $\textsc{member}$ allows querying whether the current word is in L, where empty positions are ignored. So, it returns whether $w_0 \circ \cdots \circ w_{n-1} \in L$ holds.

For the purpose of this paper, a change operation in general has the form $\delta(P)$, where P is a relation variable. This definition reflects that we are interested in changes of many elements at once. We use change operations that restrict the allowed size of a parameter relation P, as allowing the input to change arbitrarily basically turns a dynamic problem into a static problem, as one needs to solve the problem from scratch any time the input is replaced. Mostly we are interested in changes that insert or delete a polylogarithmic number of tuples with respect to the domain size.

To ensure that we are able to access changed elements with little work, so, without enumerating over all elements, we demand that each change $\delta(P)$ for a unary relation P comes with a function $\textsc{chd} \colon D \to D$ that maps the i-th element of the (linearly ordered) domain D to the i-th element in P according to the order on D, for all $i \leq |P|$. If P is a k-ary relation, there are k functions $\textsc{chd}_1, \dots, \textsc{chd}_k$ that map to the first, …, k-th element in the i-th tuple in P according to the lexicographic ordering.

Query operations have the form $Q(\bar{p})$, where $\bar{p}$ is a parameter tuple of element variables. For instance, the MEMBER query from above is a query operation without parameters. An example for a query operation on words with parameters is $\text{RANGE}(\ell, r)$, which receives two positions ℓ and r as parameters and asks whether $w_\ell \circ \cdots \circ w_r$ is in L. We also consider query operations that return domain elements instead of a truth value. For example, consider a dynamic problem NEXTINSET that has as input a set $S \subseteq \{1, \ldots, n\}$, change operations $\text{INS}(P)$ and $\text{DEL}(P)$ that respectively insert a set $P \subseteq \{1, \ldots, n\}$ of numbers into S and delete such a set from S, and query operations $\text{PRED}(i)$ and $\text{SUCC}(i)$, where $\text{PRED}(i)$ returns the predecessor of i in S (so, the largest element of S that is smaller than i) and $\text{SUCC}(i)$ returns its successor (the smallest element of S that is larger than i).

Maintenance of dynamic problems. The goal of a *dynamic program* $\mathcal{P}$ is to maintain some dynamic problem Π: it must be able to answer queries to the input structure which is allowed to change according to the change operations. To do so, $\mathcal{P}$ stores and updates an auxiliary structure $\mathcal{A}$ over some schema σ_{aux} and over the same domain as the input structure. This structure $\mathcal{A}$ consists of a set of auxiliary relations and auxiliary functions.

A first-order dynamic program expresses an update of its auxiliary structure as well as the answer to a query by means of first-order formulas and terms. For each query operation Q with parameters $\bar{p}$ it has a *query rule* of the form **on query** $Q(\bar{p})$ **yield** $\varphi_Q(\bar{p})$, where φ_Q is a (first-order) *query formula* or *query term*, depending on the type of the result of Q, over the combined schema $\sigma_{\text{in}} \cup \sigma_{\text{aux}}$ of input and auxiliary structure.

For each change operation $\delta(P)$ and each auxiliary relation symbol $R \in \sigma_{\text{aux}}$ with arity k the dynamic program has an *update rule* of the form

on change $\delta(P)$ **update** R **at** $(t_1(\bar{x}), \ldots, t_k(\bar{x}))$ **as** $\varphi^R_\delta(P; \bar{x})$ **for all** $\varphi_C(\bar{x})$.

Here, $\varphi^R_\delta(P; \bar{x})$ is a (first-order) *update formula* and $t_1, \ldots, t_k$ are first-order terms, all over the schema $\sigma_{\text{in}} \cup \sigma_{\text{aux}} \cup \{P, \text{CHD}\}$, and $\varphi_C(\bar{x})$ is a *constraint formula* for the tuple $\bar{x} = x_1, \ldots, x_\ell$ of variables. The constraint formula is a conjunction of inequalities $x_i \leq f_i(n)$ for $i \in \{1, \ldots, \ell\}$, using $\mathsf{FO}(\leq, +, \times)$-definable functions $f_i : \mathbb{N} \to \mathbb{N}$.

After a change $\delta(P)$ to an input structure $\mathcal{I}$ with current auxiliary structure $\mathcal{A}$, the effect of an update on the auxiliary relation R is as follows. Let $\mathcal{I}'$ be the input structure after the change. The new relation $R^{\mathcal{A}'}$ in the updated auxiliary structure $\mathcal{A}'$ contains all (old) tuples $\bar{a} \in R^{\mathcal{A}}$ such that $\bar{a}$ is *not* equal to $(t_1(\bar{b}), \ldots, t_k(\bar{b}))$ for any tuple $\bar{b}$ that satisfies φ_C. Additionally, $R^{\mathcal{A}'}$ contains all tuples $(t_1(\bar{b}), \ldots, t_k(\bar{b}))$ such that $\bar{b}$ satisfies φ_C and $\varphi^R_\delta(\bar{b})$ holds in $(\mathcal{I}', \mathcal{A}, P, \text{CHD})$.

Phrased differently, φ_C is used to enumerate all tuples $\bar{b}$ such that containment of $(t_1(\bar{b}), \ldots, t_k(\bar{b}))$ in R may change. Whether that tuple is inserted into R (if it was not already present) or deleted from R (if it was already present) depends on the evaluation of the update formula $\varphi^R_\delta(\bar{b})$ on the changed input structure and the old auxiliary structure, given the information on the change provided by P and CHD.

Update rules for auxiliary functions are defined analogously, but use an update term (instead of $\varphi^R_\delta(P; \bar{x})$) that determines the new function value for a tuple $(t_1(\bar{b}), \ldots, t_k(\bar{b}))$ such that $\bar{b}$ satisfies the constraint.

We say that a dynamic program $\mathcal{P}$ *maintains* a dynamic problem Π, if it can always answer all queries correctly. More precisely, $\mathcal{P}$ maintains Π if applying a query operation after a sequence α of change operations on an initial structure $\mathcal{I}_0$ yields the same result as evaluating the corresponding query rule on the combined input and auxiliary structure that is obtained by applying the update rules corresponding to α to $(\mathcal{I}_0, \mathcal{A}_0)$, where $\mathcal{A}_0$ is

an initial auxiliary structure. Following Patnaik and Immerman [17], we demand that the initial input structure $\mathcal{I}_0$ is *empty*, so, has empty relations and all function values being 0. The initial auxiliary structure is over the same domain as $\mathcal{I}_0$ and is defined from $\mathcal{I}_0$ by some first-order definable initialization.

The class DynFO is the class of all dynamic problems that are maintained by some first-order dynamic program.

The work of a dynamic program. It is not immediately clear how to measure the work of a dynamic program, as first-order logic is declarative. Schmidt et al. [18] introduce a detailed framework that allows for a comparatively easy way to specify updates and queries, as well as to associate a work bound. This is done via constant-time PRAMs that implement a first-order dynamic program, for which the amount of work is well-defined as the total number of the steps of the processors.

The framework of [18] introduces more intricate syntax elements for specifying dynamic programs that allow in some cases for a translation into more efficient PRAMs. For example, in the syntax of [18] one can distinguish parts of an update formula that need to be evaluated for every tuple $(t_1(\bar{x}), \dots, t_k(\bar{x}))$ and parts that only need to be evaluated once.

In this paper, we are only interested in work bounds of the form "polylogarithmic in n" or "$\mathcal{O}(n^\epsilon)$ for arbitrary $\epsilon > 0$", so single logarithmic factors are less important. This allows us to be a bit coarser.

For us, the work of a first-order formula is just the number of wires in the AC^0 circuit that is obtained by the straight-forward translation of first-order formulas into bounded-depth circuits. We allow quantifiers of the form $\exists x \leq f(n)$ and $\forall x \leq f(n)$, for $\mathsf{FO}(\leq, +, \times)$-definable functions $f\colon \mathbb{N} \to \mathbb{N}$, to prevent that quantifiers always introduce a factor of n for the work bound.

The work of a query rule is the work of its query formula. The work of an update rule with constraint formula $\varphi_C(\bar{x}) = x_1 \leq f_1(n) \wedge \dots \wedge x_\ell \leq f_\ell(n)$ is the product of the functions f_i and the work of the update formula, as the update formula needs to be evaluated $f_1(n) \cdot \dots \cdot f_\ell(n)$ times.

An example: NextInSet. To illustrate the setting, and because we use this result in Section 5, we show that NextInSet can be maintained under changes of polylogarithmic size with a polylogarithmic amount of work. Recall that NextInSet provides two queries pred(i) and succ(i) that return the predecessor respectively the successor of i in a linearly ordered set S that is subject to insertions and deletions.

▶ **Lemma 1.** *For every $c \in \mathbb{N}$ there is a $d \in \mathbb{N}$ such that* NextInSet can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under changes of size $\log^c n$ with $\mathcal{O}(\log^d n)$ work.

Proof. We adapt the proof of [18, Lemma 4.1] that shows that NextInSet can be maintained in DynFO with work $\mathcal{O}(\log n)$ under insertions and deletions of single elements. The dynamic program maintains a complete binary tree with ordered leaves $\{1, \dots, |D|\}$, where D is the domain, including a function anc such that anc(x, k) returns the k-th ancestor of x in the tree. We associate with every tree node an interval according to the leaves in its subtree. We say that a node *covers* an element if it is in the interval associated with the node. The auxiliary structure stores for each node the minimum and maximum element of S that is covered by the node. A succ(i) or pred(i) query can be answered with $\mathcal{O}(\log n)$ work using this information (cf. [18]).

To update the auxiliary information when $\log^c n$ elements are inserted or deleted, one can enumerate the $\log^c n \cdot \log n$ tree nodes for which the information needs to be updated: all nodes that lie on the path from a leave corresponding to a changed element to the root of the tree. For each such node x, the new maximum is calculated as follows:

- **Insertions.** First, find the largest inserted element that is covered by x: the inserted element i that is covered by x and is larger than any other inserted element covered by x. This element can be identified with $\log^{2c} n$ work. Compare i with the current maximum of x and update the maximum if necessary.
- **Deletions.** If the current maximum is deleted, use $\textsc{pred}(i)$ to query the predecessor for each deleted element i covered by x, using $\mathcal{O}(\log n)$ work per query. By pairwise comparison, find the largest such predecessor that is not itself deleted. If it is covered by x, it is the new maximum, otherwise there is none.

The minimum of x is updated symmetrically. In total, an update can be performed using polylogarithmic work.

Formally, the update rule for the auxiliary function max that maps an inner node x to the maximal element of S covered by x is as follows:

$$\begin{aligned}&\textbf{on change } \textsc{ins}(P) \textbf{ update } \max \textbf{ at } \operatorname{anc}(\textsc{chd}(x_1), x_2) \textbf{ as } \varphi^{\max}_{\textsc{ins}}(x_1, x_2)\\ &\qquad\qquad \textbf{for all } x_1 \leq \log^c n \wedge x_2 \leq \log n\end{aligned}$$

Here, we address an inner node x as the x_2-th tree-ancestor of the x_1-th changed element. The update term $\varphi^{\max}_{\textsc{ins}}(x_1, x_2)$ is defined as $\mathrm{ITE}(\max_{\text{ins}} > \max_{\text{cur}}, \max_{\text{ins}}, \max_{\text{cur}})$, where $\max_{\text{cur}} = \max(\operatorname{anc}(\textsc{chd}(x_1), x_2))$ gives the current maximum for node x, and $\max_{\text{ins}}$ is the maximal changed element $\textsc{chd}(x_i)$ such that the formula $\operatorname{anc}(\textsc{chd}(x_1), x_2) = \operatorname{anc}(\textsc{chd}(x_i), x_2)$ is satisfied. ◀

3 First-order logic on substructures of polylogarithmic size

When a string is changed in polylogarithmically many positions, first-order update formulas for maintaining some language L have to be able, at a minimum, to decide for strings of this size whether they are a member of L. This is necessary for dealing with polylogarithmically many changes of subsequent positions. In the more general case that the changes are not subsequent, changes of polylogarithmic size partition a string into as many pieces. Known information about these pieces needs then to be combined by first-order update formulas to decide membership in L for the new string.

Thus, for maintaining properties under changes of polylogarithmic size, it is helpful to know the expressive power of first-order formulas on small substructures of this size.

We use the well-known fact that many complexity classes can be captured by bounded-depth circuits of subexponential size.

For the moment we simply state the result and infer a corollary suitable for our needs. Later, in Section 6, we will re-visit these results from another perspective.

▶ **Lemma 2** ([1, cf. Lemma 8.1]). *For every language $L \in \mathsf{LOGCFL}$ and every $\epsilon > 0$ there is a d such that L can be decided by depth-d circuits of size 2^{n^ϵ}.*

It follows that first-order formulas can express LOGCFL-computable queries on substructures of polylogarithmic size, which we formalize in Corollary 3. See also [5, Theorem 3] for an analogous statement regarding NL-computable queries. Corollary 3 requires that the circuits from Lemma 2 can be constructed uniformly, which is implicit in the proof given by [1] and will be made explicit in our discussion in Section 6.

▶ **Corollary 3.** *Let k and c be arbitrary natural numbers, and let Q be a k-ary,* LOGCFL*-computable query on σ-structures. There is an* FO$(\leq, +, \times)$ *formula φ over schema $\sigma \cup \{C\}$ such that for any σ-structure $\mathcal{S}$ with n elements, any subset C of its domain of size at most $\mathcal{O}(\log^c n)$, and any k-tuple $\bar{a} \in C^k$ it holds that*

$$\bar{a} \in Q(\mathcal{S}[C]) \text{ if and only if } (\mathcal{S}, C) \models \varphi(\bar{a}).$$

Here, $\mathcal{S}[C]$ denotes the substructure of $\mathcal{S}$ induced by C. Moreover, there is an equivalent AC^0 *circuit of size $\mathcal{O}(n^\epsilon)$, for every ϵ with $0 < \epsilon < 1$.*

Proof. Let Q, c and ϵ be fixed with $0 < \epsilon < 1$. The bounded-depth circuits that are guaranteed to exist by Lemma 2 for $\hat{\epsilon} = \frac{\epsilon}{c}$ have polynomial size $2^{(\log n)^\epsilon} = \mathcal{O}(n^\epsilon)$, so, are AC^0-circuits. As every AC^0-circuit has an equivalent FO$(\leq, +, \times)$-formula, the statement follows. ◀

4 Regular languages under many changes

We show in this section that regular languages of strings and trees can be maintained in DynFO. For now, we will not aim for work-efficient updates; this aspect will be treated in Section 5. We prove these intermediate results also to introduce the techniques.

For both string and tree languages our approach is similar. Every change affects a polylogarithmic number of elements of the underlying string or tree, respectively. We enrich the substructure that is induced by these affected elements by information from the maintained auxiliary relations. Thanks to Corollary 3, the first-order update formulas can express any query from LOGCFL on this substructure, which provides sufficient information to tell whether the full structure is a member of the regular language.

▶ **Theorem 4.** *One can maintain in* DynFO$(\leq, +, \times)$
(a) *membership in any regular (string) language, and*
(b) *membership in any regular tree language*
under changes of $\log^c n$ many symbols, for any fixed $c \in \mathbb{N}$, where n is the size of the underlying string respectively tree.

Proof. Towards (a), fix a regular language L over some alphabet Σ, and let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a DFA for L.

To maintain membership in L under changes of single positions, the dynamic program by Gelade et al. [10] uses auxiliary relations of the form $S_{p,q}(k, \ell)$ for all states p and q of $\mathcal{A}$. It maintains that pairs (k, ℓ) of positions are in $S_{p,q}$ if and only if $\delta^*(p, w_k \cdots w_\ell) = q$. Here, $w = w_1 \cdots w_n$ is the input word and δ^* is the extension of the transition function δ from symbols $\sigma \in \Sigma$ to words from Σ^*. Clearly, $w \in L$ if and only if $(1, n) \in S_{q_0,q_f}$ for some accepting state $q_f \in F$ of $\mathcal{A}$; and this condition can easily be expressed by a first-order formula.

We show that these auxiliary relations can be maintained in DynFO$(\leq, +, \times)$ under changes of polylogarithmic size. In a nutshell, we show that updating these auxiliary relations after such changes boils down to computing the product of polylogarithmically many elements of the transition monoid of $\mathcal{A}$. This product can be computed in FO$(\leq, +, \times)$ by Corollary 3, as it can be done in $\mathsf{NC}^1 \subseteq$ LOGCFL.

We make this approach more precise now. When $w = w_1 \cdots w_n$ is changed to $w' = w'_1 \cdots w'_n$, the auxiliary relations $S_{p,q}$ are updated as follows. An interval (k, ℓ) is in the new auxiliary relation $S_{p,q}$ if and only if $\delta^*(p, w'_k \cdots w'_\ell) = q$. The update formula, for each interval (k, ℓ),

(1) splits $w'_k \cdots w'_\ell$ into at most $\mathcal{O}(\log^c n)$ pieces $u_0, \ldots, u_m$ according to the changes;
(2) assigns to each piece u_j its corresponding element $\gamma_j : Q \to Q$ of the transition monoid of $\mathcal{A}$, i.e., the function γ_j that maps $p' \in Q$ to $q' \in Q$ if $\mathcal{A}$ transitions from p' to q' when reading u_j;
(3) computes the product $\gamma \stackrel{\text{def}}{=} \gamma'_m \circ \cdots \circ \gamma'_0$; and
(4) lets $(k, \ell) \in S_{p,q}$ if and only if $\gamma(p) = q$.

More precisely, suppose the change operation affected the positions $\{i_1, \ldots, i_m\}$ in the interval (k, ℓ). For (1), define $P \stackrel{\text{def}}{=} \{i_0, i_1, \ldots, i_m, i_{m+1}\}$, where $i_0 = k$ and $i_{m+1} = \ell + 1$. For each $0 \leq j \leq m$, the piece u_j is defined as the substring $w'_{i_j} \cdots w'_{i_{j+1}-1}$ of $w'_{i_1} \cdots w'_{i_m}$ that starts with a changed position i_j and ends just before the next changed position (except for the border cases). For (2), the element γ_j of the transition monoid for $\mathcal{A}$ on u_j can be inferred from the stored auxiliary relations. The computation in (3) can be performed in $\mathsf{FO}(\leq, +, \times)$ due to Corollary 3.

Towards (b), fix a regular tree language L. We assume that input trees T are represented as structures with relations FirstChild and NextSibling with the obvious meaning. This is without loss of generality, as these relations can be defined by first-order formulas for any ordered tree. Using these relations, we can regard T to be a binary tree. It is well-known (see for example [16, Lemma 1]) that any regular tree language over ranked or unranked trees is accepted by a (finite deterministic bottom-up) tree automaton that reads the FirstChild-NextSibling encoding of the tree. Such a tree automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, with state set Q, transition function $\delta\colon Q \times Q \times \Sigma \to Q$, initial state q_0 and set F of accepting states, assigns in a bottom-up run to a leaf with symbol σ the state $\delta(q_0, q_0, \sigma)$ and to an inner node with symbol σ and children labeled with states p, q, respectively, the state $\delta(p, q, \sigma)$. It accepts if the state assigned to the root is in F.

To maintain membership in L under changes of a single node label, Gelade et al. in [10] mainly use binary auxiliary relations that are again of the form $S_{p,q}(u, v)$, for states $p, q \in Q$. Intuitively, $(u, v) \in S_{p,q}$ means that $\mathcal{A}$ assigns the state p to the node u, if the state for node v is forced to be q – no matter which state $\mathcal{A}$ actually assigns to v based on its subtree and the transition function δ.

We show that these auxiliary relations can be updated after changes of the labels of polylogarithmically many nodes. For this we assume the existence of the descendant relation on the tree nodes, which can be made available as an auxiliary relation. Our strategy is analogous to part (a): we show that first-order formulas can define a labeled binary tree with polylogarithmically many nodes such that from the information whether this tree is a member of some other regular tree language we can infer whether the changed input tree is in L, and how the other auxiliary relations need to be updated. Thanks to Corollary 3, this approach can be implemented by $\mathsf{FO}(\leq, +, \times)$ update formulas, as regular tree language membership (for deterministic bottom-up automata) is in $\mathsf{NC}^1 \subseteq \mathsf{LOGCFL}$.

We give some more details. Let P be the set of at most $\mathcal{O}(\log^c n)$ nodes whose labels are modified by a change, resulting in an input tree T'. Without loss of generality, we assume that for two nodes $v, v' \in P$ also their lowest common ancestor is in P, and that if for a node $v \in P$ a descendant from its left or right subtree is contained in P, also some descendant from its other subtree is in P. So, with the help of the descendant relation on T, we can define in FO a binary tree T_P with node set P and edges from a node to its "nearest descendants" in T.

We assign labels to the nodes of T_P according to the behavior of $\mathcal{A}$ on the subtree of T' that is rooted in the respective node. If $v \in P$ is a leaf of T_P, then its label is just the state $q \in Q$ that $\mathcal{A}$ assigns to this node in T'. As the only difference in the subtree rooted at v

between T and T' is the label of v itself, this state can easily be expressed using the old auxiliary relations and the transition function of $\mathcal{A}$. If $v \in P$ is an inner node of T_P, its label is a function $\gamma\colon Q \times Q \to Q$. Intuitively, this labels says that if $\mathcal{A}$ assigns in T' some states p, q to the nodes u_1, u_2 that are the children of v in T_P, then it assigns the state $\gamma(p, q)$ to v in T'. This label can be expressed in first-order logic from the old auxiliary relations as follows. Let v_1, v_2 be the children of v in T such that u_1 is a descendant of v_1 and u_2 is a descendant of v_2. If the state p is assigned to u_1 in T', then the state p' is assigned to v_1 that satisfies $(u_1, v_1) \in S_{p,p'}$. Symmetrically one can determine the state q' that is assigned to v_2 in T'. The state that is assigned to v can then be read from the transition function of $\mathcal{A}$.

Another bottom-up tree automaton can "evaluate" T_P in the natural way. By Corollary 3, an $\mathsf{FO}(\leq, +, \times)$ update formula can determine the state that this automaton assigns to every inner node, and in particular the state $q' \in Q$ that it assigns to the root of T_p. From this state, the update formula can determine the state q that $\mathcal{A}$ assigns to the root of T', and therefore can check whether $T' \in L$. The auxiliary relations of the form $S_{p,q}$ can be updated similarly. ◀

5 Regular languages, big changes, small work

So far we have seen that membership in regular languages can be maintained under polylogarithmic changes. In this section, we will re-visit this result and analyze the required work. We employ the framework for work-efficient dynamic complexity introduced in Section 2.

In [18] it was shown that regular languages can be maintained with work $\mathcal{O}(n^\epsilon)$ under single-tuple changes. We first lift this result to changes of polylogarithmic size. After arguing why polylogarithmic work cannot be achieved for polylogarithmic changes, we will then take a look at first-order definable regular languages. Here, again, we lift a result from [18] and show that such languages can be maintained with polylogarithmic work under changes of polylogarithmic size. It remains open whether membership in regular languages can be maintained in DynFO with $\mathcal{O}(\log^d n)$ work for some d under single tuple changes; in Section 7 we will discuss why proving barriers likely requires new insights.

We start by analyzing the work required for maintaining membership in regular languages under polylogarithmic changes.

▶ **Theorem 5.** *One can maintain membership in regular languages in* $\mathsf{DynFO}(\leq, +, \times)$ *with work* $\mathcal{O}(n^\epsilon)$ *under changes of size* $\log^c n$*, for* $c \in \mathbb{N}$ *and all* $\epsilon > 0$.

Proof. In Theorem 4, membership of a regular language L was maintained by storing, for each pair (k, ℓ) of positions of the input string w, the behavior of a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ accepting L when reading $w_k \cdots w_\ell$. When a symbol is changed, $\Theta(n^2)$ such pairs are affected in the worst case, resulting in an at least quadratic amount of work for processing large changes.

Schmidt et al. [18] introduced a technique for decreasing the amount of work to $\mathcal{O}(n^\epsilon)$, for all $\epsilon > 0$. The technique can be lifted to polylogarithmic changes in a straightforward fashion. For the sake of completeness we sketch the idea.

The idea in Schmidt et al. [18] is to store elements of the transition monoid of the automaton only for some well-chosen subwords called special intervals, such that each position is contained in only $\mathcal{O}(n^{\epsilon'})$ special intervals, for any $\epsilon' > 0$. These special intervals are arranged in a hierarchy whose depth depends solely on ϵ'. It is then shown that the element of the transition monoid for each (not necessarily special) interval can be computed bottom-up with the help of a constant number of special intervals.

We use the same hierarchy of intervals for polylogarithmic changes and apply the strategy from the proof of Theorem 4. Changes are now distributed to intervals of the hierarchy. As each changed symbol affects $\mathcal{O}(n^{\epsilon'})$ special intervals, $\mathcal{O}(\log^c n \cdot n^{\epsilon'})$ special intervals need to be updated.

For each special interval that contains at least one changed position, we execute steps (1)–(4) from the proof of Theorem 4. Step (1) splits the interval into pieces, each containing exactly one changed position; this requires $\mathcal{O}(\log^c n)$ work. Step (2) computes the transition monoid element of each piece; this requires constant work per piece. Step (3) computes the product of up to $\mathcal{O}(\log^c n)$ monoid elements; this can be done in $\mathcal{O}(n^{\epsilon''})$ work, for any $\epsilon'' > 0$ by Corollary 3. Step (4) requires a constant amount of work. This sums up to $\mathcal{O}(n^{\epsilon''})$ work needed to update one pair (k, ℓ) in a relation $S_{p,q}$.

In total, $\mathcal{O}(\log^c n \cdot n^{\epsilon'} \cdot n^{\epsilon''}) = \mathcal{O}(n^{\epsilon})$ work is needed to process an update of size $\log^c n$, for suitable choices of ϵ', ϵ''. ◀

We now show that one cannot do much better than stated in the previous theorem: there are regular languages which cannot be maintained with polylogarithmic work under polylogarithmic changes. Even more, we will exactly characterize the languages that can be maintained with polylogarithmic work under such changes.

A language is *star-free* if it can be expressed by a regular expression without Kleene star but with negation. Star-free languages have many equivalent characterizations: A language is star-free if and only if it is first-order definable (with only $<$) if and only if its syntactic monoid is aperiodic [20]. The *syntactic monoid* of L is the monoid with the least number of elements that recognizes L. Here, a monoid M *recognizes* a language $L \subseteq \Sigma^*$ if there is a morphism $h\colon \Sigma^* \to M$ and a set $F \subseteq M$ such that $L = h^{-1}(F)$. A monoid M is *aperiodic* if there is a k such that $m^k = m^{k+1}$ for all $m \in M$.

▶ **Theorem 6.** *For a regular language L, the following are equivalent:*

(a) *L is star-free.*

(b) *For all $c \in \mathbb{N}$ and some $d \in \mathbb{N}$ depending on c, L can be maintained in* $\mathsf{DynFO}(\leq, +, \times)$ *with work $\mathcal{O}(\log^d n)$ under changes of size $\mathcal{O}(\log^c n)$.*

We first prove that non-star-free languages cannot be maintained with polylogarithmic work under polylogarithmic changes. The following lemma will be helpful.

▶ **Lemma 7.** *If a language L can be maintained in* $\mathsf{DynFO}(\leq, +, \times)$ *with $f(n)$ work under insertions of size $\log n$ then there is a constant-depth circuit of size $\mathcal{O}(f(2^n))$ that decides L (for inputs of size n).*

Proof sketch. Suppose L can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ with $f(n)$ work under insertions of size $\log n$. From the update formulas for L one can construct a constant-depth circuit family which has $\mathcal{O}(f(n))$ many gates for inputs of length $\log n$, using the standard conversion from first-order formulas to circuits. Such a circuit simulates the update formulas for the insertion of the input into an empty structure. The circuit family thus has $\mathcal{O}(f(2^n))$ many gates for inputs of length n. ◀

It is well-known that for computing the number of 1s modulo a prime p occurring in a bit string of length n, a Boolean circuit of depth ℓ requires $2^{\Omega(n^{1/2\ell})}$ gates (see [24] or, for instance, [13, Theorem 12.27] for a modern exposition).

▶ **Lemma 8.** *If a regular language L is not star-free, then it cannot be maintained in* $\mathsf{DynFO}(\leq, +, \times)$ *with work $\mathcal{O}(\log^d n)$ under changes of size $\mathcal{O}(\log n)$, for any $d \in \mathbb{N}$.*

Proof. Suppose that L is a regular language which is not star-free. Then its syntactic monoid M is not aperiodic. Therefore, there must be an $m \in M$ as well as $k, \hat{p} \in \mathbb{N}$ such that $m^k = m^{k+\hat{p}}$ and $m^{k+i} \neq m^k$ for all $1 \leq i < \hat{p}$, that is, from some point onward, multiplying m with itself becomes periodic with period $\hat{p}$.

Assume, towards a contradiction, that L can be maintained with work $\mathcal{O}(\log^d n)$ under changes of size $\mathcal{O}(\log n)$, for some $d \in \mathbb{N}$. Then, by Lemma 7, there is a constant-depth circuit of size $\mathcal{O}(f(2^n)) = \mathcal{O}(\log^d 2^n) = \mathcal{O}(n^d)$ for L, and therefore also a circuit family $\mathcal{C}$ of this size for the word problem for M.

Let p be the smallest prime factor of $\hat{p}$ and let $t = \frac{\hat{p}}{p}$. From the family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$, we construct a constant-depth circuit family $\mathcal{D} = (D_n)_{n\in\mathbb{N}}$ of size n^d which computes the number of 1's in a 0-1-string modulo p. The idea is to use the period of the element $m \in M$ for simulating "modulo p". For an input of length n, the circuit D_n with input $a_1 \dots a_n$ simulates C_{tn+k} for the input $m_1 \dots m_{tn+k}$ defined via

$$m_i = \begin{cases} m, & \text{for } i \leq k \\ m, & \text{if } i > k \text{ and } a_{(i-k) \bmod n} = 1 \\ 1, & \text{if } i > k \text{ and } a_{(i-k) \bmod n} = 0 \end{cases} .$$

So, from an input with ℓ many 1s we get an input that contains m exactly $t\ell + k$ times. So, $m_1 \cdots m_{tn+k} = m^k$ if and only if $\ell \equiv 0 \pmod{p}$.

The existence of $\mathcal{D}$ contradicts Smolensky's lower bound [24]. ◀

We now show that star-free languages can be maintained with polylogarithmic work under polylogarithmic changes. In [18], this was shown for single tuple changes.

▶ **Lemma 9.** *One can maintain membership in star-free languages in* $\mathsf{DynFO}(\leq, +, \times)$ *with work* $\mathcal{O}(\log^d n)$ *under changes of size* $\log^c n$*, for all* $c \in \mathbb{N}$ *and some* $d \in \mathbb{N}$ *depending on* c*.*

Proof. The statement follows from a detailed analysis of the dynamic programs used in [18, Theorem 5.3] (see also the long version [19, Theorem B.4]), which is based on the approach used in [9].

We sketch the main ideas and necessary adaptations. Instead of maintaining membership of a star-free language L, we aim at maintaining range queries for L's syntactic monoid M. The dynamic problem $\textsc{Range}(M)$ for a monoid M has as input sequences $m_0 \cdots m_{n-1}$ of elements of M, allows for setting positions of this sequence to some element $m \in M \cup \{\epsilon\}$, and support queries of the form $\textsc{Range}(\ell, r)$ which return the product $m_\ell \circ \cdots \circ m_r$.

Syntactic monoids M of star-free languages have one of the following four forms [14]:

(a) $M = \{1\}$
(b) $M = \{1, \sigma, \dots, \sigma^k = \sigma^{k+1}\}$ for some k
(c) $\sigma\sigma' = \sigma$ for all $\sigma, \sigma' \in M - \{1\}$
(d) $M = V \cup T$ for submonoids $V, T \subsetneq M$ such that $\sigma_M \sigma_T \in T - \{1\}$ for all $\sigma_M \in M, \sigma_T \in T - \{1\}$

The idea in [19, Theorem B.4]) and [9] is to inductively construct dynamic programs for $\textsc{Range}(M)$ for each of the Cases (a)–(d). For Case (a) this is trivial. Cases (b) and (c) can be maintained using a single NextInSet instance: For case (b), one needs to find at most k non-neutral symbols. In case (c), the product depends on the first symbol differing from 1. This instance changes only at polylogarithmic many positions and can therefore be updated with polylogarithmic work due to Lemma 1.

Case (d) is more complicated. Suppose m is the input sequence of monoid elements. The dynamic program for $\textsc{Range}(M)$ relies on queries to the following data structures:

- A RANGE(V) instance for the sequence $v = v_0 \cdots v_{n-1}$, where $v_i = m_i$ if $m_i \in V$ and else $v_i = 1$
- A RANGE(T) instance for the sequence $t = t_0 \cdots t_{n-1}$, where $t_i = m_i$ if $m_i \in T - \{1\}$ and else $t_i = 1$.
- A NEXTINSET instance for the set of *switching positions* i where $m_i \in T - \{1\}$, but $m_{i+1} \notin T - \{1\}$.
- A RANGE(T) instance maintaining the sequence $u = u_0 \cdots u_{n-1}$, where u_i is the product $m_{j+1} \dots m_i$ of the elements between the last switching position j before i and i if i is a switching position, and 1 otherwise.

All four data structures can be maintained in polylogarithmic work under changes of polylogarithmic size: NEXTINSET due to Lemma 1 and both RANGE(V) and RANGE(T) by induction.

The dynamic program for RANGE(M) can answer queries by combining the answers of constantly many queries to the above data structures (see [19, Theorem B.4]) and [9]).

Next, we describe the strategy how the dynamic program for RANGE(M) handles updates of polylogarithmic size to the input sequence m. To this end the program:

(1) Updates the sequences v and t.

(2) Computes the new switching positions: Each changed position in m possibly causes a constant number of elements to change their switching status (depending on the update and its position in its old block, see [18]). For each such position i, check whether i is a switching position after the update by checking whether m_i, m_{i+1} are in $T - \{1\}$.

(3) Computes the new block product u_i for all switching positions of affected blocks.

(4) Updates the NEXTINSET instance for the switching positions, the RANGE(T) instances for t and u, and the RANGE(V) instance for v.

Steps (1) and (2) are straightforward and only update polylogarithmically many positions. Step (3) requires only polylogarithmically many work, as only polylogarithmically many positions changed their switching status and thus the blocks can be updated by polylogarithmically many queries to the (old) RANGE(T) instance for u, each requiring polylogarithmic work. Step (4) can be done in polylogarithmic work, since for each of the instances only changes of polylogarithmic size occurred and those can be handled with polylogarithmic work. ◀

This also concludes the proof of Theorem 6.

6 Expressive power of quantification over small sets

In this section, we re-visit the power of first-order logic on small substructures. In the previous sections, via Corollary 3 and its usage, we have seen that for maintaining properties with first-order update formulas under changes of polylogarithmic size, it is essential to know the expressive power of first-order logic on substructures of polylogarithmic size. Corollary 3 relies on the existence of constant-depth circuits of subexponential size for all LOGCFL problems, see Lemma 2.

We now take a different perspective and relate the expressive power of first-order logic on small substructures to the expressive power of second-order logics where second-order quantification is restricted to relations of small size. The results of this section provide an alternative proof of Lemma 2, but also give further insights into second-order logics with size-restricted quantification, which we think is interesting in its own right.

An element of a domain D of size n carries $\log n$ bits of information. Intuitively, assuming the presence of arithmetic relations, this means that in substructures over a subdomain $D' \subseteq D$ of size $\log n$, first-order quantifiers can quantify subsets: if, e.g., an existential quantifier selects a number $k \in \{0, \dots, n-1\}$, then the subset $A \subset D'$ is selected where the i-th element of D' is in A if the i-th bit of k in its binary encoding is 1.

Similarly, first-order quantifiers over D can quantify subsets of size $\sqrt[d]{|D'|}$ over subdomains $D' \subseteq D$ of polylogarithmic size $\log^c n$, for some $d > c$. The binary string of length $\log n$ that is obtained from an existentially quantified number $k \in \{0, \dots, n-1\}$ is split into $\frac{\log n}{c \log\log n}$ segments of length $c \log\log n$ each. The quantified subset $A \subset D'$ contains the i-th element of D' if one segment is the binary encoding of i. As $\frac{\log n}{c \log\log n} \in \Omega(\sqrt[d]{|D'|}) = \Omega(\sqrt[d]{\log^c n})$ for all $d \geq c+1$, the claim follows.

The quantification of relations with arity $k \geq 1$ can be simulated as well over subdomains of size $\log^c n$: a tuple of k elements can be encoded by a bit string of length $ck \log\log n$.

This gives rise to the following variants of second-order logic, where the size of quantified sets is restricted. For $r \in \mathbb{N}$, the subset $r\mathsf{SO}$ of second-order logic allows quantification over relations of arity at most r. For a fixed function $f : \mathbb{N} \to \mathbb{N}$, we define $f(n)$-$r\mathsf{SO}$ to be the logic that has the syntax of $r\mathsf{SO}$ but restricts quantification to relations with at most $f(n)$ many tuples, where n is the size of the underlying domain. For $r = 1$, this logic is called $f(n)$-MSO.

Of course, $f(n)$-$r\mathsf{SO}$ has the same expressive power as $r\mathsf{SO}$ if $f \in \Omega(n^r)$. We will now highlight that it is still very expressive when the size of quantified sets is bounded by functions $f \in o(n)$. Our main interest is in functions $f(n) = \sqrt[c]{n}$, for arbitrary fixed c, due to the above motivation.

In the rest of this section we discuss the power of $f(n)$-$r\mathsf{SO}$ (see Subsection 6.1) and in particular of $f(n)$-MSO (see Subsection 6.2).

6.1 The power of $f(n)$-SO

The restriction of SO to quantification over relations of size $\sqrt[c]{n}$ is very expressive, especially if we allow formulas to use built-in arithmetic relations.

There are complete problems from each level of the polynomial hierarchy contained in $\sqrt[c]{n}$-MSO for each $c \in \mathbb{N}$. For instance, a padded version of 3-colorability, where positive instances with m edges come with m^c isolated nodes (and instances without such isolated nodes are negative), is still NP-complete yet expressible in $\sqrt[c]{n}$-$\exists\mathsf{MSO}$.

While some of its problems are contained, it is unlikely that all of NP is contained in $\sqrt[c]{n}$-$r\mathsf{SO}$ for some $i > 1$ and any $r \in \mathbb{N}$, as this would contradict the exponential time hypothesis: 3-SAT cannot be solved in time $2^{o(n)}$ [12]. This is due to the fact that all $\sqrt[c]{n}$-$r\mathsf{SO}$-definable problems have (bounded-depth) circuits of subexponential size.

▶ **Proposition 10.** *If a problem L can be expressed by a $\sqrt[c]{n}$-$r\mathsf{SO}$ formula, for some $c, r \in \mathbb{N}$, then there is a $d \in \mathbb{N}$ such that L can be decided by a FO-uniform family of depth-d circuits of size $2^{\mathcal{O}(\log n \sqrt[c]{n})}$.*

Proof sketch. We use the naïve translation from formulas to circuits. Suppose there is an $\sqrt[c]{n}$-$r\mathsf{SO}$ formula φ that expresses L, for some natural numbers c and r. Let d be the depth of the syntax tree of φ. Given some n, from the syntax tree of φ we construct in the natural way a depth-d Boolean circuit C_n for input strings of length n as follows. Every existential quantifier is replaced by an "or" gate, with fan-in n for a first-order quantifier and with fan-in upper-bounded by $n^{r\sqrt[c]{n}}$ for an $\sqrt[c]{n}$-$r\mathsf{SO}$ quantifier. Analogously, universal quantifiers

are replaced by "and" gates with fan-in n or $n^{r\sqrt[c]{n}}$, respectively. The obtained circuit is a tree of depth d and of degree at most $n^{r\sqrt[c]{n}}$, so the number of gates is upper-bounded by $(n^{r\sqrt[c]{n}})^{d+1} = 2^{\mathcal{O}(\log n \sqrt[c]{n})}$. ◀

▶ **Corollary 11.** *If* 3-SAT $\in \sqrt[c]{n}$-$r\mathsf{SO}$ *for some* $i, r \in \mathbb{N}$ *with* $i > 1$*, then the exponential time hypothesis fails.*

Thus, NP can likely not be captured by $\sqrt[c]{n}$-rSO. This raises the question, which complexity classes can be captured.

The following theorem shows that $\sqrt[c]{n}$-rSO captures LOGCFL. In conjunction with Proposition 10, it gives an alternative proof for Lemma 2 (which is from [1, Lemma 8.1]).

▶ **Theorem 12.** *Let* Q *be a query that is computable in* LOGCFL*. There is an* $r \in \mathbb{N}$ *such that* Q *is expressible in* $\sqrt[c]{n}$-$r\mathsf{SO}(\leq, +, \times)$*, for every* $c \in \mathbb{N}$*.*

To prove this result, we show that there is an $r \in \mathbb{N}$ such that $\sqrt[c]{n}$-$r\mathsf{SO}$ contains a problem that is LOGCFL-complete under $\mathsf{FO}(\leq, +, \times)$ reductions, for every $c \in \mathbb{N}$. It follows that for every problem $L \in \mathsf{LOGCFL}$ there is a d-dimensional $\mathsf{FO}(\leq, +, \times)$ reduction to the complete problem, for some $d \in \mathbb{N}$, and therefore that L is in $\sqrt[c]{n}$-$(dr)\mathsf{SO}(\leq, +, \times)$ for every $c \in \mathbb{N}$.

The problem we consider is the word problem for groupoids. A (finite) *groupoid* $\mathcal{G} = (G, \circ, 1)$ consists of a finite set G and a binary operation $\circ\colon G \times G \to G$ on this set with identity 1. Note that $\circ$ does not need to be associative. The F *word problem* over a groupoid $\mathcal{G}$ for a fixed set $F \subseteq G$ asks: given a sequence $w_1, \ldots, w_n$ of elements of G, can one introduce parentheses such that $w_1 \circ \cdots \circ w_n$ evaluates to an element from F?

▶ **Lemma 13** ([2, Corollary 3.4])**.** *There is a groupoid* $\mathcal{G} = (G, \circ, 1)$ *and a set* $F \subseteq G$ *such that the* F *word problem over* $\mathcal{G}$ *is* LOGCFL*-complete under* $\mathsf{FO}(\leq, +, \times)$*-reductions.*

▶ **Proposition 14.** *Let* $\mathcal{G} = (G, \circ, 1)$ *be a groupoid and let* $F \subseteq G$ *be a set of groupoid elements. The* F *word problem over* $\mathcal{G}$ *is in* $\sqrt[c]{n}$-2SO*, for every* $c \in \mathbb{N}$*.*

Proof. Let $c \in \mathbb{N}$ be fixed and let $w_1, \ldots, w_n$ be the input sequence. We first explain how the problem can be solved for sub-sequences $w_\ell, \ldots, w_r$ of length $r - \ell + 1 = \sqrt[c]{n}$ of the input. Note that $\sqrt[c]{n}$ parentheses are sufficient to fully determine the evaluation order on this sub-sequence. A $\sqrt[c]{n}$-2SO formula φ_h^1 can existentially quantify $|G|$ binary relations P_g, for all $g \in G$, with the intention that if $(i_1, i_2) \in P_g$ then there is an evaluation order such that $w_{i_1} \circ \cdots \circ w_{i_2}$ evaluates to g, and this evaluation order is encoded by further tuples in the relations $(P_g)_{g \in G}$. A first-order formula can check whether these quantified relations encode a consistent evaluation that results in a fixed element $h \in G$.

For the following formulation of φ_h^1 we assume that the input sequence is encoded by sets R_g, for each $g \in G$. The construct $z + 1$ is an abbreviation for the successor of z which can be easily expressed in FO using the linear order on the positions. For ease of presentation, we assume that parentheses are also introduced for single positions, so the evaluation order for two positions w_1, w_2 is represented as $((w_1) \circ (w_2))$. Technically, then $2\sqrt[c]{n}$ parentheses are necessary which can be encoded by quantifying two relations of size $\sqrt[c]{n}$ for every $g \in G$. We ignore this detail.

$$\varphi_h^1(\ell, r) = \exists P_{g_1} \cdots \exists P_{g_{|G|}} \Big[P_h(\ell, r) \wedge \bigwedge_{g \in G} \forall x \forall y \Big[\big(x \geq \ell \wedge y \leq r \wedge P_g(x, y) \big) \rightarrow \Big[(x = y \wedge R_g(x)) \vee \exists z \bigvee_{g_1 \circ g_2 = g} (P_{g_1}(x, z) \wedge P_{g_2}(z + 1, y)) \Big] \Big] \Big]$$

This formula φ_h^1 for sequences of length $\sqrt[c]{n}$ can be lifted to sequences of length $\sqrt[c]{n^2}$, as the evaluation tree for a sequence of this length can be decomposed into $\mathcal{O}(\sqrt[c]{n})$ subtrees of size at most $\sqrt[c]{n}$: think of deleting $\mathcal{O}(\sqrt[c]{n})$ tree edges such that the tree decomposes into a forest with components of size at most $\sqrt[c]{n}$. For each of these components, the formulas φ_h^1 describe a possible evaluation. On the basis of this evaluation, another $\sqrt[c]{n}$-2SO formula can express the possible evaluation of the whole sequence.

More formally, the formula $\varphi_h^2(\ell, r)$ that expresses whether a subsequence $w_\ell \circ \cdots \circ w_r$ of length $r - \ell + 1 = \sqrt[c]{n^2}$ of the input can be evaluated to $h \in G$ is obtained from $\varphi_h^1(\ell, r)$ by replacing the subformula $(x = y \wedge R_g(x))$ by $\varphi_g^1(x, y)$ and the atoms $P_{g_1}(x, z)$ and $P_{g_2}(z+1, y)$ by $\varphi_{g_1}^1(x, z)$ and $\varphi_{g_2}^1(z+1, y)$, respectively. Repeating this step yields a formula $\varphi_h^c(\ell, r)$ that expresses whether the full sequence $w_1 \circ \cdots \circ w_r$ can be evaluated to h. The final formula expressing the problem only needs to check that $\varphi_f^c(1, n)$ holds for some $f \in F$. ◀

6.2 The power of $f(n)$-MSO

We have seen that $\sqrt[c]{n}$-rSO is quite powerful. It turns out that the same is true already for the case $r = 1$. As an example of the expressive power of $\sqrt[c]{n}$-MSO, we outline that it captures MSO on strings and trees for all $c \in \mathbb{N}$.

We first show that $\sqrt[c]{n}$-MSO can express graph reachability. Recall that reachability is expressible in full MSO by the formula

$$\varphi_{\text{REACH}}(s,t) \stackrel{\text{def}}{=} \forall X \Big(\big(X(s) \wedge \forall x \forall y \big(\big(X(x) \wedge E(x,y) \big) \to X(y) \big) \big) \to X(t) \Big)$$

which uses that a node t can be reached from a node s if and only if all sets of nodes that include s and are closed under outgoing edges also include t.

▶ **Theorem 15.** *Graph reachability can be expressed in $\sqrt[c]{n}$-MSO, for every $c \geq 1$.*

Proof. The idea is simple and has been used similarly in Savitch's Theorem, in the context of small-depth circuits for small distance connectivity (see [3]) and for trading time with alternations (see Nepomnjascii's Theorem and [1]): For determining whether there is a path from s to t of length up to n in $\sqrt[c]{n}$-MSO, for some $c \in \mathbb{N}$, recursively decompose such paths into $\sqrt[c]{n}$ pieces until only paths of length $\sqrt[c]{n}$ remain.

More precisely, we first construct a $\sqrt[c]{n}$-MSO formula that expresses reachability via paths of length at most $\sqrt[c]{n}$ in graphs G of size n. Such a formula $\varphi_{\sqrt[c]{n}-\text{REACH}}(s,t)$ basically states that there is a subset W of nodes (of size at most $\sqrt[c]{n}+1$) such that t is reachable from s in $G[W]$. As we can only quantify over sets of size at most $\sqrt[c]{n}$, the formula asserts that there is an edge from s to a node v such that t is reachable from v via a path of length at most $\sqrt[c]{n} - 1$:

$$\varphi_{\sqrt[c]{n}-\text{REACH}}(s,t) \stackrel{\text{def}}{=} s = t \vee \exists v \, \exists W \, \forall X \Big(E(s,v) \wedge W(v) \\ \wedge \big(X \subseteq W \wedge X(v) \wedge \forall x \forall y \big(\big(X(x) \wedge W(y) \wedge E(x,y) \big) \to X(y) \big) \big) \to X(t) \Big).$$

With a Savitch-like construction, this formula can be lifted to paths of greater length. A path of length $\sqrt[c]{n^2}$ can be decomposed into $\sqrt[c]{n}$ paths of length $\sqrt[c]{n}$. So, the $\sqrt[c]{n}$-MSO formula $\varphi_{\sqrt[c]{n^2}-\text{REACH}}(s,t)$ that we obtain from $\varphi_{\sqrt[c]{n}-\text{REACH}}(s,t)$ by replacing atoms $E(x,y)$ with $\varphi_{\sqrt[c]{n}-\text{REACH}}(x,y)$ expresses reachability along paths of length $\sqrt[c]{n^2}$. Repeating this step c times results in an $\sqrt[c]{n}$-MSO formula that expresses reachability along paths of length $\sqrt[c]{n^c} = n$, that is, a formula that expresses graph reachability. ◀

With similar decompositions as used in the proof of the previous theorem, it is easy to show that $\sqrt[c]{n}$-MSO is as expressive as MSO on strings and trees.

▶ **Theorem 16.** *The following queries can be expressed in $\sqrt[c]{n}$-MSO, for every $c \geq 1$:*

(a) *membership in any regular language,*

(b) *membership in any regular tree language.*

Proof. Fix some $c \geq 1$.

Towards (a), let L be some regular language over an alphabet Σ and let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton (DFA, with state set Q, transition function δ, initial state q_0 and set F of accepting states) that decides L. We outline how an $\sqrt[c]{n}$-MSO formula for L can be constructed.

The usual approach to express membership in L in MSO is to construct a formula $\psi^{p,q}(j,k)$ for any pair p, q of states of $\mathcal{A}$ such that $\psi^{p,q}(j,k)$ is satisfied on a string $w = w_1 \cdots w_n$ if $\mathcal{A}$ goes from state p to state q while reading the word $w_j \cdots w_k$, so, if $\delta^*(p, w_j \cdots w_k) = q$. This formula existentially quantifies for all states $q \in Q$ the set of positions from w such that $\mathcal{A}$ assumes the state q after reading this position. The formula then checks whether for all pairs of neighboring positions these choices are consistent with the transition function δ.

An almost identical $\sqrt[c]{n}$-MSO-formula $\psi^{p,q}_{\sqrt[c]{n}}(j,k)$ can check whether $\delta^*(p, w_{j+1} \cdots w_k) = q$ holds, as long as $w_{j+1} \cdots w_k$ is a substring of length at most $\sqrt[c]{n}$. Similar to the proof of Theorem 15, this formula can be lifted to substrings of larger size.

Towards (b), we follow a similar approach and construct formulas for larger and larger parts of the input tree. We introduce some notation to define subtrees of a tree more easily. Let $T = (V, E, r)$ be a rooted tree, $t \in V$ and $B \subseteq V$. The tree $T(t, B)$ is the subtree of T rooted at t, but all inner nodes of this subtree that are in B become leaves. More formally, let $T(t) = (V', E')$ be the subtree of T rooted at t, and let $B' \stackrel{\text{def}}{=} B \cap V'$. The tree $T(t, B)$ results from $T(t)$ by removing all children of B' and their subtrees. Like in the proof of Theorem 4(b), we assume without loss of generality that T is a binary tree.

Let L be a regular tree language over an alphabet Σ and let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a corresponding tree automaton. Suppose we are given a set B of tree nodes and a partition $B_{q_1}, \ldots, B_{q_\ell}$ of this set that associates a state $q_i \in Q$ to every node from B. Analogously to part (a), there is an $\sqrt[c]{n}$-MSO formula $\psi^q_{\sqrt[c]{n}}(t, B_{q_1}, \ldots, B_{q_\ell})$ that expresses for trees $T(t, B)$ of size $\sqrt[c]{n}$ that $\mathcal{A}$ assigns the state q to t in $T(t, B)$ if the initial states on the leaf nodes from B are as given by the partition. This formula existentially quantifies for each state from Q a set of nodes and verifies that $\mathcal{A}$ actually assigns these states to the corresponding nodes. By using constantly many copies of each quantifier, one can obtain an analogous formula for subtrees of size $\mathcal{O}(\sqrt[c]{n})$.

A formula $\psi^q_{\sqrt[c]{n^2}}(t, B_{q_1}, \ldots, B_{q_\ell})$ for subtrees $T(t, B)$ of size $\sqrt[c]{n^2}$ existentially quantifies a set B' of nodes as well as a partition into ℓ sets $B'_{q_1}, \ldots, B'_{q_\ell}$ with the intention that for each $t' \in B' \cup \{t\}$ the subtrees $T(t', B \cup B')$ have size at most $2\sqrt[c]{n}$. It then checks that for all $t' \in B' \cup \{t\}$ the formula $\psi^q_{\sqrt[c]{n}}(t', B_{q_1} \cup B'_{q_1}, \ldots, B_{q_\ell} \cup B'_{q_\ell})$ is satisfied.

We iterate this step and obtain a formula $\psi^q_n(t)$ that expresses whether $\mathcal{A}$ assigns the state q to t in $T(t)$, and the final formula for the word problem of L only needs to check whether $\psi^f_n(r)$ holds for any accepting state $f \in F$. ◀

As mentioned before, Theorem 16 shows that $\sqrt[c]{n}$-MSO and MSO have the same expressive power on words and trees, respectively, for each $c \in \mathbb{N}$. This is not true any more if the quantifiers are further restricted.

▶ **Proposition 17.** *For $f(n) \in n^{o(1)}$ there is no $f(n)$-*MSO *formula φ that expresses the membership in the language*

$$L_{\mathtt{aa}} = \{w \in \{a,b\}^* \mid w \text{ has an even number of } a\text{'s}\}$$

Proof. Fix $f(n) \in n^{o(1)}$. We assume that there is an $f(n)$-MSO formula φ that expresses $L_{\mathtt{aa}}$ and aim for a contradiction to Håstad's lower bound on the size of constant-depth circuits for Parity [11].

Analogously to the proof of Proposition 10, there is a d such that for any n we can obtain from φ a depth-d circuit C_n for input strings of length n. Gates of this circuit have degree at most $n^{f(n)}$, so the number of gates is upper-bounded by $(n^{f(n)})^{d+1} = 2^{\log n(d+1)f(n)}$ which for $f(n) \in n^{o(1)}$ is not in $2^{\Omega(n^{\frac{1}{d-1}})}$, the lower bound for depth-d circuits that compute a parity function on n input bits [11]. ◀

7 Discussion and future directions

In this paper we have seen that regular languages can be maintained under polylogarithmic changes. We also have seen that results on work-efficiency from [18] can be transferred from single-tuple changes to polylogarithmic changes for regular languages. We further discussed the power of first-order logic on small structures, as this is an essential tool in our proofs.

We highlight three open questions. While we have seen that regular languages cannot be maintained with polylogarithmic work under polylogarithmic changes (see Lemma 8), it remains open whether polylogarithmic work suffices for single tuple changes. Lower bounds in this case seem to require new techniques, as membership in regular languages can be maintained with polylogarithmic work under single tuple changes for change sequence up to polylogarithmic length. Thus, lower bounds have to exploit long change sequences, but unfortunately most known lower bound proof techniques in dynamic complexity theory are expected to be applicable only for constant-length change sequences.

▶ **Open question 1.** Can membership in regular languages be maintained in DynFO with polylogarithmic work under single tuple changes?

Apart from knowing that membership in context-free languages is in DynFO under single tuple changes [10, Theorem 4.1], our knowledge about dynamic membership for context-free languages is very limited.

▶ **Open question 2.** Can membership in all context-free languages be maintained under changes of non-constant size?

The dynamic program used in [10, Theorem 4.1] yields a naïve work bound $\mathcal{O}(n^7)$ for membership in context-free languages as it uses 4-ary auxiliary relations which are updated using three nested existential quantifiers. Using significantly less work than $\mathcal{O}(n^{\omega-1})$, where ω is the matrix multiplication exponent, is likely not possible, since the k-Clique conjecture fails if $\mathcal{O}(n^{\omega-1-\epsilon})$ work suffices for some $\epsilon > 0$ [18, Theorem 6.4].

▶ **Open question 3.** How much work is required for maintaining membership in context-free languages (under single tuple changes)?

References

1 Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. Minimizing disjunctive normal form formulas and AC^0 circuits given a truth table. *SIAM J. Comput.*, 38(1):63–84, 2008. doi:10.1137/060664537.

2 François Bédard, François Lemieux, and Pierre McKenzie. Extensions to Barrington's m-program model. *Theor. Comput. Sci.*, 107(1):31–61, 1993. doi:10.1016/0304-3975(93)90253-P.

3 Xi Chen, Igor Carboni Oliveira, Rocco A. Servedio, and Li-Yang Tan. Near-optimal small-depth lower bounds for small distance connectivity. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 612–625. ACM, 2016. doi:10.1145/2897518.2897534.

4 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. *J. ACM*, 65(5):33:1–33:24, 2018. doi:10.1145/3212685.

5 Samir Datta, Pankaj Kumar, Anish Mukherjee, Anuj Tawari, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of reachability: How many changes can we handle? In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 122:1–122:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.122.

6 Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. A strategy for dynamic programs: Start over and muddle through. *Logical Methods in Computer Science*, 15(2), 2019. doi:10.23638/LMCS-15(2:12)2019.

7 Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. Reachability and distances under multiple changes. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 120:1–120:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.120.

8 Guozhu Dong and Jianwen Su. First-order incremental evaluation of datalog queries. In *Database Programming Languages (DBPL-4), Proceedings of the Fourth International Workshop on Database Programming Languages – Object Models and Languages*, pages 295–308, 1993.

9 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997. doi:10.1145/256303.256309.

10 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19:1–19:36, 2012. doi:10.1145/2287718.2287719.

11 Johan Håstad. Almost optimal lower bounds for small depth circuits. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986. doi:10.1145/12130.12132.

12 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

13 Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012. doi:10.1007/978-3-642-24508-4.

14 Kenneth Krohn and John Rhodes. Algebraic theory of machines. i. prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965. doi:10.1016/S0022-0000(67)80007-2.

15 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004. doi:10.1007/978-3-662-07003-1.

16 Leonid Libkin. Logics for unranked trees: An overview. *Log. Methods Comput. Sci.*, 2(3), 2006. doi:10.2168/LMCS-2(3:2)2006.

17 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997. doi:10.1006/jcss.1997.1520.

18 Jonas Schmidt, Thomas Schwentick, Till Tantau, Nils Vortmeier, and Thomas Zeume. Work-sensitive dynamic complexity of formal languages. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures – 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 490–509. Springer, 2021. doi:10.1007/978-3-030-71995-1_25.

19 Jonas Schmidt, Thomas Schwentick, Till Tantau, Nils Vortmeier, and Thomas Zeume. Work-sensitive dynamic complexity of formal languages. *CoRR*, abs/2101.08735, 2021. arXiv:2101.08735.

20 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control.*, 8(2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.

21 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Dynamic complexity under definable changes. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, volume 68 of *LIPIcs*, pages 19:1–19:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICDT.2017.19.

22 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Dynamic complexity under definable changes. *ACM Trans. Database Syst.*, 43(3):12:1–12:38, 2018. doi:10.1145/3241040.

23 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Sketches of dynamic complexity. *SIGMOD Rec.*, 49(2):18–29, 2020. doi:10.1145/3442322.3442325.

24 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 77–82, 1987. doi:10.1145/28395.28404.

25 Larry Stockmeyer and Uzi Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13(2):409–422, 1984. doi:10.1137/0213027.

26 Nils Vortmeier. *Dynamic expressibility under complex changes*. PhD thesis, TU Dortmund University, Germany, 2019. doi:10.17877/DE290R-20434.

Completeness of Sum-Over-Paths for Toffoli-Hadamard and the Dyadic Fragments of Quantum Computation

Renaud Vilmart
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, LMF, 91190, Gif-sur-Yvette, France

Abstract

The "Sum-Over-Paths" formalism is a way to symbolically manipulate linear maps that describe quantum systems, and is a tool that is used in formal verification of such systems.

We give here a new set of rewrite rules for the formalism, and show that it is complete for "Toffoli-Hadamard", the simplest approximately universal fragment of quantum mechanics. We show that the rewriting is terminating, but not confluent (which is expected from the universality of the fragment). We do so using the connection between Sum-over-Paths and graphical language ZH-Calculus, and also show how the axiomatisation translates into the latter.

Finally, we show how to enrich the rewrite system to reach completeness for the dyadic fragments of quantum computation – obtained by adding phase gates with dyadic multiples of π to the Toffoli-Hadamard gate-set – used in particular in the Quantum Fourier Transform.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Equational logic and rewriting

Keywords and phrases Quantum Computation, Verification, Sum-Over-Paths, Rewrite Strategy, Toffoli-Hadamard, Completeness

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.36

Related Version *Full Version*: `https://arxiv.org/abs/2205.02600`

Funding PEPR integrated project EPiQ ANR-22-PETQ-0007, part of Plan France 2030.

1 Introduction

Sum-Over-Paths (SOP) is a formalism used to represent and manipulate quantum processes in a symbolic way, introduced in 2018 by Amy [3]. Its first important feature is its capacity to translate from most common descriptions of quantum processes in polynomial time and space. The formalism hence provides a intermediary view between usual (matrix) semantics and these usual process descriptions. Its second crucial feature is that it comes equipped with a rewrite system that simplifies the term, without altering its semantics.

Despite its links [17, 18] with graphical languages such as the ZH-Calculus [4] – which will be used in the following –, it provides a different view on the quantum processes, representing them as weighted sums of Dirac kets and bras (a very familiar notation in quantum mechanics).

The formalism has seen several applications, the first of which being verification. Verification is a crucial aspect of computations in the quantum realm, where physical constraints (like no-cloning, or the fundamental probabilistic nature of quantum) make it impossible to do debugging the way we do on classical algorithms. More specifically, the SOP formalism was introduced as a solution to circuit equivalence: To check the equivalence between two circuits $\mathcal{C}_1$ and $\mathcal{C}_2$, the system represents $\mathcal{C}_2^\dagger \circ \mathcal{C}_1$ as an SOP term (where $\mathcal{C}_2^\dagger$ can be seen as the inverse of $\mathcal{C}_2$, easy to describe from it). It then tries to reduce it to the identity. If successful, this shows $\mathcal{C}_1$ and $\mathcal{C}_2$ to represent the same unitary. Otherwise, the system searches for a witness that the term at hand does not represent the identity. As such, the system has

been used in several different projects (e.g. [11, 15]) to check precisely for circuit equivalence. It was later extended to account for families of morphisms and used within environment Qbricks [7, 8] together with automated solvers to verify algorithms and routines such as quantum phase estimation, Grover's search and Shor's algorithm.

Amongst other applications of the Sum-Over-Paths, we may cite noiseless simulation of quantum processes, where the rewrite strategy is used to reduce the number of variables in the term, effectively decreasing the number of summands when expanding the term to actually compute its semantics. It is for instance one of the simulators implemented in the supercomputer Atos QLM [13].

While the initial suggestion for Sum-Over-Paths focussed on the Clifford+T fragment – a universal fragment of quantum computing, i.e. a restriction still capable of approximating with arbitrary precision any quantum process –, it also provided some interesting result for the Clifford fragment. It is known that the latter is not universal [1], and actually efficiently simulable with a classical computer, so it is a good test for the relevance of a formalism to check how it handles them. And indeed, it was shown [3] a "weak" form of confluence of the rewrite system in the Clifford fragment. More precisely, in this fragment, $\mathcal{C}_2^\dagger \circ \mathcal{C}_1$ reduces (in polynomial time) to the identity if and only if $\mathcal{C}_2$ and $\mathcal{C}_1$ represent the same unitary operator.

However, SOP terms may represent more than unitary operator, but actually any linear map. With those, it is still possible to define the above restrictions, and the rewrite system was extended in [25] to get confluence for the – not necessarily unitary – Clifford fragment. When moving to a universal fragment – like Clifford+T – it is expected that we cannot provide a rewrite system with all the good properties of the Clifford case: either reduction is not polynomial, or there is no confluence, or we need an infinite number of rewrites, ... The reason for this is that if we could provide such a system, circuit equivalence would become polynomial, while we know that it is QMA-complete – a quantum variant of NP-complete – [6, 14]. A weaker property than that of confluence we can ask for is completeness: the question here is to decide whether two equivalent terms can be turned into one another, *with the assumption that rewrites can be used in both directions* (in that case, we rather speak of an equational theory, or axiomatisation, than a rewrite system).

Contributions. In this paper, we address the problem of completeness first for arguably the simplest universal fragment of quantum computing, which is *Toffoli-Hadamard*. We provide a fairly simple rewrite system that we show complete for the fragment, and also exhibit two important drawbacks: the non-confluence of the rewrite strategy and the potential explosion of the size of the morphisms during the rewrite. We then show how the rewrite strategy can be tweaked to reach completeness for every dyadic fragment – where we allow phase gates with phase a multiple of $\frac{\pi}{2^k}$ for some k –, a restriction that encompasses Clifford, Clifford+T and Toffoli-Hadamard, and is crucially used in the Quantum Fourier Transform, a central block for algorithms such as Shor's and Quantum Phase Estimation.

Structure of the paper. After reviewing the Sum-Over-Paths formalism in Section 2, the ZH-Calculus in Section 3 and the links between the two in Section 4, we show the completeness result for the Toffoli-Hadamard fragment in Section 5. The extension to the dyadic fragments is then handled in Section 6.
The missing proofs can be found in the full version.

2 Sums-Over-Paths

2.1 The Morphisms

Sums-Over-Paths [3] are a way to symbolically describe linear maps of dimensions powers of 2 over the complex numbers. These linear maps form a †-compact monoidal category [19, 21] denoted **Qubit** where the objects are natural numbers (this makes the category a PROP [16, 26]), where morphisms from n to m are linear maps $\mathbb{C}^{2^n} \to \mathbb{C}^{2^m}$, and where $(\cdot \circ \cdot)$ (resp. $(\cdot \otimes \cdot)$) is the usual composition (resp. tensor product) of linear maps. The category is endowed with a *symmetric braiding* $\sigma_{n,m} : n+m \to m+n$, as well as a *compact structure* $(\eta_n : 0 \to 2n, \epsilon_n : 2n \to 0)$. Furthermore, there exists an inductive contravariant endofunctor $(\cdot)^\dagger$, that behaves properly with the symmetric braiding and the compact structure. For more information on these structures, see [21].

The formalism of SOP relies heavily on the Dirac notation for quantum states and operators of **Qubit**. The two canonical states of a single qubit are denoted $|0\rangle$ and $|1\rangle$. They form a basis of $\mathbb{C}^2$, and can be viewed as vectors $|0\rangle = \begin{pmatrix}1 & 0\end{pmatrix}^\intercal$ and $|1\rangle = \begin{pmatrix}0 & 1\end{pmatrix}^\intercal$. A 1-qubit state is then merely a normalised linear combination of these two elements. Using $(\cdot \otimes \cdot)$, they can be used to build the basis states of larger systems, e.g. $|010\rangle := |0\rangle \otimes |1\rangle \otimes |0\rangle$ is a basis state of a 3-qubit system. Again, the state of an arbitrary n-qubit system is a normalised linear combination of the 2^n basis states. We will use extensively the following notation $\langle x|$ to represent the dagger (transpose conjugate) of $|x\rangle$. The identity on a qubit can then be expressed in Dirac notation as $\mathbb{I} := |0\rangle\langle 0| + |1\rangle\langle 1|$, where $|x\rangle\langle y| := |x\rangle \circ \langle y|$.

We give in the following the definition of Sum-Over-Paths of [25], which differs from [3] in the way the input qubits are treated, by making them more symmetric with the outputs. This makes some concepts, like the † or the compact structure, more natural.

▶ **Definition 1 (SOP).** *We define* **SOP** *as the collection of objects* $\mathbb{N}$ *and morphisms between them that are tuples* $f : n \to m := (s, \vec{y}, P, \vec{O}, \vec{I})$*, which we write:* $s \sum_{\vec{y} \in V^k} e^{2i\pi P(\vec{y})} \left|\vec{O}(\vec{y})\right\rangle\!\left\langle\vec{I}(\vec{y})\right|$

where $s \in \mathbb{R}$*,* $\vec{y} \in V^k$ *with* V *a set of variables,* $P \in \mathbb{R}[X_1, \ldots, X_k]/(X_i^2 - X_i)$ *is called the* phase polynomial *of* f[1]*,* $\vec{O} \in (\mathbb{F}_2[X_1, \ldots, X_k])^m$ *and* $\vec{I} \in (\mathbb{F}_2[X_1, \ldots, X_k])^n$ *–* $\mathbb{F}_2$ *being the binary field, whose only two elements are its additive and multiplicative identities, denoted* 0 *and* 1*.*

Compositions are obtained as[2]*:*

- $f \circ g := \frac{s_f s_g}{2^m} \sum_{\substack{\vec{y}_f, \vec{y}_g \\ \vec{y} \in V^m}} e^{2i\pi\left(P_g + P_f + \frac{\vec{O}_g \cdot \vec{y} + \vec{I}_f \cdot \vec{y}}{2}\right)} \left|\vec{O}_f\right\rangle\!\left\langle\vec{I}_g\right|$ *where* $m = \left|\vec{I}_f\right| = \left|\vec{O}_g\right|$
- $f \otimes g := s_f s_g \sum_{\vec{y}_f, \vec{y}_g} e^{2i\pi(P_g + P_f)} \left|\vec{O}_f \vec{O}_g\right\rangle\!\left\langle\vec{I}_f \vec{I}_g\right|$

We distinguish particular morphisms:

- *Identity morphisms* $id_n : \sum_{\vec{y} \in V^n} |\vec{y}\rangle\langle\vec{y}|$

[1] The quotient in the phase polynomial means that we consider each occurrence of the square of a variable to be equal to the variable itself $X_i^2 - X_i = 0$, since they will be evaluated over $\{0, 1\}$. We can further constrain the polynomial by taking it modulo 1, but only when considered as an element of a group, once all the products have been evaluated, as otherwise all phase polynomials would be evaluated to 0 as $P = P \times 1 = P \times 0 = 0$.

[2] To avoid further clutter, we may not specify the variables of polynomials, e.g. P_g actually stands for $P_g(\vec{y}_g)$, $\vec{O}_g$ for $\vec{O}_g(\vec{y}_g)$ etc...

- *Symmetric braidings* $\sigma_{n,m} = \sum_{\vec{y}_1,\vec{y}_2} |\vec{y}_2,\vec{y}_1\rangle\langle\vec{y}_1,\vec{y}_2|$
- *Morphisms for compact structure* $\eta_n = \sum_{\vec{y}} |\vec{y},\vec{y}\rangle\langle|$ *and* $\epsilon_n = \sum_{\vec{y}} |\rangle\langle\vec{y},\vec{y}|$

We also distinguish two functors that have **SOP** *as a domain:*

- *The* †*-functor is given by:* $f^\dagger := s\sum_{\vec{y}} e^{-2i\pi P} \left|\vec{I}\right\rangle\left\langle\vec{O}\right|$
- *The functor* $[\![\cdot]\!] : \mathbf{SOP} \to \mathbf{Qubit}$ *is defined as:* $[\![f]\!] := s \sum_{\vec{y}\in\{0,1\}^k} e^{2i\pi P(\vec{y})} \left|\vec{O}(\vec{y})\right\rangle\left\langle\vec{I}(\vec{y})\right|$

The †-functor is particularly important to characterise maps that are unitary – the pure transformations that are allowed by quantum mechanics: f is called unitary if $[\![f^\dagger \circ f]\!] = id$.

▶ **Example 2.** The Hadamard and Toffoli gates (which justify the name of the first fragment we will consider in the following), can be represented in this formalism as:

$$H := \frac{1}{\sqrt{2}} \sum_{y_0,y_1} e^{2i\pi\frac{y_0y_1}{2}} |y_1\rangle\langle y_0| \qquad \text{Tof} := \sum_{y_0,y_1,y_2} |y_0,y_1,y_2 \oplus y_0y_1\rangle\langle y_0,y_1,y_2|$$

It can be checked that both operators are unitary.

As is customary, we consider equality of the SOP morphisms up to α-conversion, i.e. renaming of the variables. Notice that the definition of the composition $(\cdot \circ \cdot)$ gets somewhat involved. This is to cope with the way we deal with the inputs, which can be any boolean polynomial. The additional terms $\frac{\vec{O}_g\cdot\vec{y}+\vec{I}_f\cdot\vec{y}}{2}$ enforce that $O_{gi} = I_{fi}$ for all $0 \le i < m$. Indeed, when summing over the variable y_i, we get $(1 + e^{i\pi(O_{gi}+I_{fi})})$ – which is non-null only when $O_{gi} = I_{fi}$ – as a factor of the whole morphism. This presentation has the advantage of keeping the size of the morphism polynomial with the size of the quantum circuit – or ZH-diagram, see below – it can be built from, no matter what gate set is used. A downside, however, is that the above does not directly constitute a category, as for instance $id \circ id \neq id$. However, it suffices to quotient the formalism with rewrite rules to turn it into a proper category [25], hence justifying the use of the term "functor" for the last two maps.

2.2 A Rewrite System

We hence give in Figure 1 a set of rewrite rules denoted $\xrightarrow[\text{TH}]{}$ that induces an equational theory $\underset{\text{TH}}{\sim}$ (the symmetric and transitive closure of $\xrightarrow[\text{TH}]{}$).

We need in the conditions of all the rules the function Var, that, given a set or list of polynomials, gives the set of all variables used in them. We call *internal variable* a variable that is present in the morphism t but not in its inputs/outputs, i.e. a variable y_0 such that $y_0 \in \text{Var}(t) \setminus \text{Var}(\vec{O},\vec{I})$. It is worth noting that searching for an occurrence of, and applying any of these rules *once* can be done in polynomial time.

The rules (ket) and (bra) correspond to changes of variables that are necessary to get a unique normal form in the Clifford case [25], and the rule (Elim) simply gets rid of a variable that is used nowhere in the term and simply contributes to a global phase (since that variable is supposed to range over two values, it contributes to a multiplicative scalar 2).

The rules (HHgen), (HHnl) and (Z) all stem from a particular observation: In the morphism $t = \sum e^{2i\pi\left(\frac{y_0}{2}\widehat{Q}+R\right)} \left|\vec{O}\right\rangle\left\langle\vec{I}\right|$ where y_0 is internal and not in R, if Q is evaluated to 1, then the whole morphism is interpreted as null. This is exactly what (Z) captures – and the conditions on R, $\vec{O}$ and I are simply here to avoid applying the rule indefinitely.

The rule (HHgen) deals with a case where the polynomial Q can be forced to 0, whilst the rule (HHnl) factorises different such polynomials Q into one.

$$\sum_{\vec{y}} e^{2i\pi P} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right| \xrightarrow[y_0 \notin \mathrm{Var}(P,\vec{O},\vec{I})]{} 2 \sum_{\vec{y}\setminus\{y_0\}} e^{2i\pi P} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right| \tag{Elim}$$

$$t = \sum e^{2i\pi\left(\frac{y_0}{2}(y_i\widehat{Q}+\widehat{Q'}+1)+R\right)} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right| \xrightarrow[\substack{y_0 \notin \mathrm{Var}(Q,Q',R,\vec{O},\vec{I}) \\ y_i \notin \mathrm{Var}(Q,Q') \\ QQ'=Q'}]{} t[y_i \leftarrow 1 \oplus Q'] \tag{HHgen}$$

$$t = \sum e^{2i\pi\left(\frac{y_0}{2}\widehat{Q}+\frac{y'_0}{2}\widehat{Q'}+R\right)} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right| \xrightarrow[y_0,y'_0 \notin \mathrm{Var}(Q,Q',R,\vec{O},\vec{I})]{} 2t[y'_0 \leftarrow y_0 \oplus y_0 Q] \tag{HHnl}$$

$$t = \sum_{\vec{y}} e^{2i\pi(P)} |\cdots, \overbrace{y_0 \oplus O'_i}^{O_i}, \cdots\rangle\!\left\langle\vec{I}\right| \xrightarrow[\substack{O'_i \neq 0 \\ y_0 \notin \mathrm{Var}(O_1,\ldots,O_{i-1},O'_i)}]{} t[y_0 \leftarrow O_i] \tag{ket}$$

$$t = \sum_{\vec{y}} e^{2i\pi(P)} \left|\vec{O}\right\rangle\!\langle\cdots, \overbrace{y_0 \oplus I'_i}^{I_i}, \cdots| \xrightarrow[\substack{I'_i \neq 0 \\ y_0 \notin \mathrm{Var}(\vec{O},I_1,\ldots,I_{i-1},I'_i)}]{} t[y_0 \leftarrow I_i] \tag{bra}$$

$$s \sum_{\vec{y}} e^{2i\pi\left(\frac{y_0}{2}+R\right)} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right| \xrightarrow[\substack{R\neq 0 \text{ or } \vec{O},\vec{I}\neq\vec{0} \\ y_0 \notin \mathrm{Var}(R,\vec{O},\vec{I})}]{} \sum_{y_0} e^{2i\pi\left(\frac{y_0}{2}\right)} |0,\cdots,0\rangle\langle 0,\cdots,0| \tag{Z}$$

Figure 1 Rewrite system $\xrightarrow[\mathrm{TH}]{}$.

▶ Remark 3. When performing certain rules, we have to substitute a variable by a boolean polynomial Q. We need to be able to understand Q as a phase polynomial, as the variable can occur in P. The map $\widehat{(\cdot)} : \mathbb{F}_2[X_1,\ldots,X_k] \to \mathbb{R}[X_1,\ldots,X_k]/(X_i^2 - X_i)$, serves this purpose. It is inductively defined as:

$$\widehat{Q_1 Q_2} = \widehat{Q_1}\widehat{Q_2} \qquad \widehat{Q_1 \oplus Q_2} = \widehat{Q_1} + \widehat{Q_2} - 2\widehat{Q_1}\widehat{Q_2} \qquad \widehat{y_i} = y_i \qquad \widehat{\alpha} = \alpha$$

▶ Remark 4. When rewriting SOP-morphisms for simplification or verification, it can be beneficial to not only reduce the number of variables – which is what all rules but (ket/bra) do –, but also to keep the size of the phase polynomial as short as possible. In that respect, the rule (HHgen) can be generalised to:

$$t = \sum e^{2i\pi\left(\frac{y_0}{2}(y_i\widehat{Q}+\widehat{QQ'}+1)+R\right)} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right| \xrightarrow[\substack{y_0 \notin \mathrm{Var}(Q,Q',R,\vec{O},\vec{I}) \\ y_i \notin \mathrm{Var}(Q,Q')}]{} t[y_i \leftarrow 1 \oplus Q'] \tag{HHgen'}$$

where the polynomial Q' can here be smaller (in the number of terms) than the one in (HHgen). However, finding a "minimal" Q' for this rule is a hard problem, as it requires the use of boolean Groebner bases [20]. (HHgen) can be seen as a particular case of (HHgen'), where $Q' \leftarrow QQ'$, as $Q \times QQ' = QQ'$. The rule (HHgen) is sufficient for the scope of this paper.

In [3] was introduced a particular and important rule:

$$t = \sum e^{2i\pi\left(\frac{y_0}{2}(y_i+\widehat{Q})+R\right)} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right| \xrightarrow[\substack{y_0 \notin \mathrm{Var}(Q,R,\vec{O},\vec{I}) \\ y_i \notin \mathrm{Var}(Q)}]{} 2 \sum e^{2i\pi R[y_i \leftarrow \widehat{Q}]} \left(\left|\vec{O}\right\rangle\!\left\langle\vec{I}\right|\right)[y_i \leftarrow Q] \tag{HH}$$

This one is a particular case of the rule (HHgen) (with additional use of the rule (Elim)), where $Q \leftarrow 1$, $Q' \leftarrow Q \oplus 1$. Moreover, the rule gave enough power to the formalism to become a †-compact PROP [25]. We can extend this result here thanks to:

▶ **Proposition 5.**

$$\forall t_1, t_2 \in \mathbf{SOP},\quad t_1 \underset{\text{TH}}{\sim} t_2 \implies \begin{cases} A \circ t_1 \circ B \underset{\text{TH}}{\sim} A \circ t_2 \circ B & \text{for all } A,\ B \text{ composable} \\ A \otimes t_1 \otimes B \underset{\text{TH}}{\sim} A \otimes t_2 \otimes B & \text{for all } A,\ B \end{cases}$$

Thanks to this Proposition, and since $\mathbf{SOP}/\underset{\text{HH}}{\sim}$ is a †-compact PROP by [25], we get:

▶ **Corollary 6.** $\mathbf{SOP}/\underset{\text{TH}}{\sim}$ *is a †-compact PROP.*

The set of rules was obviously chosen so as to preserve the semantics:

▶ **Proposition 7** (Soundness). *For any two* **SOP** *morphisms* t_1 *and* t_2*, if* $t_1 \xrightarrow[\text{TH}]{*} t_2$*, then* $[\![t_1]\!] = [\![t_2]\!]$.

▶ **Example 8.** The following morphism:

$$\sum_{\vec{y}} e^{2i\pi(\frac{y_0 y_1 y_2}{2} + \frac{y_2}{2} + \frac{y_2 y_3 y_4}{2})} |y_4\rangle\langle y_0|$$

is irreducible using the rules of [3] and [25]. However, here it can be reduced to:

$$\sum_{y_0,y_1,y_2,y_3,y_4} e^{2i\pi(\frac{y_0 y_1 y_2}{2} + \frac{y_2}{2} + \frac{y_2 y_3 y_4}{2})} |y_4\rangle\langle y_0|$$

$$\xrightarrow[y_3 \leftarrow y_1 \oplus y_0 y_1 y_2]{\text{(HHnl)}} 2 \sum_{y_0,y_1,y_2,y_4} e^{2i\pi(\frac{y_0 y_1 y_2}{2} + \frac{y_2}{2} + \frac{y_1 y_2 y_4}{2} + \frac{y_0 y_1 y_2 y_4}{2})} |y_4\rangle\langle y_0|$$

$$\xrightarrow[y_1 \leftarrow 1]{\text{(HHgen)}} 2 \sum_{y_0,y_2,y_4} e^{2i\pi(\frac{y_0 y_2}{2} + \frac{y_2}{2} + \frac{y_2 y_4}{2} + \frac{y_0 y_2 y_4}{2})} |y_4\rangle\langle y_0|$$

The first rewrite can be made clearer by writing the phase polynomial as $\frac{y_1}{2}(y_0 y_2) + \frac{y_3}{2}(y_2 y_4) + \frac{y_2}{2}$, and the second one by writing it as $\frac{y_2}{2}\,(y_1(y_0 + y_4 + y_0 y_4) + 0 + 1)$. Recall that variables are binary variables, so $y_i^2 = y_i$.

3 The ZH-Calculus

The graphical calculi ZX, ZW and ZH [4, 9, 10] are calculi for quantum computing, with a tight link with the Sum-Over-Paths formalism [17, 18, 25], and whose completeness was proven in particular for the Toffoli-Hadamard fragment [5, 12, 23, 24].

This fragment of quantum mechanics is approximately universal [2, 22], and it is arguably the simplest one with this property. This is the fragment we will be interested in, in most of the following of the paper; and the associated completeness result will be paramount in the development of the following.

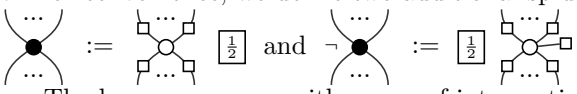
We choose to present here the ZH-Calculus, because of its proximity with both **SOP** and the Toffoli-Hadamard fragment. Notice however that there exist translations between all the aforementioned graphical calculi, so by composition, we can connect **SOP** to all of them.

ZH is a PROP whose morphisms – read here from top to bottom – are composed (sequentially $(\cdot \circ \cdot)$ or in parallel $(\cdot \otimes \cdot)$) from Z-spiders and H-spiders:

- $Z^n_m : n \to m ::$  , called Z-spider
- $H^n_m(r) : n \to m ::$  , called H-spider, with a parameter $r \in \mathbb{C}$

When r is not specified, the parameter in the H-spider is taken to be -1.

ZH is made a †-compact PROP, which means it also has a symmetric structure $\sigma_{n,m} ::$  , a compact structure $\left(\eta_n :: \right.$  , $\left.\epsilon_n :: \right.$ $\left.\right)$, and a †-functor $(\cdot)^\dagger : \mathbf{ZH}^{\mathrm{op}} \to \mathbf{ZH}$. It is defined by: $(Z^n_m)^\dagger := Z^m_n$ and $(H^n_m(r))^\dagger := H^m_n(\overline{r})$ where $\overline{r}$ is the complex conjugate of r. For convenience, we define two additional spiders:

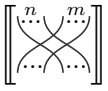


The language comes with a way of interpreting the morphisms as morphisms of **Qubit**. The standard interpretation $[\![\cdot]\!] : \mathbf{ZH} \to \mathbf{Qubit}$ is a †-compact-PROP-functor, defined as:

$$[\![\text{  }]\!] = |0^m\rangle\langle 0^n| + |1^m\rangle\langle 1^n| \qquad [\![\text{}]\!] = |0\rangle\langle 0| + |1\rangle\langle 1|$$

$$[\![\text{  }]\!] = \sum_{j_k, i_k \in \{0,1\}} r^{j_1 \dots j_m i_1 \dots i_n} |j_1, \dots, j_m\rangle\langle i_1, \dots, i_n|$$

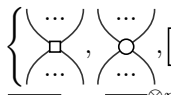
$$[\![\text{  }]\!] = \sum_{i_k \in \{0,1\}} |i_1, \dots, i_n, i_1, \dots, i_n\rangle = [\![\text{  }]\!]^\dagger$$

$$[\![\text{  }]\!] = \sum_{i_k, j_k \in \{0,1\}} |j_1, \dots, j_m, i_1, \dots, i_n\rangle\langle i_1, \dots, i_n, j_1, \dots, j_m|$$

Notice that we used the same symbol for two different functors: the two interpretations $[\![\cdot]\!] : \mathbf{SOP} \to \mathbf{Qubit}$ and $[\![\cdot]\!] : \mathbf{ZH} \to \mathbf{Qubit}$. It should be clear from the context which one is to be used.

The language is universal: $\forall f \in \mathbf{Qubit}$, $\exists D_f \in \mathbf{ZH}$, $[\![D_f]\!] = f$. In other words, the interpretation $[\![\cdot]\!]$ is surjective.

The language comes with an equational theory, which in particular gives the axioms for a †-compact PROP. We will not present it here.

We can easily define a restriction of **ZH** that exactly captures the Toffoli-Hadamard fragment of quantum mechanics [5, 23], as the language generated by:  .

Notice that the two black spiders can still be defined if we also define  :=  $^{\otimes p}$. We denote this restriction by $\mathbf{ZH}_{\mathrm{TH}}$.

This restriction is provided with an equational theory, given in Figure 2[3], that makes it complete.

▶ **Theorem 9** ([23] Completeness of $\mathbf{ZH}_{\mathrm{TH}}$/ $\mathrm{ZH}_{\mathrm{TH}}$).

$$\forall D_1, D_2 \in \mathbf{ZH}_{\mathrm{TH}}, \quad [\![D_1]\!] = [\![D_2]\!] \iff \mathrm{ZH}_{\mathrm{TH}} \vdash D_1 = D_2$$

[3] The axiomatisation provided here is that of [23]. It was later simplified in [5] in a fragment that is very close to the one we consider, but does *not* contain the scalar $\frac{1}{\sqrt{2}}$. As we would rather have this scalar in the language (to properly represent the Hadamard gate), instead of giving a mix of the two axiomatisation, we decided to stick to the first one.

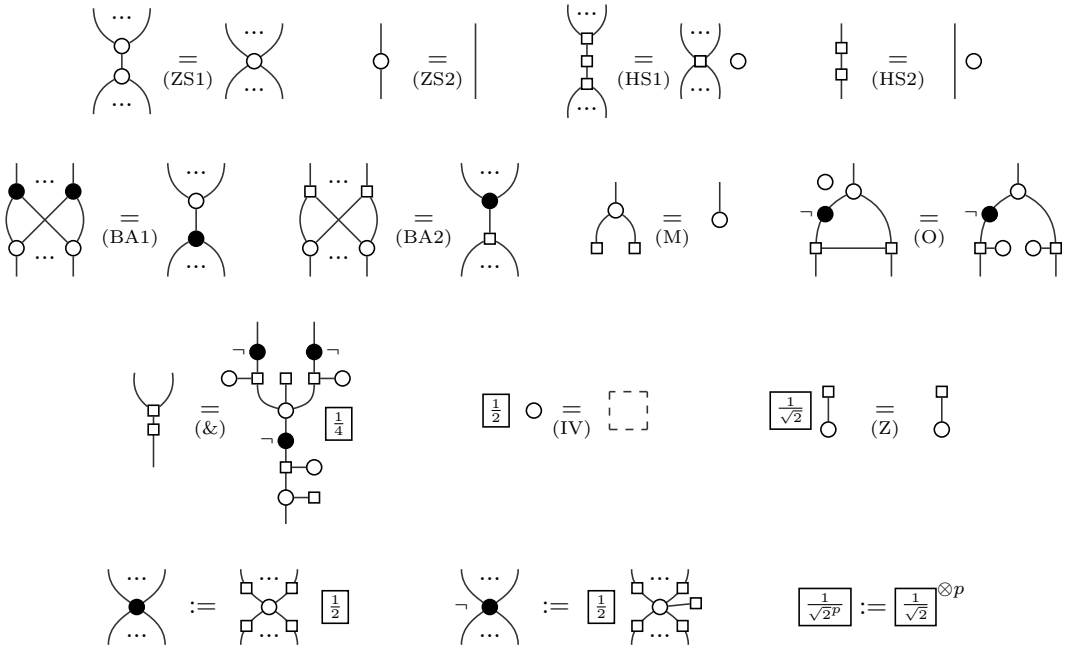


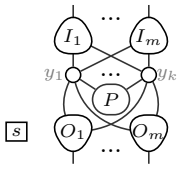
Figure 2 Set of rules $\mathrm{ZH}_{\mathrm{TH}}$ [23].

4 Translations between SOP and ZH

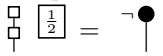
4.1 From SOP to ZH

It is possible to translate **SOP** morphisms to ZH-diagrams using interpretation $[\cdot]^{\mathrm{ZH}} : \mathbf{SOP} \to \mathbf{ZH}$. A description of $[\cdot]^{\mathrm{ZH}} : \mathbf{SOP} \to \mathbf{ZH}$ was defined in [17, 18] and in [25]. We choose the latter definition as it fits our definition of **SOP**.

$$\left[s \sum_{\vec{y}} e^{2i\pi P} \, |O_1, \ldots, O_m\rangle\langle I_1, \ldots, I_n| \right]^{\mathrm{ZH}} :=$$



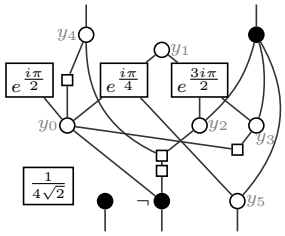
where the row of Z-spiders represents the variables $y_1, \ldots, y_k$. Informally:

- each monomial $\alpha y_{i_1} \ldots y_{i_s}$ in P gives a single H-spider with parameter $e^{i\frac{\alpha}{2\pi}}$ and connected to the Z-spiders that represent y_{i_1},...,y_{i_s}
- each monomial $y_{i_1} \ldots y_{i_s}$ in O_i is represented by  where the inputs are connected to the Z-spiders that represent y_{i_1},...,y_{i_s}. Notice that the only (non-zero) constant monomial is 
- these monomials are then added to form O_i thanks to 
- the nodes I_i are defined similarly, but upside-down

For more details, see [25].

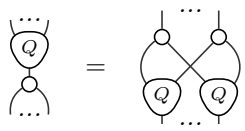
▶ **Example 10.** The **SOP** morphism:

$$\frac{1}{2\sqrt{2}} \sum_{\vec{y}} e^{2i\pi\left(\frac{1}{4}y_0 + \frac{1}{2}y_4y_0 + \frac{1}{8}y_5y_0y_1 + \frac{3}{4}y_1y_2y_3 + \frac{1}{2}y_0y_3\right)} \, |0, 1 \oplus y_0 \oplus y_4y_2, y_5\rangle\langle y_4, y_5 \oplus y_2 \oplus y_3|$$

is mapped to 

The boolean polynomials as defined above are given in their (unique) expanded form. These can easily be shown to be copied through the white node:

▶ **Lemma 11.**

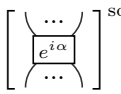


This translation preserves the semantics:

▶ **Proposition 12** ([25]). $[\![[\cdot]^{\mathrm{ZH}}]\!] = [\![\cdot]\!]$.

4.2 From ZH to SOP

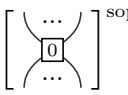
Any **ZH**-diagram can be understood as a **SOP**-morphism. To do so, we use the PROP-functor $[\cdot]^{\mathrm{sop}} : \mathbf{ZH} \to \mathbf{SOP}$ defined as:

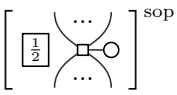


$$:= \sum e^{2i\pi \frac{\alpha}{2\pi} x_1 \dots x_n y_1 \dots y_m} |y_1, \dots, y_m\rangle\langle x_1, \dots, x_n|$$

$:= s\,|\rangle\langle|$ for $s \in \mathbb{R}$

$:= \sum_y |y, \dots, y\rangle\langle y, \dots, y|$

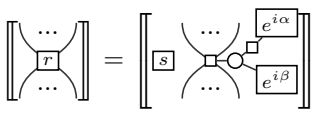


$:=$


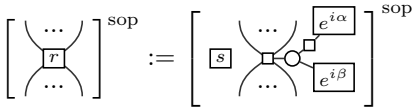
The functor furthermore maps the symmetric braiding (resp. the compact structure) of **ZH** to the symmetric braiding (resp. the compact structure) of **SOP**.

This does not give a full description of $[\cdot]^{\mathrm{sop}}$, as we did not describe the interpretation of the H-spider for all parameters, but only for phases and 0. However, any H-spider can be decomposed using the previous ones:

▶ **Lemma 13.** *For any $r \in \mathbb{C}$ such that $|r| \notin \{0, 1\}$, there exist $s \in \mathbb{C}$, $\alpha, \beta \in \mathbb{R}$ such that:*

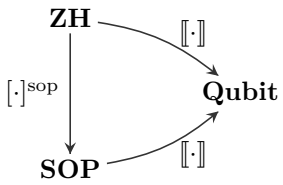


As a consequence, we extend the definition of $[\cdot]^{\mathrm{sop}}$ by:



This interpretation of ZH-diagrams as **SOP**-morphisms preserves the semantics:

▶ **Proposition 14** ([25]). $[\![[\cdot]^{\mathrm{sop}}]\!] = [\![\cdot]\!]$. *In other words, the following diagram commutes:*



The composition of the two interpretations is the identity up to small rewrites:

▶ **Proposition 15** ([25]). $\left[[\cdot]^{\mathrm{ZH}}\right]^{\mathrm{sop}} \underset{\mathrm{TH}}{\sim} (\cdot)$

4.3 Restrictions of SOP

Recall that $\mathbf{ZH}_{\mathrm{TH}}$ exactly captures the Toffoli-Hadamard fragment of quantum mechanics. We can then use the two interpretations to define the Toffoli-Hadamard fragment of **SOP**. We actually go a step beyond and define a family of fragments indexed by n:

▶ **Definition 16** ($\mathbf{SOP}[\frac{1}{2^n}]$). *We define* $\mathbf{SOP}[\frac{1}{2^n}]$ *as the restriction of* **SOP** *to morphisms of the form:* $t = \frac{1}{\sqrt{2}^p} \sum e^{2i\pi \frac{P}{2^n}} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right|$ *where* $p \in \mathbb{Z}$ *and* P *has integer coefficients.*

The Toffoli-Hadamard fragment is then the first such restriction ($n = 1$):

▶ **Proposition 17.** $\mathbf{SOP}[\frac{1}{2}]$ *captures exactly the Toffoli-Hadamard fragment of quantum mechanics.*

Proof. We can prove this by showing that $[\mathbf{ZH}_{\mathrm{TH}}]^{\mathrm{sop}} \subseteq \mathbf{SOP}[\frac{1}{2}]$ and that $\left[\mathbf{SOP}[\frac{1}{2}]\right]^{\mathrm{ZH}} \subseteq \mathbf{ZH}_{\mathrm{TH}}$. The two claims are straightforward verifications, and use the fact that compositions of $\mathbf{SOP}[\frac{1}{2}]$-morphisms give $\mathbf{SOP}[\frac{1}{2}]$-morphisms.
Then, $[\![\mathbf{ZH}_{\mathrm{TH}}]\!] = [\![[\mathbf{ZH}_{\mathrm{TH}}]^{\mathrm{sop}}]\!] \subseteq \left[\!\left[\mathbf{SOP}[\frac{1}{2}]\right]\!\right] = \left[\!\left[\left[\mathbf{SOP}[\frac{1}{2}]\right]^{\mathrm{ZH}}\right]\!\right] \subseteq [\![\mathbf{ZH}_{\mathrm{TH}}]\!]$, so:

$$\left[\!\left[\mathbf{SOP}[\tfrac{1}{2}]\right]\!\right] = [\![\mathbf{ZH}_{\mathrm{TH}}]\!]$$ ◀

Notice in particular that the Hadamard and Toffoli gates given in Example 2 lie in this fragment. Not all of $\mathbf{SOP}[\frac{1}{2}]$ can be generated by these two gates however, as $\mathbf{SOP}[\frac{1}{2}]$ comprises linear maps that are not unitary, i.e. such that $[\![t^\dagger \circ t]\!] \neq id$.

5 Completeness for Toffoli-Hadamard

In this section, we aim to show that the set of rules $\underset{\mathrm{TH}}{\longrightarrow}$ captures the whole Toffoli-Hadamard fragment of quantum mechanics. We do so by transporting the similar result from $\mathbf{ZH}_{\mathrm{TH}}$ to $\mathbf{SOP}[\frac{1}{2}]$. First, we show:

▶ **Proposition 18.** $\forall D_1, D_2 \in \mathbf{ZH}_{\mathrm{TH}}, \quad \mathrm{ZH}_{\mathrm{TH}} \vdash D_1 = D_2 \implies [D_1]^{\mathrm{sop}} \underset{\mathrm{TH}}{\sim} [D_2]^{\mathrm{sop}}$

We can then use the previous proposition to show the main result of this paper:

▶ **Theorem 19.** $\mathbf{SOP}[\frac{1}{2}]/\underset{\mathrm{TH}}{\sim}$ *is complete, i.e.:* $\forall t_1, t_2 \in \mathbf{SOP}[\frac{1}{2}], \quad [\![t_1]\!] = [\![t_2]\!] \iff t_1 \underset{\mathrm{TH}}{\sim} t_2$

Proof. Let t_1 and t_2 be two $\mathbf{SOP}[\frac{1}{2}]$-morphisms such that $[\![t_1]\!] = [\![t_2]\!]$. By Proposition 12: $[\![[t_1]^{\mathrm{ZH}}]\!] = [\![[t_2]^{\mathrm{ZH}}]\!]$.
By completeness of $\mathbf{ZH}_{\mathrm{TH}}/\,\mathrm{ZH}_{\mathrm{TH}}$ (Theorem 9): $\mathrm{ZH}_{\mathrm{TH}} \vdash [t_1]^{\mathrm{ZH}} = [t_2]^{\mathrm{ZH}}$
Thanks to Proposition 18: $\left[[t_1]^{\mathrm{ZH}}\right]^{\mathrm{sop}} \underset{\mathrm{TH}}{\sim} \left[[t_2]^{\mathrm{ZH}}\right]^{\mathrm{sop}}$. Finally, by Proposition 15:

$$t_1 \underset{\mathrm{TH}}{\sim} \left[[t_1]^{\mathrm{ZH}}\right]^{\mathrm{sop}} \underset{\mathrm{TH}}{\sim} \left[[t_2]^{\mathrm{ZH}}\right]^{\mathrm{sop}} \underset{\mathrm{TH}}{\sim} t_2 \qquad \blacktriangleleft$$

The rewrite system is however not sufficient to get to a unique normal form, as:

▶ **Lemma 20** (Non-Confluence). *The rewrite system* $\underset{\mathrm{TH}}{\longrightarrow}$ *is not confluent.*

Proof. The $\mathbf{SOP}[\frac{1}{2}]$-morphism: $t = \sum e^{2i\pi\left(\frac{1}{2}y_0y_6+\frac{1}{2}y_8y_9y_6+\frac{1}{2}y_4y_5y_6+\frac{1}{2}y_8y_9y_{12}\right)}\,|y_0\rangle$
can be reduced to (at least) three different non-reducible morphisms:

- $t \xrightarrow[\mathrm{HH}(y_6,[y_0\leftarrow y_4y_5\oplus y_8y_9])]{} 2\sum e^{2i\pi\left(\frac{1}{2}y_8y_9y_{12}\right)}\,|y_4y_5\oplus y_8y_9\rangle$
- $t \xrightarrow[\mathrm{HHnl}(y_4,y_8)]{} 2\sum e^{2i\pi\left(\frac{1}{2}y_9y_4y_5y_6+\frac{1}{2}y_0y_6+\frac{1}{2}y_9y_4y_6+\frac{1}{2}y_9y_{12}y_4+\frac{1}{2}y_4y_5y_6y_9y_{12}+\frac{1}{2}y_4y_5y_6\right)}\,|y_0\rangle$

  $\xrightarrow[\mathrm{HH}(y_6,[y_0\leftarrow y_9y_{12}y_4y_5\oplus y_9y_4\oplus y_9y_4y_5\oplus y_4y_5])]{} 4\sum e^{2i\pi\left(\frac{1}{2}y_9y_{12}y_4\right)}\,|y_9y_{12}y_4y_5\oplus y_9y_4\oplus y_9y_4y_5\oplus y_4y_5\rangle$
- $t \xrightarrow[\mathrm{HHnl}(y_6,y_{12})]{} 2\sum e^{2i\pi\left(\frac{1}{2}y_0y_8y_9y_6+\frac{1}{2}y_0y_6+\frac{1}{2}y_8y_9y_6+\frac{1}{2}y_4y_5y_6y_8y_9+\frac{1}{2}y_4y_5y_6\right)}\,|y_0\rangle$

  $\xrightarrow[\mathrm{HHgen}(y_6,[y_0\leftarrow y_4y_5\oplus y_8y_9\oplus y_4y_5y_8y_9])]{} 2\sum e^{2i\pi\left(\frac{1}{2}y_8y_9y_6\right)}\,|y_4y_5\oplus y_8y_9\oplus y_4y_5y_8y_9\rangle$ ◀

Another important downside is the potential explosion of the size of the phase polynomial:

▶ **Lemma 21.** *Applying (HHnl) k times in a row on an SOP morphism with phase polynomial of size $O(k)$ may give a morphism with phase polynomial of size $O(2^k)$.*

Proof. For any $k \geq 1$ we can define the following term:

$$t_k := \sum e^{2i\pi\sum_{i=0}^{k}\frac{y_{i0}}{2}(y_{i1}+y_{i2}+1)}$$

on which we can apply (HHnl) k times in a row. In that case we end up with:

$$t_k \to^k 2^k \sum e^{2i\pi\left(\frac{y_0}{2}\prod_{i=0}^{k}(y_{i1}+y_{i2})+1\right)}$$

While t_k has only $3(k+1)$ terms (each of degree at most 2) in its phase polynomial, it can rewrite into a morphism with $2^{k+1}+1$ terms (each of degree at most 3). ◀

Hence, if one were to perform simplifications with this rewrite system, they ought to give special attention as to where and in which order to apply the rules.

6 Completeness for the Dyadic Fragment

We show here how we can turn an $\mathbf{SOP}[\frac{1}{2^{n+1}}]$-morphism into an $\mathbf{SOP}[\frac{1}{2^n}]$-morphism in a "reversible" manner. This will allow us to extend the completeness result to all the restrictions $\mathbf{SOP}[\frac{1}{2^n}]$. This is particularly interesting as the phase gates with dyadic multiples of π, used in particular in the quantum Fourier transform, belong in these fragments:

$$R_Z\left(p\frac{\pi}{2^k}\right) := \sum_{y_0} e^{2i\pi\cdot\frac{p}{2^{k-1}}}\,|y_0\rangle\langle y_0|$$

6.1 Ascending the Dyadic Levels

These transformations between restrictions of **SOP** are more easily defined on **SOP**-morphisms of a particular shape, namely, when their phase polynomial is reduced to a single monomial. Because of this, we show how a **SOP**-morphism can be turned into a composition of these.

▶ **Lemma 22.** *Let* $P = \sum m_i \in \mathbb{R}[X_1, \dots, X_k]/(X_i^2 - X_i)$*, and* $t = s \sum e^{2i\pi P} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right|$*. Then:*

$$\left[\begin{array}{l} \left(s \sum \left|\vec{O}\right\rangle\!\langle y_0, ..., y_k|\right) \circ \\ \left(\sum e^{2i\pi m_1} |y_0, ..., y_k\rangle\!\langle y_0, ..., y_k|\right) \circ \dots \circ \left(\sum e^{2i\pi m_\ell} |y_0, ..., y_k\rangle\!\langle y_0, ..., y_k|\right) \\ \qquad\qquad \circ \left(\sum |y_0, ..., y_k\rangle\!\left\langle\vec{I}\right|\right) \end{array}\right] \xrightarrow[\text{HH}]{*} t$$

Notice that this decomposed form is not unique, as different orderings on the monomials of P define different orderings of the compositions. However, this will not matter.

A particular care is sadly needed for the overall scalar. Because of this, we will first focus on a slightly different notion of restriction of **SOP**.

▶ **Definition 23 (SOP$[\frac{1}{2^n}]'$).** *We define* **SOP**$[\frac{1}{2^n}]'$ *as the restriction of* **SOP** *to morphisms of the form:* $t = \dfrac{1}{2^p} \sum e^{2i\pi \frac{P}{2^n}} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right|$ *where* P *has integer coefficients.*

The only difference with **SOP**$[\frac{1}{2^n}]$ is that the overall scalar is now a power of $\frac{1}{2}$ and not of $\frac{1}{\sqrt{2}}$. There always exists a **SOP**$[\frac{1}{2^n}]'$-morphism that represents the same linear map as any **SOP**$[\frac{1}{2^n}]$-morphism.

▶ **Lemma 24.** $\left[\!\left[\frac{1}{\sqrt{2}} \sum_{y_0 \in V} e^{2i\pi\left(\frac{1}{8} + \frac{3}{4} y_0\right)}\right]\!\right] = 1$*. Hence:*

$$\forall t \in \mathbf{SOP}[\tfrac{1}{2^n}],\ \exists t' \in \mathbf{SOP}[\tfrac{1}{2^{\max(3,n)}}]',\quad [\![t]\!] = [\![t']\!]$$

Proof. If $t \in \mathbf{SOP}[\frac{1}{2^n}]$ and $t \notin \mathbf{SOP}[\frac{1}{2^n}]'$, then:
$t' := t \otimes \left(\frac{1}{\sqrt{2}} \sum e^{2i\pi\left(\frac{1}{8} + \frac{3}{4} y_0\right)}\right) \in \mathbf{SOP}[\frac{1}{2^{\max(3,n)}}]'$ and $[\![t']\!] = [\![t]\!]$. ◀

We can now define the family of maps that will link the different levels of the "dyadic levels":

▶ **Definition 25.** *For any* $k \geq 1$*, we define the functor* $\lfloor \cdot \rceil_k : \mathbf{SOP}[\frac{1}{2^{k+1}}]' \to \mathbf{SOP}[\frac{1}{2^k}]'$*, first for morphisms* $t = s \sum e^{2i\pi \frac{\ell}{2^{k+1}} y_{i_1} \dots y_{i_q}} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right|$ *with phase polynomial of size 0 or 1:*

$$t \mapsto \begin{cases} s \sum e^{2i\pi \frac{\ell/2}{2^k} y_{i_1} \dots y_{i_q}} \left|\vec{O}, y'\right\rangle\!\left\langle\vec{I}, y'\right| = t \otimes id & \text{if } \ell \bmod 2 = 0 \\ s \sum e^{2i\pi \frac{y_{i_1} \dots y_{i_q}}{2^k}\left((\ell-1)/2 + y'\right)} \left|\vec{O}, y'\right\rangle\!\left\langle\vec{I}, y' \oplus y_{i_1} \dots y_{i_q}\right| & \text{if } \ell \bmod 2 = 1 \end{cases}$$

The functor is then extended to any **SOP**$[\frac{1}{2^{k+1}}]'$*-morphism by the decomposition of Lemma 22 (and given a particular ordering on the monomials of the phase polynomial).*

Since $\lfloor \cdot \rceil_k$ is defined to be a functor, we have $\lfloor \cdot \circ \cdot \rceil_k = \lfloor \cdot \rceil_k \circ \lfloor \cdot \rceil_k$. We can show that the ordering of the monomials has no real importance. Indeed, suppose $t_1 = \sum e^{2i\pi \frac{\ell_1}{2^{k+1}} y_{i_1} \dots y_{i_q}} |\vec{y}\rangle\!\langle\vec{y}|$ and $t_2 = \sum e^{2i\pi \frac{\ell_2}{2^{k+1}} y_{j_1} \dots y_{j_r}} |\vec{y}\rangle\!\langle\vec{y}|$. Then: $\lfloor t_1 \circ t_2 \rceil_k = \lfloor t_2 \circ t_1 \rceil_k$ quite obviously when either $\ell_1 \bmod 2 = 0$ or $\ell_2 \bmod 2 = 0$, but also when $\ell_1 \bmod 2 = \ell_2 \bmod 2 = 1$:

$$\lfloor t_1 \circ t_2 \rfloor_k \xrightarrow[\text{HH}]{} \sum e^{2i\pi \left(\begin{array}{r} \frac{y_{i_1}\dots y_{i_q}}{2^k}\big((\ell_1-1)/2+y'\big)+\frac{y_{j_1}\dots y_{j_r}}{2^k}\big((\ell_2-1)/2+y'\big) \\ +\frac{y_{i_1}\dots y_{i_q}y_{j_1}\dots y_{j_r}}{2^k}(1-2y') \end{array} \right)} |\vec{y},y'\rangle\langle\vec{y},y'\oplus y_{i_1}\dots y_{i_q}\oplus y_{j_1}\dots y_{j_r}| \xrightarrow[\text{HH}]{} \lfloor t_2 \circ t_1 \rfloor_k$$

Notice however that $\lfloor \cdot \rfloor_k$ adds an input and an output, so necessarily $\lfloor \cdot \otimes \cdot \rfloor_k \neq \lfloor \cdot \rfloor_k \otimes \lfloor \cdot \rfloor_k$.

The functors $\lfloor \cdot \rfloor_k$ map terms with the same semantics to terms with the same semantics:

▶ **Proposition 26.** $\forall t_1, t_2 \in \mathbf{SOP}[\frac{1}{2^{k+1}}]', \quad [\![t_1]\!] = [\![t_2]\!] \implies [\![\lfloor t_1 \rfloor_k]\!] = [\![\lfloor t_2 \rfloor_k]\!]$

6.2 Going Back

We now show how to reverse the functors $\lfloor \cdot \rfloor_k$.

▶ **Definition 27.** *For any $k \geq 1$, we define the (partial) map $\lceil \cdot \rceil_k : \mathbf{SOP}[\frac{1}{2^k}]' \to \mathbf{SOP}[\frac{1}{2^{k+1}}]'$ as:*

$$\forall t : n+1 \to m+1 \in \mathbf{SOP}[\tfrac{1}{2^k}]', \quad \lceil t \rceil_k := (id_m \otimes \langle 0|) \circ t \circ (id_n \otimes \sum e^{2i\pi \frac{y_0}{2^{k+1}}} |y_0\rangle)$$

Notice that $\lceil \cdot \rceil_k$ can only be applied on morphisms that have at least one input and one output.

$\lceil \cdot \rceil_k$ reverses the action of $\lfloor \cdot \rfloor_k$ (up to some rewrites):

▶ **Proposition 28.** $\lceil \lfloor \cdot \rfloor_k \rceil_k \underset{\text{TH}}{\sim} (\cdot)$ *and* $t_1 \underset{\text{TH}}{\sim} t_2 \implies \lceil t_1 \rceil \underset{\text{TH}}{\sim} \lceil t_2 \rceil$ *for any two terms* t_1, t_2.

6.3 Completeness

We may now show completeness first for $\mathbf{SOP}[\frac{1}{2^{k+1}}]'$ and then tweak the equational theory to extend the result to $\mathbf{SOP}[\frac{1}{2^{k+1}}]$.

▶ **Theorem 29** (Completeness of $\mathbf{SOP}[\frac{1}{2^{k+1}}]'/\underset{\text{TH}}{\sim}$)**.**

$$\forall t_1, t_2 \in \mathbf{SOP}[\tfrac{1}{2^{k+1}}]', \quad [\![t_1]\!] = [\![t_2]\!] \iff t_1 \underset{\text{TH}}{\sim} t_2$$

Proof. Let $t_1, t_2 \in \mathbf{SOP}[\frac{1}{2^{k+1}}]'$ such that $[\![t_1]\!] = [\![t_2]\!]$. By Proposition 26:

$$[\![\lfloor \dots \lfloor t_1 \rfloor_k \dots \rfloor_1]\!] = [\![\lfloor \dots \lfloor t_2 \rfloor_k \dots \rfloor_1]\!]$$

Since $\lfloor \dots \lfloor t_i \rfloor_k \dots \rfloor_1 \in \mathbf{SOP}[\frac{1}{2}]' \subset \mathbf{SOP}[\frac{1}{2}]$, by completeness of this fragment (Theorem 19):

$$\lfloor \dots \lfloor t_1 \rfloor_k \dots \rfloor_1 \underset{\text{TH}}{\sim} \lfloor \dots \lfloor t_2 \rfloor_k \dots \rfloor_1$$

Finally, by Proposition 28: $t_1 \underset{\text{TH}}{\sim} \lceil \dots \lceil \lfloor \dots \lfloor t_1 \rfloor_k \dots \rfloor_1 \rceil_1 \dots \rceil_k \underset{\text{TH}}{\sim} \lceil \dots \lceil \lfloor \dots \lfloor t_2 \rfloor_k \dots \rfloor_1 \rceil_1 \dots \rceil_k \underset{\text{TH}}{\sim} t_2$. ◀

This is not entirely satisfactory, as we would like to relate any two morphisms of the same interpretation. However:

▶ **Lemma 30.** *If $t_1 \in \mathbf{SOP}[\frac{1}{2^{k+1}}]'$ and $t_2 \in \mathbf{SOP}[\frac{1}{2^{k+1}}] \setminus \mathbf{SOP}[\frac{1}{2^{k+1}}]'$, then $t_1 \not\underset{\text{TH}}{\sim} t_2$.*

Proof. There is no rule in $\xrightarrow[\text{TH}]{}$ that changes the overall scalar from an odd power of $\frac{1}{\sqrt{2}}$ to an even one, or vice-versa. ◀

However, adding a single rule:

$$\sum_{\vec{y}} e^{2i\pi\left(\frac{1}{8}+\frac{3}{4}y_0+R\right)} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right| \xrightarrow[y_0\notin \mathrm{Var}(R,\vec{O},\vec{I})]{} \sqrt{2} \sum_{\vec{y}\setminus\{y_0\}} e^{2i\pi R} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right| \tag{$\sqrt{2}$}$$

fixes this caveat. This rule can also be recovered from the more general one:

$$\sum_{\vec{y}} e^{2i\pi\left(\frac{y_0}{4}+\frac{y_0}{2}\widehat{Q}+R\right)} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right| \xrightarrow[y_0\notin \mathrm{Var}(Q,R,\vec{O},\vec{I})]{} \sqrt{2} \sum_{\vec{y}\setminus\{y_0\}} e^{2i\pi\left(\frac{1}{8}-\frac{1}{4}\widehat{Q}+R\right)} \left|\vec{O}\right\rangle\!\left\langle\vec{I}\right| \tag{ω}$$

which was already used in [3, 18, 25] to deal with the Clifford fragment of quantum mechanics.

With this additional rule at hand, we can derive the general completeness theorem:

▶ **Theorem 31** (Completeness of $\mathbf{SOP}[\frac{1}{2^{k+1}}]/\underset{\text{TH},}{\sim}$). *Let us write* $\xrightarrow[\text{TH},]{} := \xrightarrow[\text{TH}]{} + \{(\sqrt{2})\}$. *Then:* $\forall t_1, t_2 \in \mathbf{SOP}[\frac{1}{2^{k+1}}],\quad [\![t_1]\!] = [\![t_2]\!] \iff t_1 \underset{\text{TH},}{\sim} t_2$

Proof. Let $t_1, t_2 \in \mathbf{SOP}[\frac{1}{2^{k+1}}]$ such that $[\![t_1]\!] = [\![t_2]\!]$. Let us also write:

$$t_{\sqrt{2}} := \frac{1}{\sqrt{2}} \sum e^{2i\pi\left(\frac{1}{8}+\frac{3}{4}y_0\right)}$$

We define t_i' as: $\quad t_i' := \begin{cases} t_i & \text{if } t_i \in \mathbf{SOP}[\frac{1}{2^{k+1}}]' \\ t_i \otimes t_{\sqrt{2}} & \text{if } t_i \notin \mathbf{SOP}[\frac{1}{2^{k+1}}]' \end{cases}$.

It is easy to check that $t_i' \in \mathbf{SOP}[\frac{1}{2^{\max(3,k+1)}}]'$ and that $t_i \underset{\text{TH},}{\sim} t_i'$. By Theorem 29:

$$t_1 \underset{\text{TH},}{\sim} t_1' \underset{\text{TH},}{\sim} t_2' \underset{\text{TH},}{\sim} t_2$$ ◀

We hence have completeness for all dyadic fragments of quantum computation. By taking their union, we can get completeness for the "whole dyadic fragment".

▶ **Definition 32.** *Let* $\mathbf{SOP}[\mathbb{D}] := \bigcup_{k=1}^{\infty} \mathbf{SOP}[\frac{1}{2^k}]$ *be the whole dyadic fragment of quantum computation.*

▶ **Corollary 33** (Completeness of $\mathbf{SOP}[\mathbb{D}]/\underset{\text{TH},}{\sim}$).

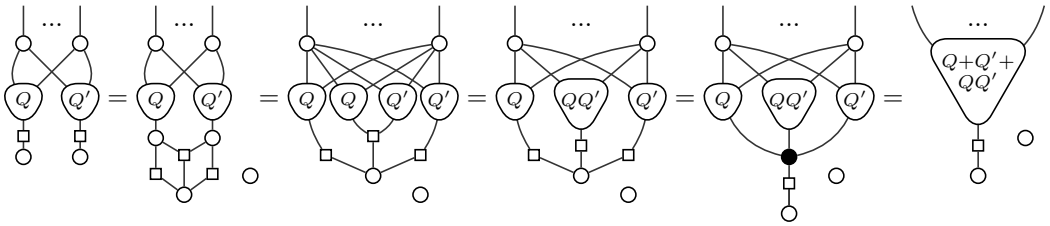
$$\forall t_1, t_2 \in \mathbf{SOP}[\mathbb{D}],\quad [\![t_1]\!] = [\![t_2]\!] \iff t_1 \underset{\text{TH},}{\sim} t_2$$

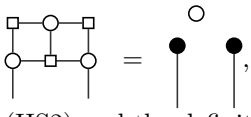
7 Conclusion and Discussion

We have given a new rewrite system for the Toffoli-Hadamard fragment of Sums-Over-Paths, and showed the induced equational theory to be complete. We then extended this rewrite strategy by adding a single new rewrite, which we then proved to be complete for the whole dyadic fragment. As expected from the universality of the fragments at hand, we do not get all the nice properties of the rewriting in the Clifford fragment. In particular, we showed that the rewrite strategies given above are not confluent, and that the size of the terms may grow exponentially when rules are applied carelessly. Whether one of the above two drawbacks can be removed by a different rewrite system remains an open question.

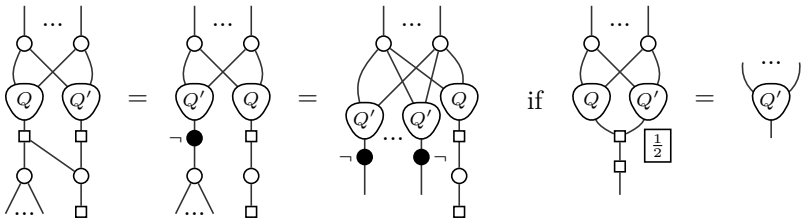
Using the translation from **SOP** to **ZH**, this time, we can make sense of the **SOP** rewrite rules as graphical ones. We will focus on the two rules that were not present in the previous works on **SOP**, namely (HHgen) and (HHnl). Let us start with the latter.

(HHnl) turns an occurrence of $\frac{y_0}{2}\widehat{Q}+\frac{y_0'}{2}\widehat{Q}'$ into $\frac{y_0}{2}(\widehat{Q}+\widehat{Q}'+\widehat{Q}\widehat{Q}')$, when the two variables are linked to nothing else than their respective polynomials Q and Q'. The induced **ZH** identity can be derived using its rules:



(where the first equality uses equality  , the second, third and last use Lemma 11, and the fourth uses (ZS1), (HS2) and the definition of the black node).
Although the overall number of nodes usually increases, the number of white nodes that amount to **SOP**-variables (i.e. white nodes that are not part of a polynomial) decreases.

Rule (HHgen) is a bit more tricky to deal with in particular as it involves a non-trivial side condition. Hence, we do not provide a derivation of the equality, but only state it. With the pattern $\frac{y_0}{2}(y_i\widehat{Q}+\widehat{Q}'+1)$ we get $\frac{y_0}{2}(y_i\widehat{Q}+1)$ with all other occurrences of y_i replaced by $Q'\oplus 1$:



This paper, together with the above small study of how the rewrites translate as ZH transformations, really shows how the two formalisms (SOP and ZH) give different and complementary approaches to rewriting and simplifying representations of quantum processes.

We provided new rewrites that allow simplification in the terms – in that they decrease the number of variables – with the aim of completeness. A next important step for verification, simulation and simplification using SOP is to determine which rewrites, or which variants, are the most relevant to the task at hand.

References

1 Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Phys. Rev. A*, 70:052328, November 2004. doi:10.1103/PhysRevA.70.052328.

2 Dorit Aharonov. A simple proof that Toffoli and Hadamard are quantum universal. *eprint arXiv*, January 2003. arXiv:quant-ph/0301040.

3 Matthew Amy. Towards large-scale functional verification of universal quantum circuits. In Peter Selinger and Giulio Chiribella, editors, *Proceedings of the 15th International Conference on Quantum Physics and Logic, Halifax, Canada, 3-7th June 2018*, volume 287 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–21, 2019. doi:10.4204/EPTCS.287.1.

4 Miriam Backens and Aleks Kissinger. ZH: A complete graphical calculus for quantum computations involving classical non-linearity. In Peter Selinger and Giulio Chiribella, editors, *Proceedings of the 15th International Conference on Quantum Physics and Logic, Halifax, Canada, 3-7th June 2018*, volume 287 of *Electronic Proceedings in Theoretical Computer Science*, pages 23–42, 2019. doi:10.4204/EPTCS.287.2.

5 Miriam Backens, Aleks Kissinger, Hector Miller-Bakewell, John van de Wetering, and Sal Wolffs. Completeness of the ZH-calculus, 2021.

6 Adam D. Bookatz. QMA-complete problems. *Quantum Information and Computation*, 14(5&6):361–383, May 2014. doi:10.26421/qic14.5-6-1.

7 Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. An automated deductive verification framework for circuit-building quantum programs. In Nobuko Yoshida, editor, *Programming Languages and Systems*, pages 148–177, Cham, 2021. Springer International Publishing.

8 Christophe Chareton, Sébastien Bardin, Dongho Lee, Benoît Valiron, Renaud Vilmart, and Zhaowei Xu. Formal methods for quantum programs: A survey, 2021. doi:10.48550/arXiv.2109.06493.

9 Bob Coecke and Ross Duncan. Interacting quantum observables: Categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, April 2011. doi:10.1088/1367-2630/13/4/043016.

10 Bob Coecke and Aleks Kissinger. The compositional structure of multipartite quantum entanglement. In *Automata, Languages and Programming*, pages 297–308. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-14162-1_25.

11 Niel de Beaudrap, Xiaoning Bian, and Quanlong Wang. Fast and Effective Techniques for T-Count Reduction via Spider Nest Identities. In Steven T. Flammia, editor, *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, volume 158 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:23, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TQC.2020.11.

12 Amar Hadzihasanovic. A diagrammatic axiomatisation for qubit entanglement. In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 573–584, July 2015. doi:10.1109/LICS.2015.59.

13 Mohammad Haidar, Marko J. Rančić, Thomas Ayral, Yvon Maday, and Jean-Philip Piquemal. Open source variational quantum eigensolver extension of the quantum learning machine (qlm) for quantum chemistry, 2022. doi:10.48550/arXiv.2206.08798.

14 Dominik Janzing, Pawel Wocjan, and Thomas Beth. Non-identity check is qma-complete. *International Journal of Quantum Information*, 03(03):463–473, September 2005. doi:10.1142/s0219749905001067.

15 Aleks Kissinger and John van de Wetering. Reducing the number of non-Clifford gates in quantum circuits. *Phys. Rev. A*, 102:022406, August 2020. doi:10.1103/PhysRevA.102.022406.

16 Stephen Lack. Composing PROPs. In *Theory and Applications of Categories*, volume 13, pages 147–163, 2004. URL: http://www.tac.mta.ca/tac/volumes/13/9/13-09abs.html.

17 Louis Lemonnier. Relating high-level frameworks for quantum circuits. Master's thesis, Radbound University, 2019. URL: https://www.cs.ox.ac.uk/people/aleks.kissinger/papers/lemonnier-high-level.pdf.

18 Louis Lemonnier, John van de Wetering, and Aleks Kissinger. Hypergraph simplification: Linking the path-sum approach to the ZH-calculus. In Benoît Valiron, Shane Mansfield, Pablo Arrighi, and Prakash Panangaden, editors, Proceedings 17th International Conference on *Quantum Physics and Logic,* Paris, France, June 2 - 6, 2020, volume 340 of *Electronic Proceedings in Theoretical Computer Science*, pages 188–212. Open Publishing Association, 2021. doi:10.4204/EPTCS.340.10.

19 Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5. Springer Science & Business Media, 2013.

20 Yosuke Sato, Shutaro Inoue, Akira Suzuki, Katsusuke Nabeshima, and Ko Sakai. Boolean gröbner bases. *J. Symb. Comput.*, 46(5):622–632, May 2011. doi:10.1016/j.jsc.2010.10.011.

21 Peter Selinger. A survey of graphical languages for monoidal categories. In *New Structures for Physics*, pages 289–355. Springer, 2010.

22 Yaoyun Shi. Both Toffoli and controlled-not need little help to do universal quantum computing. *Quantum Information & Computation*, 3(1):84–92, 2003. URL: http://portal.acm.org/citation.cfm?id=2011515.

23 John van de Wetering and Sal Wolffs. Completeness of the Phase-free ZH-calculus, April 2019. arXiv:1904.07545.

24 Renaud Vilmart. A ZX-calculus with triangles for Toffoli-Hadamard, Clifford+T, and beyond. In Peter Selinger and Giulio Chiribella, editors, *Proceedings of the 15th International Conference on Quantum Physics and Logic, Halifax, Canada, 3-7th June 2018*, volume 287 of *Electronic Proceedings in Theoretical Computer Science*, pages 313–344, 2019. doi:10.4204/EPTCS.287.18.

25 Renaud Vilmart. The structure of sum-over-paths, its consequences, and completeness for clifford. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures*, pages 531–550, Cham, 2021. Springer International Publishing.

26 Fabio Zanasi. *Interacting Hopf Algebras – the theory of linear systems*. PhD thesis, Université de Lyon, 2015. URL: http://www.zanasi.com/fabio/files/thesis2015.pdf.

String Diagrams for Non-Strict Monoidal Categories

Paul Wilson
University of Southampton, UK

Dan Ghica
University of Birmingham, UK

Fabio Zanasi
University College London, UK

Abstract

Whereas string diagrams for strict monoidal categories are well understood, and have found application in several fields of Computer Science, graphical formalisms for non-strict monoidal categories are far less studied. In this paper, we provide a presentation by generators and relations of string diagrams for non-strict monoidal categories, and show how this construction can handle applications in domains such as digital circuits and programming languages. We prove the correctness of our construction, which yields a novel proof of Mac Lane's strictness theorem. This in turn leads to an elementary graphical proof of Mac Lane's *coherence* theorem, and in particular allows for the inductive construction of the canonical isomorphisms in a monoidal category.

2012 ACM Subject Classification Theory of computation

Keywords and phrases String Diagrams, Strictness, Coherence

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.37

Related Version *Extended Version*: https://arxiv.org/abs/2201.11738

1 Introduction

String diagrams are a rigorous graphical notation for category theory that is proving useful in a broad variety of application domains, such as quantum systems [9], computational linguistics [8], digital circuits [12], or signal flow analysis [5]. What the majority of string diagrammatic notations have in common is that they are devised for monoidal categories in which the tensor is *strict*, i.e. the associator and unitor morphisms are identities. As Joyal and Street explain in their seminal *Geometry of Tensor Calculus* [16], the choice of using a strict monoidal category was motivated by convenience ("simplicity of exposition") and by a wish to focus on "aspects other than the associativity of tensor product". Furthermore, they believed that "most results obtained with the hypothesis that a tensor category is strict can be reformulated and proved without this condition."

Indeed, in terms of mathematical power, this statement is true. However, string diagrams have been used increasingly as a convenient *syntax* for languages with models in (strict) monoidal categories. And, when used as syntax, the distinction between strict and non-strict tensor becomes relevant, if not in terms of mathematical expressiveness then at least as a mechanism of abstraction. This is why modern programming languages, and even some modern hardware design languages such SystemVerilog [22], use non-strict features such as *tuples* and *structs* which can nest in non-trivial ways. These non-strict structures could be manually "strictified" by the programmer by flattening them into arrays. Using such programmer conventions instead of native syntactic support does not entail a loss of expressiveness, but a loss of code readability, convenience, and general programmer effectiveness.

In this paper, we address the problem of expanding the graphical language of string diagrams with the required features that allow the expression of non-strict tensors. What makes the language of strict tensors convenient for the graphical representation is that objects are naturally represented as *lists of wires*. This suggests that string diagrams make use of strictness in an essential way and, indeed, naive attempts to define string diagram languages for non-strict monoidal categories can render the notation so heavy-going as to lose the intuitiveness that makes it so attractive in the first place. A more sophisticated solution, which we propose here, is to deliberately use the strictification of a possibly non-strict monoidal category in order to make string diagrams function in this setting with a minimum of additional overhead. These points will be illustrated with examples in Section 2.

Concretely, the basic idea is to use new operations to "pack" pairs of wires into single wires with internal tensorial structure and to "unpack" structured wires into pairs of wires labelled with the tensor component objects. The repeated application of unpacking can flatten any wire with an arbitrarily complex tensor structure into a list of wires labelled with elementary objects. Other new operations are used to "hide" or "reveal" wires labelled with the tensor unit. These four families of new operations are used to define the associators and the unitors of the strictified category.

1.1 Contributions

We propose a strictification construction yielding a graphical language for non-strict monoidal categories. With respect to traditional string diagrams, it provides a more fine-grained representation of tensoring, whose usefulness we demonstrate in motivating examples drawn from circuit theory and programming language semantics. The bulk of the paper is then dedicated to showing that the construction is correct, i.e. the strictified category in which string diagrams live is monoidally equivalent to the original non-strict category. Our proof of monoidal equivalence is new: in contrast to Mac Lane's we do not rely on the coherence theorem, and instead construct the functors of the equivalence explicitly. Consequently, we are able to give a new elementary proof of the coherence theorem: we show *graphically* that the free monoidal category on a single generator forms a preorder. The remainder of the coherence result is largely a reformulation Mac Lane's original corollary, but in a way that we believe has pedagogical value. For brevity, we reserve discussion of the corollary for an extended version of the paper [23].

1.2 Related Work

The use of adapter morphisms which can "pack" and "unpack" wires has been explored in various forms. Our adapters can be recovered as instances of more sophisticated constructs used in the study of coherence of weakly distributive categories [3]. These categories use two distinct tensors and an additional kind of wire ("thinning links") in their string diagram language. If this additional structure were to be somehow omitted the remaining constructs would be rather similar to ours. However, this is not explored in [3], and it is not obvious how this streamlined setting would lead to coherence results for "mere" monoidal categories as a corresponding simplification of the coherence of weakly distributive categories. A non-diagrammatic approach giving a type theory for symmetric monoidal categories can also be found in [21]. The idea also appears in the study of quantum [6] and reversible [7] circuits, although these do not study the connection to the strictness and coherence theorems. The distinctive feature of our paper is to show explicitly how introducing adapters allows non-strict categories to take advantage of graph based datastructures for *strict* monoidal

categories such as those of [24]. More concretely, by *explicitly* defining the functors mapping between a non-strict category and its "strictified" counterpart, we obtain datastructure-independent algorithms for translating between strict and non-strict settings. In addition, we formulate our approach using presentations by generators and relations, which brings the strictness question much closer to current graphical calculi approaches to circuits such as [17, 25, 13, 4, 2]. Finally, our approach allows us to give a self-contained proof of Mac Lane's strictness theorem that does not rely on the coherence theorem, allowing us to avoid its associated pitfalls.

1.3 Synopsis

In Section 2 we present our graphical calculus for (non-strict) monoidal categories, in the form of a strictification procedure. Subsections 2.1, 2.2, and 2.3 illustrate a series of motivating examples. Section 3 justifies our construction by proving that it yields an equivalence of categories. Section 4 revisits MacLane's Coherence theorem and some of its consequences in light of the approach we presented. Section 5 is dedicated to conclusions and future work.

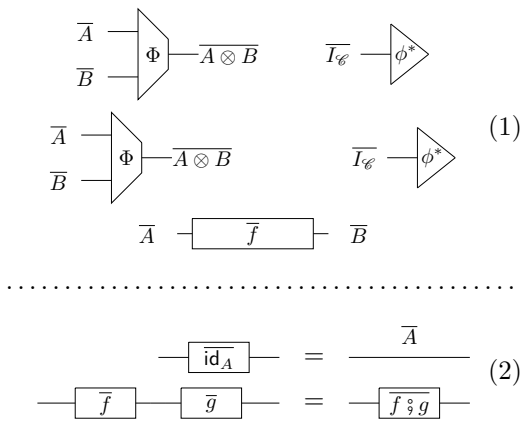
2 A graphical language for (non-strict) monoidal categories

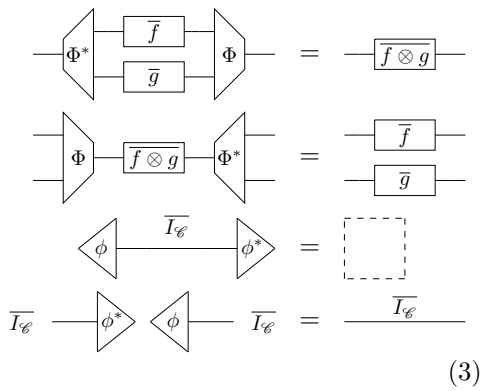
We assume familiarity with string diagrams for strict monoidal categories, see e.g. [20]. Let us fix an arbitrary (non-strict) monoidal category $\mathscr{C}$. We construct its *strictification* as the strict monoidal category $\overline{\mathscr{C}}$ defined as follows.

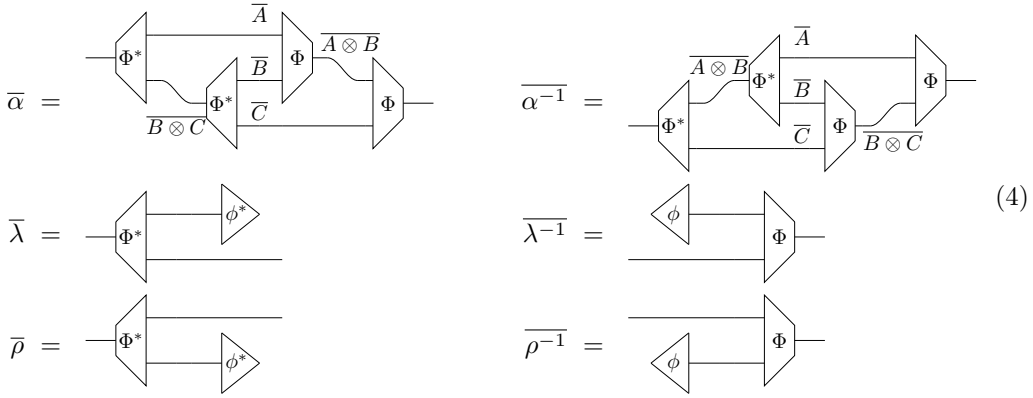
▶ **Definition 1.** *$(\overline{\mathscr{C}}, \bullet)$ is the strict monoidal category freely generated by:*

1. *Objects $\overline{A}$ for each $A \in \mathscr{C}$*
2. *Generators (1), with $\overline{f} : \overline{A} \to \overline{B}$ for each $f : A \to B \in \mathscr{C}$*
3. *functoriality equations (2)*
4. *adapter equations (3), and*
5. *associator/unitor equations (4)*

Note that because $\overline{\mathscr{C}}$ is strict by definition, we are entitled to use string diagrammatic notation. Thus, a morphism with m inputs and n outputs will have domain and codomain of the form $\overline{A_1} \bullet \cdots \bullet \overline{A_m}$ and $\overline{B_1} \bullet \cdots \bullet \overline{B_n}$, respectively.







(4)

This is a functorial construction, yielding a monoidal equivalence between $\mathscr{C}$ and $\overline{\mathscr{C}}$, as we will prove in Section 3. Note that although the category $\overline{\mathscr{C}}$ is essentially the same as that given by Mac Lane [18, p. 257], its construction differs in one key respect. Namely, to define his equivalent strict category, Mac Lane relies on the coherence theorem to define both composition of arrows and to ensure the functors in the equivalence are monoidal. In contrast, the adapter generators and equations of $\overline{\mathscr{C}}$ mean that Definition 1 does not require use of the coherence theorem, and can therefore be used to prove it.

The functoriality equations are so-called as they ensure functoriality of the construction. The "adapter" equations and "associator/unitor" equations further ensure this functor is *monoidal* and it forms one half of a *monoidal equivalence*. Sec. 3 will make it clear that these equations are essentially obtained by freely adding the morphisms required by the definition of a monoidal functor (2).

Besides its mathematical significance, the interest of this construction lies in providing a means of manipulating morphisms of non-strict monoidal categories graphically. In particular, the ϕ and ϕ^* generators can be used to explicitly summon and dispell the monoidal unit, while the Φ and Φ^* generators can be thought of as systematic ways of packing and unpacking wires into more complex wires with internal structure. The next subsections will showcase how this additional layer of structure can be useful in categorical models of computation.

2.1 Circuit Description Languages with Tuples

Categorical models of circuit description languages are a prime source of examples of monoidal categories, for instance combinational [17] or sequential [12] circuits. The graphical representation of circuits also fits naturally and intuitively the box-and-wire model used by string diagrams. More precisely, the circuit description languages in *loc. cit.* (and variations thereof) are instances of *strict* monoidal categories.

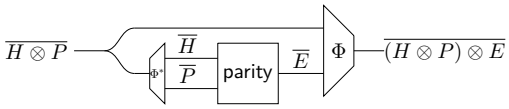
From the point of view of *expressiveness*, i.e. realising circuits with certain desired behaviours, the strict setting does not introduce any limitations. Consistent with this observation, standard hardware description languages (HDL) such as Verilog can also be modelled using a strict monoidal tensor. However, larger and more complex designs stand to benefit from the additional level of structure which a non-strict tensor can offer and, indeed, more modern HDLs, intended for more complex designs, such as SystemVerilog have syntactic facilities which require a non-strict tensor: *struct*s.

Consider the following simple example. Suppose that some circuitry is needed to process network packets, which consist of a header (of size $h = 96$ bits), a payload (of size $p = 896$ bits) and an error-correcting trailer (of size $e = 32$ bits). In the older Verilog language, the

header and the payload can be combined in a single, wider, data bus of $h + p = 992$ bits, but the two components can only be extracted using numerical indexing. This is a primitive form of "flattening" a data structure into an array, and in the more modern SystemVerilog it can be avoided by using a *struct*. This means that a data type of "message" (say m) can access its components as *fields* (projections), namely $m.h$ and $m.p$. Since structs can have other structs as fields the way in which the components are associated is relevant, which means that the tensor must no longer be strict.

On the other hand, "flattening" the structure of a data bus to an array of bits can be useful. In the current example, in computing the error-correcting code e, the way the message is partitioned into header and payload is no longer relevant, so it is convenient to unpack the tensor $h \otimes p$ into a flat array of $h + p$ wires from which an error-correcting code e is computed by a generic circuit of the appropriate width. Structures that can be flattened like this are called in SystemVerilog *packed structs*, and to model them properly both strict and non-strict tensorial facilities are required in the categorical model.

Finally, the error-correcting code can be packed with the original message into an error-correcting message with three components. It is obviously important to be able to retrieve the header, payload, and error-correcting code separately from the message, and it should be equally obvious that once the internal structure of the message is non-trivial a calculus of indices would be a complicated, awkward, and error-prone way to access the components.



Graphically, this circuit is represented above. In order to make this diagram completely formal, what we are using here is the $\overline{\mathscr{C}}$ construction described in Sec. 2 applied to one of the categories $\mathscr{C}$ of digital circuits (combinational or sequential) mentioned earlier. This gives us the best of both worlds: the "non-strictness" of circuits-with-tuples, and the graphical syntax of string diagrams.

2.1.0.1 Strictifying Strict Categories

The "strictification" procedure is not just useful for providing a graphical syntax for non-strict monoidal categories, but can also provide a more ergonomic syntax for monoidal categories that are *already* strict. Suppose we wish to work in Lafont's strict monoidal category of circuits [17], and suppose we would like to define the "parity" function used earlier. Using our construction, we can define it recursively as follows:

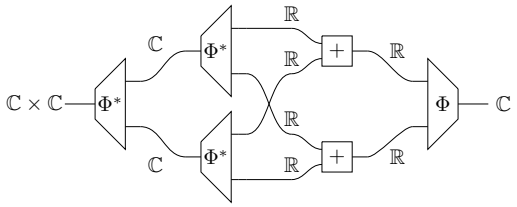
Notice that in the "base" language of Lafont's PROP of circuits we cannot truly depict this diagram, since there is no way to treat a bundle of n wires as a pair of 1 and $n - 1$ wires. To do this formally we require the adapter morphisms as defined in Section 2.

2.2 Programming Languages

Programming languages, largely based on the lambda calculus, commonly include product formation as a syntactic feature. Therefore, a graphical syntax based on its categorical model, as used for example in [1], needs to have a non-strict tensor. However, having only

the non-strict tensor leads to an awkward graphical syntax in which all generators have a single wire going in and a single wire going out. Diagrams in which the interfaces can be intermediated using lists of wires require mechanisms for strictification. This can be realised by applying the strictification construction to a Cartesian closed category, which will allow the expression of examples such as the one below.

Consider the simple task of summing two complex numbers, whose real and imaginary parts are encoded as floating-point numbers. That is, while we have a primitive type of reals, we model complex numbers as pairs $\mathbb{C} = \mathbb{R} \times \mathbb{R}$. A natural way to write such a program in a diagrammatic form is pictured below.



Even in categorical models of the simply-typed λ-calculus (STLC) without product, strictification has a role to play. As usual, this role is cloaked in informality which in some contexts can lead to ambiguity. STLC is interpreting by giving meaning to type judgements $\Gamma \vdash t : T$ with Γ a context, t a term, and T a type. The context $\Gamma = x_1 : T_1, \ldots, x_n : T_n$ is a list of typed variables which is interpreted as the tensor $T_1 \otimes \cdots \otimes T_n$, virtually always treated as if it were strict. This informal strictification can be problematic though when product types are used, as the objects T_i in the interpretation of the context also contain tensors. So the strictification must be fine-grained enough to allow only the flattening of those tensors representing the comma of the context, and not those of the product formation. Our approach offers this level of granularity.

2.3 Strict vs. Non-Strict String Diagrams

Our final example concerns the usability problems of non-strict diagrams *without* strictification and illustrate how our approach to strictification with packing and unpacking wires makes rigorous the intuition that formulating certain properties in terms of strict monoidal categories does not entail a loss of generality. Our example uses braided autonomous categories. Here, each object A has a dual A^*, there exists a family of isomorphisms $c_{A,B} : A \otimes B \to B \otimes A$ called braidings, and families of adjunctions $\eta_A : I \to A^* \otimes A$, $\epsilon : A \otimes A^* \to I$ with certain properties which we may elide in the formulation of the example. Consider the property of braided monoidal categories to be autonomous if and only if they are right-autonomous [15, Prop. 7.2]. The proof is formulated in terms of string diagrams in [20, Lem. 4.17], which makes it more intuitive.

The idea of the proof is to show that isomorphisms $b_A : A^{**} \to A$, $b_A^{-1} : A \to A^{**}$ can be constructed. They are defined as follows:

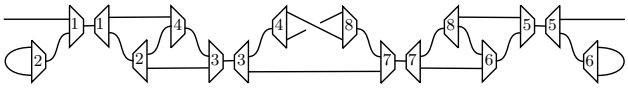
$$b_A = A^{**} \xrightarrow{\eta_A \otimes \mathsf{id}} A^* \otimes A \otimes A^{**} \xrightarrow{\mathsf{id} \otimes c_{A,A^{**}}} A^* \otimes A^{**} \otimes A \xrightarrow{\epsilon_{A^*} \otimes \mathsf{id}} A$$

$$b_A^{-1} = A \xrightarrow{\mathsf{id} \otimes \eta_{A^*}} A \otimes A^{**} \otimes A^* \xrightarrow{c^{-1}_{A^{**},A}} A^{**} \otimes A \otimes A^* \xrightarrow{\mathsf{id} \otimes \epsilon_A} A^{**}.$$

The fact that $b_A ; b_A^{-1} = \mathsf{id}$ becomes elegantly obvious when the terms are rendered as string diagrams which can be manipulated graphically:

The exposition includes the standard caveat that "Here we have written, without loss of generality, as if [the category] were strict monoidal." We shall now show, graphically, that this is indeed the case.

First we note that in the non-strict setting (without strictification) all string diagrams must be equipped with gadgets that make sure that there is a single wire on the left, and a single wire on the right. These gadgets are of course the bundlers and unbundlers introduced earlier. Therefore, in the non-strict setting, taking into account all the relevant associators, the diagram for b_A becomes much more complicated, denying the intuitiveness we expect from a graphical notation (see below).



This is why a naive approach to non-strict string diagram construction is not effective. However, the complications are only an artefact of the construction of the diagram in a purely non-strict setting. The strictification equations come to rescue and, in this case, cancel out all bundler-unbundler pairs in the order indicated by the numerical labels attached to them, resulting in exactly the same diagram of b_A that was constructed in the strict setting. So, indeed, working in the strict setting implied no loss of generality!

3 Strictness

We now show that $\mathscr{C}$ is monoidally equivalent to $\overline{\mathscr{C}}$, constituting a proof of Mac Lane's strictness theorem, since $\mathscr{C}$ is an arbitrary monoidal category. Our approach is to define monoidal functors $\mathcal{S} : \mathscr{C} \to \overline{\mathscr{C}} : \mathcal{N}$, and we begin by recalling the definition of monoidal functor.

▶ **Definition 2** (Monoidal Functor). *Let $(\mathscr{C}, \otimes, I_{\mathscr{C}})$ and $(\mathscr{D}, \bullet, I_{\mathscr{D}})$ be monoidal categories. A monoidal functor is a functor $F : \mathscr{C} \to \mathscr{D}$ equipped with natural isomorphisms $\Phi_{X,Y} : F(X) \bullet F(Y) \to F(X \otimes Y)$ and $\phi : I_{\mathscr{D}} \to F(I_{\mathscr{C}})$ such that the following diagrams commute for all objects $A, B, C \in \mathscr{C}$.*

$$\begin{array}{ccc}
(F(A) \bullet F(B)) \bullet F(C) & \xleftarrow{\alpha_{\mathscr{D}}} & F(A) \bullet (F(B) \bullet F(C)) \\
\downarrow{\scriptstyle \Phi_{A,B} \bullet \mathsf{id}_{F(C)}} & & \downarrow{\scriptstyle \mathsf{id}_{F(A)} \bullet \Phi_{B,C}} \\
F(A \otimes B) \bullet F(C) & & F(A) \bullet F(B \otimes C) \\
\downarrow{\scriptstyle \Phi_{A\otimes B,C}} & & \downarrow{\scriptstyle \Phi_{A,B\otimes C}} \\
F((A \otimes B) \otimes C) & \xleftarrow[F(\alpha_{\mathscr{C}})]{} & F(A \otimes (B \otimes C))
\end{array} \tag{5}$$

$$\begin{array}{ccc}
F(A) \bullet I_{\mathscr{D}} & \xrightarrow{\mathsf{id}_{F(A)} \bullet \phi} & F(A) \bullet F(I_{\mathscr{C}}) \\
\downarrow{\scriptstyle \rho_{\mathscr{D}}} & & \downarrow{\scriptstyle \Phi_{A,I_{\mathscr{C}}}} \\
F(A) & \xleftarrow[F(\rho_{\mathscr{C}})]{} & F(A \otimes I_{\mathscr{C}})
\end{array}
\qquad
\begin{array}{ccc}
I_{\mathscr{D}} \bullet F(B) & \xrightarrow{\phi \bullet \mathsf{id}_{F(B)}} & F(I_{\mathscr{C}}) \bullet F(B) \\
\downarrow{\scriptstyle \lambda_{\mathscr{D}}} & & \downarrow{\scriptstyle \Phi_{I_{\mathscr{C}},B}} \\
F(B) & \xleftarrow[F(\lambda_{\mathscr{C}})]{} & F(I_{\mathscr{C}} \otimes B)
\end{array} \tag{6}$$

With this definition it is straightforward to see how to define a monoidal functor from $\mathscr{C}$ to $\overline{\mathscr{C}}$.

▶ **Definition 3.** *Let $\mathcal{S} : \mathscr{C} \to \overline{\mathscr{C}}$ be the* strictification functor *defined on objects and morphisms as $\mathcal{S}(A) := \overline{A}$ and $\mathcal{S}(f) := \overline{f}$, respectively*

▶ **Proposition 4.** *$(\mathcal{S}, \Phi, \phi)$ is a monoidal functor.*

Proof. $\mathcal{S}$ preserves identities and composition (and is therefore a functor) by the functor equations (2):

$$\mathcal{S}(\mathsf{id}_A) = \overline{\mathsf{id}_A} = \mathsf{id}_{\overline{A}} \qquad \mathcal{S}(f \fatsemi g) = \overline{f \fatsemi g} = \overline{f} \fatsemi \overline{g} = \mathcal{S}(f) \fatsemi \mathcal{S}(g)$$

It is a *monoidal* functor using the adapter generators $\Phi = \Phi$ and $\phi = \phi$ from (1). For this to work, we must have that Φ is a natural isomorphism and ϕ an isomorphism, respectively. This is a straightforward consequence of the adapter equations (3): $\Phi^* \fatsemi (\overline{f} \bullet \overline{g}) = \Phi^* \fatsemi (\overline{f} \bullet \overline{g}) \fatsemi \Phi \fatsemi \Phi^* = \overline{f \otimes g} \fatsemi \Phi^*$ and $\phi \fatsemi \phi^* = \mathsf{id}$ by definition. Similarly, we require that the diagrams of (5) and (6) commute. Again, this is precisely what the the associator/unitor equations (4) state, and so $\mathcal{S}$ is a monoidal functor. ◀

▶ Remark 5. Notice that $\overline{\mathscr{C}}$ is *defined* by freely adding the requirements of Definition 2. Generators Φ and ϕ and equations (3) give the natural isomorphism Φ and isomorphism ϕ, while the commuting diagrams (5) and (6) are precisely the "associator/unitor" equations (4).

We can now define the other half of the monoidal equivalence $\mathcal{S} \dashv \mathcal{N}$. In doing so, we'll make use of the fact that morphisms of a monoidal category can be written in a "sequential normal form" (Appendix A), i.e. as a series of "slices"

$$(\mathsf{id} \otimes g_1 \otimes \mathsf{id}) \fatsemi (\mathsf{id} \otimes g_2 \otimes \mathsf{id}) \fatsemi \ldots \fatsemi (\mathsf{id} \otimes g_n \otimes \mathsf{id})$$

where each g_i is a generator. We take advantage of this form to define $\mathcal{N}$: our definition is defined on "slices" $\mathsf{id}_X \bullet q \bullet \mathsf{id}_Y$ for some generator q, and then freely on composition so that $\mathcal{N}(f \fatsemi g) = \mathcal{N}(f) \fatsemi \mathcal{N}(g)$.

▶ **Definition 6.** *We define the nonstrictification functor $\mathcal{N} : \overline{\mathscr{C}} \to \mathscr{C}$ inductively on objects:*

$$\mathcal{N}(I_{\overline{\mathscr{C}}}) := I_{\mathscr{C}} \qquad \mathcal{N}(\overline{A}) := A \qquad \mathcal{N}(\overline{A} \bullet R) := A \otimes \mathcal{N}(R)$$

And on morphisms we give a recursive definition, with the following base cases:

$$\begin{aligned}
\mathcal{N}(\mathsf{id}_{I_{\overline{\mathscr{C}}}}) &:= \mathsf{id}_{I_{\mathscr{C}}} \\
\mathcal{N}(\overline{f}) &:= f \\
\mathcal{N}(\Phi_{A,B}) &:= \mathsf{id}_{A \otimes B} = \mathcal{N}(\Phi^*_{A,B}) \\
\mathcal{N}(\phi) &:= \mathsf{id}_{I_{\mathscr{C}}} = \mathcal{N}(\phi^*)
\end{aligned}
\qquad
\begin{aligned}
\mathcal{N}(\overline{f} \bullet \mathsf{id}_Y) &:= f \otimes \mathsf{id}_{\mathcal{N}(Y)} \\
\mathcal{N}(\Phi_{A,B} \bullet \mathsf{id}_Y) &:= \alpha_{A,B,\mathcal{N}(Y)} \\
\mathcal{N}(\Phi^*_{A,B} \bullet \mathsf{id}_Y) &:= \alpha^{-1}_{A,B,\mathcal{N}(Y)} \\
\mathcal{N}(\phi \bullet \mathsf{id}_Y) &:= \lambda^{-1}_{\mathcal{N}(Y)} \\
\mathcal{N}(\phi^* \bullet \mathsf{id}_Y) &:= \lambda_{\mathcal{N}(Y)}
\end{aligned}
\qquad
\begin{aligned}
\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet \overline{f}) &:= \mathsf{id}_A \otimes f \\
\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet \Phi_{B,C}) &:= \mathsf{id}_{A \otimes (B \otimes C)} \\
\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet \Phi^*_{B,C}) &:= \mathsf{id}_{A \otimes (B \otimes C)} \\
\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet \phi) &:= \rho^{-1}_A \\
\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet \phi^*) &:= \rho_A
\end{aligned}$$

With a single recursive case, for $q \in \{\Phi, \phi, \Phi^*, \phi^*, \mathsf{id}_{\overline{Q}}\}$

$$\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet q \bullet r) := \mathsf{id}_A \otimes \mathcal{N}(q \bullet r)$$

Finally take $\mathcal{N}(f \fatsemi g) := \mathcal{N}(f) \fatsemi \mathcal{N}(g)$.

This definition is well defined with respect to the equations of Definition 1; we give a proof in Appendix B, where we also note that $N(f)$ is the same regardless of which "sequential normal form" decomposition we choose for f.

▶ Remark 7. The definition of $\mathcal{N}$ can be explained more intuitively in terms of programming. If we think of each "slice" of the sequential normal form as a list of primitive arrows of $\mathscr{C}$, then the definition of $\mathcal{N}$ is essentially a list recursion in which we have a separate case for 1, 2, and n-element lists.

Now we will show that $\mathcal{N}$ is a *monoidal* functor. To do this, we must specify the "coherence maps": a natural isomorphism $\Psi_{X,Y} : \mathcal{N}(X) \otimes \mathcal{N}(Y) \to \mathcal{N}(X \bullet Y)$ and isomorphism $\psi : I_{\mathscr{C}} \to \mathcal{N}(I_{\overline{\mathscr{C}}})$ as mandated by Definition 2.

▶ **Definition 8.** *We define Ψ, the coherence natural isomorphism for $\mathcal{N}$, in the following cases:*

$$\Psi_{I_{\overline{\mathscr{C}}},I_{\overline{\mathscr{C}}}} := \lambda_{I_{\mathscr{C}}} = \rho_{I_{\mathscr{C}}} \qquad \Psi_{X,I_{\overline{\mathscr{C}}}} := \rho_{\mathcal{N}(X)} \qquad \Psi_{I_{\overline{\mathscr{C}}},Y} := \lambda_{\mathcal{N}(Y)}$$

$$\Psi_{\overline{A},Y} := \mathsf{id}_{A \otimes \mathcal{N}(Y)} \qquad \Psi_{\overline{A} \bullet X,Y} := \alpha^{-1}_{A,\mathcal{N}(X),\mathcal{N}(Y)} \fatsemi (\mathsf{id}_A \otimes \Psi_{X,Y})$$

▶ **Definition 9.** *The coherence isomorphism ψ for $\mathcal{N}$ is defined as follows:*

$$\psi_{I_{\mathscr{C}}} := \mathsf{id}_{I_{\mathscr{C}}}$$

▶ Remark 10. Note that both $\lambda_{I_{\mathscr{C}}}$ and $\rho_{I_{\mathscr{C}}}$ have the correct type as a choice for $\Psi_{I_{\overline{\mathscr{C}}},I_{\overline{\mathscr{C}}}}$. In fact, they are equal: unitors coincide at the unit object, i.e. $\lambda_{I_{\mathscr{C}}} = \rho_{I_{\mathscr{C}}}$, as noted in [10, Corollary 2.2.5].

▶ **Proposition 11.** *$(\mathcal{N}, \Psi, \psi)$ is a monoidal functor.*

Proof. It is clear that Ψ and ψ are natural isomorphisms since they are both composites of natural isomorphisms. Thus it remains to check the diagrams of Definition 2 commute.

The squares (6) commute because $\psi = \mathsf{id}$, and $\Psi_{A,I_{\overline{\mathscr{C}}}} = \rho$ and $\Psi_{I_{\overline{\mathscr{C}}},B} = \lambda$ by definition.

Now let us check that the hexagon (5) commutes. Note that in the following we use that $\mathcal{N}(\alpha_{\overline{\mathscr{C}}}) = \mathsf{id}$, because $\overline{\mathscr{C}}$ is strict, and so the hexagon axiom becomes a pentagon.

We will approach the problem inductively, checking base cases where $A = I$ and $A = \overline{A}$, and finally the inductive step with $A = \overline{A} \bullet R$. Let us begin with $A = I$, and taking the outer path of the hexagon we calculate as follows:

$$\begin{aligned}
&(\mathsf{id}_{I_{\mathscr{C}}} \otimes \Psi_{B,C}) \fatsemi \Psi_{I_{\mathscr{C}},B \bullet C} \fatsemi \Psi^{-1}_{B,C} \fatsemi (\Psi_{I_{\mathscr{C}},B} \otimes \mathsf{id}_{\mathcal{N}(C)})^{-1} \\
&\qquad = (\mathsf{id}_{I_{\mathscr{C}}} \otimes \Psi_{B,C}) \fatsemi \lambda_{\mathcal{N}(B \bullet C)} \fatsemi \Psi^{-1}_{B,C} \fatsemi (\lambda_{\mathcal{N}(B)} \otimes \mathsf{id}_{\mathcal{N}(C)})^{-1} \\
&\qquad = \lambda_{\mathcal{N}(B) \otimes \mathcal{N}(C)} \fatsemi (\lambda_{\mathcal{N}(B)} \otimes \mathsf{id}_{\mathcal{N}(C)})^{-1} \\
&\qquad = \alpha_{I_{\mathscr{C}},\mathcal{N}(B),\mathcal{N}(C)}
\end{aligned}$$

Wherein we expanded the definition of Ψ, then used naturality of $\Psi_{B,C}$ before applying the monoidal triangle lemma of [10, (2.12)].

Now consider the second base case, where A is the "singleton list" $\overline{A}$. In this case, the hexagon diagram commutes immediately because $\Psi_{\overline{A},B} = \mathsf{id}_{\overline{A} \otimes \mathcal{N}(B)}$ and $\Psi_{\overline{A},B \bullet C} = \mathsf{id}_{\overline{A} \otimes \mathcal{N}(B \bullet C)}$. More explicitly, we calculate as follows, starting again with the outer path of the hexagon and expanding definitions:

$$\begin{aligned}
&(\mathsf{id}_A \otimes \Psi_{B,C}) \fatsemi \Psi_{A,B \bullet C} \fatsemi \Psi^{-1}_{A \bullet B,C} \fatsemi (\Psi_{\overline{A},B} \otimes \mathsf{id}_{\mathcal{N}(C)}) \\
&\qquad = (\mathsf{id}_A \otimes \Psi_{B,C}) \fatsemi (\mathsf{id}_A \otimes \Psi_{B,C})^{-1} \fatsemi \alpha_{A,\mathcal{N}(B),\mathcal{N}(C)} \\
&\qquad = \alpha_{A,\mathcal{N}(B),\mathcal{N}(C)}
\end{aligned}$$

Finally let us prove the inductive step. Assume that the hexagon commutes for objects R, B, C, giving us the equation

$$\Psi_{R,B\bullet C} \fatsemi \Psi^{-1}_{R\bullet B,C} = (\mathsf{id}_{\mathcal{N}(R)} \otimes \Psi^{-1}_{B,C}) \fatsemi \alpha_{\mathcal{N}(R),\mathcal{N}(B),\mathcal{N}(C)} \fatsemi (\Psi_{R,B} \otimes \mathsf{id}_{\mathcal{N}(C)})$$

We may then rewrite the following subterm of the monoidal hexagon as follows:

$$\mathsf{id}_A \otimes (\Psi_{R,B\bullet C} \fatsemi \Psi^{-1}_{R\bullet B,C}) = \mathsf{id}_A \otimes (\mathsf{id}_{\mathcal{N}(R)} \otimes \Psi^{-1}_{B,C}) \fatsemi \mathsf{id}_A \otimes \alpha_{\mathcal{N}(R),\mathcal{N}(B),\mathcal{N}(C)} \fatsemi \mathsf{id}_A \otimes (\Psi_{R,B} \otimes \mathsf{id}_{\mathcal{N}(C)})$$

We can then rewrite $\mathsf{id}_A \otimes \alpha_{\mathcal{N}(R),\mathcal{N}(B)\mathcal{N}(C)}$ using the monoidal category pentagon axiom, and then use naturality of α to reduce the outer path of the monoidal hexagon until we are left with $\alpha_{A\otimes\mathcal{N}(R),\mathcal{N}(B),\mathcal{N}(C)}$, as required. ◀

Finally, we must check that $\mathcal{S}$ and $\mathcal{N}$ indeed form an equivalence. First, recall the definition

▶ **Definition 12** (Equivalence of categories)**.**
An equivalence is a pair of functors $\mathscr{C} \underset{G}{\overset{F}{\rightleftarrows}} \mathscr{D}$ *and a pair of natural isomorphisms* $\eta : \mathsf{id}_{\mathscr{C}} \to G \circ F$ *and* $\epsilon : F \circ G \to \mathsf{id}_{\mathscr{D}}$*.*

We begin by showing naturality of η.

▶ **Proposition 13.** $\mathcal{N} \circ \mathcal{S} = \mathsf{id}_{\mathscr{C}}$*.*

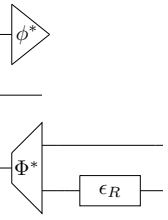
Proof. $\mathcal{N}(\mathcal{S}(f)) = \mathcal{N}(\overline{f}) = f = \mathsf{id}_{\overline{\mathscr{C}}}(f)$ ◀

▶ Remark 14. Note that Proposition 13 shows that the composite $\mathcal{N} \circ \mathcal{S}$ is actually *equal* to the identity functor, and thus $\eta_A = \mathsf{id}_A$. In fact, this will make the composite of the two functors a *split idempotent*.

Now we prove naturality of ϵ. This proof is somewhat more involved: unlike 13, the composite $\mathcal{S} \circ \mathcal{N}$ is merely isomorphic to the identity functor, not equal on the nose. Thus, we begin with an inductive definition:

▶ **Definition 15.** *We define the (monoidal) natural isomorphism* $\epsilon : \mathcal{S} \circ \mathcal{N} \to \mathsf{id}_{\overline{\mathscr{C}}}$ *for the composite* $\mathcal{S} \circ \mathcal{N}$ *inductively:*

$$\begin{aligned} \epsilon_{I_{\overline{\mathscr{C}}}} &:= \phi^* \\ \epsilon_{\overline{A}} &:= \mathsf{id}_{\overline{A}} \\ \epsilon_{\overline{A}\bullet R} &:= \Phi^* \fatsemi (\mathsf{id}_{\overline{A}} \bullet \epsilon_R) \end{aligned} \qquad = \quad \text{(7)}$$



▶ **Proposition 16.** *If* ϵ *is natural for* f *and* g*, then it is natural for* $f \fatsemi g$*.*

Proof. Take morphisms $f : X \to Y$ and $g : Y \to Z$. By assumption, we have:

$$\mathcal{S}(\mathcal{N}(f)) = \epsilon_X \fatsemi f \fatsemi \epsilon_Y^{-1} \qquad \mathcal{S}(\mathcal{N}(g)) = \epsilon_Y \fatsemi g \fatsemi \epsilon_Z^{-1}$$

from which we can derive

$$\begin{aligned} \epsilon_X^{-1} \fatsemi \mathcal{S}(\mathcal{N}(fg)) \fatsemi \epsilon_Z = \epsilon_X^{-1} \fatsemi \mathcal{S}(\mathcal{N}(f) \fatsemi \mathcal{N}(g)) \fatsemi \epsilon_Z &= \epsilon_X^{-1} \fatsemi \mathcal{S}(\mathcal{N}(f)) \fatsemi \mathcal{S}(\mathcal{N}(g)) \fatsemi \epsilon_Z \\ &= \epsilon_X^{-1} \fatsemi \epsilon_X \fatsemi f \fatsemi \epsilon_Y^{-1} \fatsemi \epsilon_Y \fatsemi g \fatsemi \epsilon_Z^{-1} \fatsemi \epsilon_Z \\ &= f \fatsemi g \end{aligned} \quad (8)$$

as required. ◀

▶ **Proposition 17.** *$\epsilon : \mathcal{S} \circ \mathcal{N} \to \mathsf{id}_{\overline{\mathscr{C}}}$ is a monoidal natural isomorphism.*

Proof. We begin by showing naturality inductively, having already proven the inductive step for composition in Proposition 16. We again use Proposition 36–that each morphism f in $\overline{\mathscr{C}}$ can be decomposed into "slices" $f = t_1 \fatsemi \ldots \fatsemi t_n$ with each t_i of the form $\mathsf{id}_X \bullet g_i \bullet \mathsf{id}_Y$, with each $g_i : A \to B$ a generator. It thus suffices to prove that $t = \epsilon^{-1}_{X \bullet A \bullet Y} \fatsemi \mathcal{S}(\mathcal{N}(t)) \fatsemi \epsilon_{X \bullet B \bullet Y}$ for an arbitrary "slice" t. One can check this by a second induction whose base case and inductive step correspond to the definition of $\mathcal{N}$ (Definition 6). To be precise, one can check this property graphically for each base case $\mathcal{N}(\mathsf{id}_{\overline{I_{\mathscr{C}}}}) \ldots \mathcal{N}(\mathsf{id}_{\overline{A}})$, and additionally for the inductive step $\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet q \bullet r)$. Finally, note that ϵ is indeed a *monoidal* natural transformation, which can be verified by another straightforward induction. ◀

▶ **Theorem 18** (Mac Lane's Strictness Theorem)**.** *For any monoidal category $\mathscr{C}$ there is a monoidally equivalent strict category.*

Proof. $\mathcal{S}$ and $\mathcal{N}$ are monoidal functors by Propositions 4 and 11, and they form a monoidal equivalence with η and ϵ by Propositions 13 and 17. Since $\mathscr{C}$ was arbitrary, the proof is complete. ◀

Note that in contrast to Mac Lane's proof of Theorem 18, we make no reference to the coherence theorem. We can therefore make use of the strictness theorem to prove coherence, which is the subject Section 4.

4 Coherence

We can now give an elementary proof of Mac Lane's *coherence theorem*. In [18], Mac Lane gives his theorem in two parts: Theorem 1 [18, p. 166] and its corollary [18, p. 169]. The "meat" of the proof is in the former part, corresponding to our Section 4.1. We do not give a proof of Mac Lane's corollary here, but the interested reader may find a graphical exposition in the extended version of this paper [23].

Mac Lane begins by defining a certain preorder $\mathscr{W}$, which he then shows enjoys the following property:

▶ **Theorem 19** (Mac Lane's Coherence Theorem [18, p. 166])**.** *Let $\mathscr{M}$ be an arbitrary monoidal category, and let M be an object of $\mathscr{M}$. Then there is a unique strict monoidal functor $\mathscr{W} \to \mathscr{M}$ such that $W \mapsto M$.*

In contrast, we will define $\mathscr{W}$ so this unique functor is easy to construct, and then use $\overline{\mathscr{W}}$ to give a *graphical proof* that $\mathscr{W}$ is a preorder. Note that the monoidal functor in question is *strict*, so its coherence maps are identities.

4.1 The free monoidal category on one generator

We begin by defining $\mathscr{W}$. Again, recall that our definition differs from Mac Lane; we will later show that this definition indeed yields a preorder in order to guarantee that we indeed prove the same theorem.

▶ **Definition 20.** *We define $\mathscr{W}$ as the monoidal category freely generated by a single object W and no morphisms except those required by the definition of a monoidal category.* [1]

[1] Mac Lane denotes the generating object as $(-)$ to suggest an "empty place". We follow Peter Hines' convention [14] and use W instead.

▶ Remark 21. The objects of $\mathscr{W}$ are $I_{\mathscr{W}}$, W, and their tensor products. The arrows are $\mathsf{id}, \rho, \lambda, \alpha$ and their composites and tensor products.

It is now clear that the statement of Mac Lane's Theorem 1 holds for our definition of $\mathscr{W}$:

▶ **Proposition 22.** *Given an arbitrary monoidal category $\mathscr{M}$ and object $M \in \mathscr{M}$, there is a unique strict monoidal functor $\mathscr{W} \to \mathscr{M}$ with $W \mapsto M$.*

Proof. Suppose $\mathsf{U} : \mathscr{W} \to \mathscr{M}$ is such a (strict) monoidal functor. Then we must have that $\mathsf{U}(W) = M$ by assumption, and

$$\mathsf{U}(I) = I \quad \mathsf{U}(A \otimes M) = \mathsf{U}(A) \otimes \mathsf{U}(M) \quad \mathsf{U}(f) = f, f \in \{\alpha, \lambda, \rho, \mathsf{id}\} \quad \mathsf{U}(f \otimes g) = \mathsf{U}(f) \otimes \mathsf{U}(g)$$

because U is strict. But this accounts for all objects and morphisms of $\mathscr{W}$, and so U must be unique. ◀

However, to constitute a proof of the coherence theorem we must now *prove* that $\mathscr{W}$ is a preorder. Our argument proceeds in three main steps. We will show the following:

1. For any monoidal category $\mathscr{C}$, If $\overline{\mathscr{C}}$ is a preorder, then so is $\mathscr{C}$
2. $\overline{\mathscr{W}}$ is generated solely by adapters $\{\Phi, \phi\}$ and their inverses.
3. $\overline{\mathscr{W}}$ is a preorder (which we prove graphically)

The first two steps are straightforward; we address them now. The third requires more work, and is contained in Section 4.2.

▶ **Proposition 23.** *If $\overline{\mathscr{C}}$ is a preorder, then so is $\mathscr{C}$.*

Proof. Let $f, g : \mathscr{C}(A, B)$. Recall that $\mathcal{N} \circ \mathcal{S} = \mathsf{id}$, and so we can derive $f = \mathcal{N}(\mathcal{S}(f)) = \mathcal{N}(\mathcal{S}(g)) = g$ where we used that $\mathcal{S}(f) = \mathcal{S}(g)$ because $\overline{\mathscr{C}}$ is a preorder. ◀

Another lemma shows we can reason about $\overline{\mathscr{W}}$ by considering only adapters:

▶ **Proposition 24.** *$\overline{\mathscr{W}}$ is generated by Φ, ϕ and their inverses.*

Proof. Arrows of $\overline{\mathscr{W}}$ are by definition either adapters Φ, ϕ, their inverses, or morphisms $\overline{f}$ for some $f \in \mathscr{W}$. But note that all such $f \in \mathscr{W}$ are either $\mathsf{id}, \rho, \lambda, \alpha$ or their composites. It is clear that each of λ, ρ, α can each be written as adapters by equations (4), so it remains to show that composites of such morphisms can also be written this way.

That is, we must show that $\mathcal{S}(f \otimes g)$ can be expressed using only adapters and their composites. This can be proved inductively: if $\mathcal{S}(f), \mathcal{S}(g)$ can be expressed using adapters, then so too can compositions $\mathcal{S}(f \fatsemi g) = \mathcal{S}(f) \fatsemi \mathcal{S}(g)$ and tensors $\mathcal{S}(f \otimes g) = \Phi \fatsemi (\mathcal{S}(f) \bullet \mathcal{S}(g)) \fatsemi \Phi^*$.

Thus every morphism of $\overline{\mathscr{W}}$ can be expressed in terms of adapters, and so the category can be said to be *generated* by (only) adapters. ◀

4.2 Graphical proof that $\overline{\mathscr{W}}$ is a preorder

We can now prove graphically that $\overline{\mathscr{W}}$ is a preorder using a normal form argument. Our approach is as follows:

1. Define for each object a `size` in $\mathbb{N}$ (Definition 25)
2. Prove all morphisms in $\overline{\mathscr{W}}$ go between objects of the same size (Proposition 26)
3. Define a canonical arrow $\mathsf{can}(A, B)$ between any two objects of the same size (Definition 31)
4. Show that any arrow is equal to the canonical one (Proposition 33)

Note that we make heavy use of Proposition 24, which lets us reason about $\overline{\mathscr{W}}$ inductively in terms of adapters and their tensors and composites.

We begin–following Mac Lane–by defining the *size* of an object (the same as Mac Lane's notion of *length* [18, p. 165]) as follows:

▶ **Definition 25.** *We define the* `size` *of an object as the number of occurrences of* W*, defined inductively:*

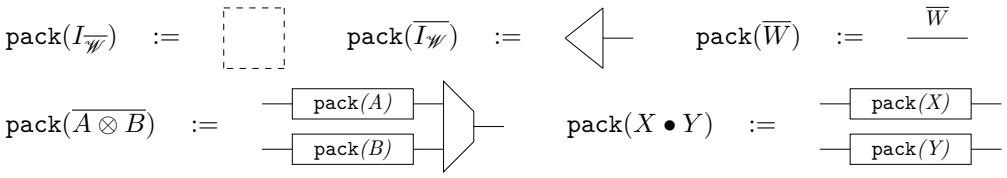
$$\texttt{size}(I_{\overline{\mathscr{W}}}) := 0 \qquad \texttt{size}(\overline{I_{\mathscr{W}}}) := 0 \qquad \texttt{size}(\overline{W}) := 1$$
$$\texttt{size}(\overline{A \otimes B}) := \texttt{size}(A) + \texttt{size}(B) \qquad \texttt{size}(X \bullet Y) := \texttt{size}(X) + \texttt{size}(Y)$$

▶ **Proposition 26.** $\overline{\mathscr{W}}$ *morphisms preserve* `size`*:*
If $f : A \to B$ *is a morphism in* $\overline{\mathscr{W}}$*, then* $\texttt{size}(A) = \texttt{size}(B)$*.*

Proof. Induction on morphisms. ◀

We will define the canonical arrow $\texttt{can}(A, B)$ in two halves, `pack` and `unpack`. To do so, we will first need some additional definitions.

▶ **Definition 27.** *We define the "packing" and "unpacking" morphisms* `pack` *and* `unpack` *in terms of objects of* $\overline{\mathscr{W}}$*. Let* $A \in \overline{\mathscr{W}}$ *be an object. Then* $\texttt{pack}(A)$ *is the morphism defined inductively as follows:*



And define $\texttt{unpack}(A)$ *as* $\texttt{pack}(A)^{-1}$*.*

▶ Remark 28. It can be more intuitive to define `unpack` first, thinking of it as the adapter which removes extraneous $I_{\mathscr{C}}$ objects and "normalises" the object into a flat array of $\overline{W}$ objects. In this view, `pack` is the adapter morphism taking a fixed number of $\overline{W}$ objects and assembling them into a certain bracketing, with unit objects inserted as appropriate.

In Definition 27 we implicitly used that $\overline{\mathscr{W}}$ is a groupoid to define `unpack`, which we now prove:

▶ **Proposition 29.** $\overline{\mathscr{W}}$ *is a groupoid.*

Proof. Generators and identities have inverses by Definition 1, which allows an inductive definition for tensor and composition, i.e. $(f \fatsemi g)^{-1} = g^{-1}f^{-1}$ and $(f \bullet g)^{-1} = f^{-1} \bullet g^{-1}$ respectively. ◀

Now, in order to define the canonical arrow as a composition of `pack` and `unpack`, we will need the following lemma which states that for objects of the same size, we can compose their `unpack` and `pack` morphisms.

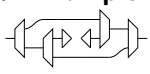
▶ **Proposition 30.** $\texttt{pack}(A) : \overline{W}^{\texttt{size}(A)} \to A$*.*
In other words, for an object A *of size* n*, the domain of* $\texttt{pack}(A)$ *is the* n*-fold* $\bullet$*-tensoring of* $\overline{W}$*.*

Proof. Simple induction on objects (the domain of each $\mathtt{pack}(A)$ is either $I_{\overline{\mathscr{W}}}$, $\overline{W}^k$ or a tensoring of terms) ◀

▶ **Definition 31.** *To each pair of objects A, B of the same size, we can define a canonical arrow as follows:*

$$\mathsf{can}(A, B) := \mathtt{unpack}(A) \fatsemi \mathtt{pack}(B)$$

Note that the composition of Definition 31 is well-typed because $\mathtt{size}(A) = \mathtt{size}(B)$ by Proposition 26: $\mathsf{cod}(\mathtt{unpack}(A)) = \overline{W}^{\mathtt{size}(A)} = \overline{W}^{\mathtt{size}(B)} = \mathsf{dom}(\mathtt{pack}(B))$

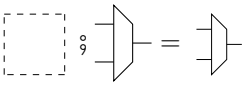
▶ **Example 32.** The canonical arrow between $W \otimes (I_{\mathscr{W}} \otimes W)$ and $(W \otimes I_{\mathscr{W}}) \otimes W$ is  . Note that this is equal to the associator $\alpha_{W,I_{\mathscr{W}},W}$.

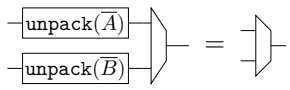
We can now show that every morphism $f : A \to B$ in $\overline{\mathscr{W}}$ is equal to $\mathsf{can}(A, B)$.

▶ **Proposition 33.** *$f = \mathtt{unpack}(A) \fatsemi \mathtt{pack}(B)$ for all $f : A \to B$ in $\overline{\mathscr{W}}$.*

Proof. By induction. On the base case–generators–the proof is straightforward; we give it for identities and generators  and  , with the proofs for inverse generators following by a symmetric argument.

$$\mathsf{can}(X, X) = \mathtt{unpack}(X) \fatsemi \mathtt{pack}(X) = \mathtt{pack}(X)^{-1} \fatsemi \mathtt{pack}(X) = \mathsf{id}_X$$

$$\mathsf{can}(I_{\overline{\mathscr{W}}}, \overline{I_{\mathscr{W}}}) = \mathtt{unpack}(I_{\overline{\mathscr{W}}}) \fatsemi \mathtt{pack}(\overline{I_{\mathscr{W}}}) =$$ 

$$\mathsf{can}(\overline{A} \bullet \overline{B}, \overline{A \otimes B}) = \mathtt{unpack}(\overline{A} \bullet \overline{B}) \fatsemi \mathtt{pack}(\overline{A \otimes B}) =$$ 

The composition of canonical morphisms is canonical:

$$\begin{aligned}
\mathsf{can}(X, Y) \fatsemi \mathsf{can}(Y, Z) &= \mathtt{unpack}(X) \fatsemi \mathtt{pack}(Y) \fatsemi \mathtt{unpack}(Y) \fatsemi \mathtt{pack}(Z) \\
&= \mathtt{unpack}(X) \fatsemi \mathtt{pack}(Y) \fatsemi \mathtt{pack}(Y)^{-1} \fatsemi \mathtt{pack}(Z) \\
&= \mathtt{unpack}(X) \fatsemi \mathtt{pack}(Z) \\
&= \mathsf{can}(X, Z)
\end{aligned}$$

And so is the tensor product:

$$\mathsf{can}(X_1, Y_1) \bullet \mathsf{can}(X_2, Y_2) =$$

$$= \mathsf{can}(X_1 \bullet X_2, Y_1 \bullet Y_2)$$ ◀

▶ **Proposition 34.** *$\overline{\mathscr{W}}$ is a preorder.*

Proof. By Proposition 26 we know that all morphisms $f : A \to B$ have the property that $\mathtt{size}(A) = \mathtt{size}(B)$. We then define for any such objects a canonical morphism $\mathsf{can}(A, B)$ in Definition 31. This canonical isomorphism is unique by Definition 33, and so $\overline{\mathscr{W}}$ is a preorder. ◀

Since we have now proven that $\overline{\mathscr{W}}$ is a preorder, it is now straightforward to prove Theorem 19. Note that this is essentially the opposite of the approach taken by Mac Lane, who *defines* a preorder, and then shows the existence of a unique strict monoidal functor.

Proof of Theorem 19. By Proposition 22 there is a unique, strict monoidal functor from $\mathscr{W}$ to an arbitrary monoidal category $\mathscr{M}$ with $W \mapsto A$ for some $A \in \mathscr{M}$. Moreover, $\overline{\mathscr{W}}$ is a preorder, and so by Proposition 23, so is $\mathscr{W}$. ◀

A first consequence of the coherence theorem is that $\mathcal{N}$ is a *strict* inverse to $\mathcal{S}$ for morphisms $f : \overline{A} \to \overline{B}$ in $\overline{\mathscr{W}}$.

▶ **Proposition 35.** *If $f : \overline{A} \to \overline{B}$ then $\mathcal{S}(\mathcal{N}(f)) = f$.*

Proof. We know that for any $A \in \mathscr{W}$ that $\mathcal{N}(\overline{A}) = A$. Thus for $f : \overline{A} \to \overline{B}$ we have $\mathcal{N}(f) : A \to B$ and thus $\mathcal{S}(\mathcal{N}(f)) : \overline{A} \to \overline{B}$. But $\overline{\mathscr{W}}$ is a preorder, so we have $\mathcal{S}(\mathcal{N}(f)) = f$. ◀

Proposition 35 guarantees that any morphism of this type formed from adapters genuinely represents a specific morphism in $\overline{\mathscr{C}}$ built from associators and unitors; we can use this fact in to restate the coherence theorem in terms of adapter morphisms. Mac Lane's corollary then follows straightforwardly: the basic idea is to "export" commuting diagrams from $\mathscr{W}$ to an arbitrary monoidal category by interpreting the objects of $\mathscr{W}$ as functors, and arrows as natural transformations. We provide a graphical exposition of this proof in the extended version of our paper [23].

5 Conclusions

The body of work on string diagrams in general is broad and growing rapidly. It is therefore slightly surprising that the fundamental issue of non-strict tensorial composition has been neglected for so long. On the one hand, this is reasonable. The assumption of strictness does not entail a loss of generality, as indeed we have confirmed via an example in Sec. 2.3. However, non-strict tupling is a basic feature of programming languages, and even hardware description languages, and modelling it using string diagrams requires the proper mathematical framework.

This framework, the main contribution of the paper given in Def. 1, shows a way to strictify a possibly non-strict monoidal category. The body of the paper proves that the definition has all the desired properties and, in the process, we discuss two new proofs for Mac Lane's strictness and coherence theorems, respectively. We believe that, as is usually the case, the string-diagrammatic perspective has pedagogical value, lending concrete intuitions to what otherwise seems like an abstract symbolic exercise.

5.1 Further work

Lack of support for non-strict tensor limits the range of many applications of string diagrams. The first immediate question to study is whether the strictification recipe we give is compatible with hierarchical string diagrams ("functorial boxes" [19]) which can be used in the representation of monoidal-closed and cartesian-closed categories. This, in turn, makes them useful for applications to programming languages with higher-order functions, such as high-level circuit synthesis [11] or automatic differentiation [1], which currently do not offer support for product. Similar considerations motivate the study of strictification of trace monoidal categories, which can be used as models of digital circuits [12].

Further, our construction expands the use of datastructures and algorithms currently limited only to the strict case (e.g., [24]). Such datastructures are typically based on graph or hypergraph representations for performance reasons; applying our construction allows us to leverage those benefits essentially for free. In cases where such datastructures and algorithms are proven correct, it may be beneficial to reproduce the proofs in this paper in a formal theorem prover in order to provide end-to-end verification of applications.

Finally, a formal understanding of non-strict monoidal categories may open the door to more graphical approaches for theorem proving. Interactive graphical theorem provers using string diagrams for strict monoidal categories such as `homotopy.io` represent a refreshingly new approach to the design of proof assistants. Since models of, for example, intuitionistic logic are non-strict, the novel string diagrams in this paper could be used perhaps to develop similar graphical proof assistant for more conventional logics.

References

1 Mario Alvarez-Picallo, Dan R. Ghica, David Sprunger, and Fabio Zanasi. Functorial string diagrams for reverse-mode automatic differentiation. *CoRR*, abs/2107.13433, 2021. `arXiv:2107.13433`.

2 John C. Baez and Brendan Fong. A compositional framework for passive linear networks. *Theory and Applications of Categories*, 33(38):1158–1222, 2018. `doi:10.48550/arXiv.1504.05625`.

3 R.F. Blute, J.R.B. Cockett, R.A.G. Seely, and T.H. Trimble. Natural deduction and coherence for weakly distributive categories. *Journal of Pure and Applied Algebra*, 113(3):229–296, 1996. `doi:10.1016/0022-4049(95)00159-X`.

4 Filippo Bonchi, Robin Piedeleu, Pawel Sobocinski, and Fabio Zanasi. Graphical affine algebra. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, June 2019. `doi:10.1109/lics.2019.8785877`.

5 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Full abstraction for signal flow graphs. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 515–526. ACM, 2015. `doi:10.1145/2676726.2676993`.

6 Titouan Carette, Yohann D'Anello, and Simon Perdrix. Quantum algorithms and oracles with the scalable ZX-calculus. *Electronic Proceedings in Theoretical Computer Science*, 343:193–209, September 2021. `doi:10.4204/eptcs.343.10`.

7 Vikraman Choudhury, Jacek Karwowski, and Amr Sabry. Symmetries in reversible programming: From symmetric rig groupoids to reversible programming languages, 2021. `doi:10.48550/arXiv.2110.05404`.

8 Bob Coecke, Edward Grefenstette, and Mehrnoosh Sadrzadeh. Lambek vs. lambek: Functorial vector space semantics and string diagrams for lambek calculus. *Ann. Pure Appl. Log.*, 164(11):1079–1100, 2013. `doi:10.1016/j.apal.2013.05.009`.

9 Bob Coecke and Aleks Kissinger. *Picturing quantum processes: A first course in quantum theory and diagrammatic reasoning*. Cambridge University Press, 2017.

10 P. I. Etingof, Shlomo Gelaki, Dmitri Nikshych, and Victor Ostrik, editors. *Tensor categories*. Number volume 205 in Mathematical surveys and monographs. American Mathematical Society, 2015. URL: `https://klein.mit.edu/~etingof/egnobookfinal.pdf`.

11 Dan R. Ghica. Geometry of synthesis: a structured approach to VLSI design. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 363–375. ACM, 2007. `doi:10.1145/1190216.1190269`.

12 Dan R. Ghica, Achim Jung, and Aliaume Lopez. Diagrammatic semantics for digital circuits. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.CSL.2017.24`.

13 Dan R. Ghica, George Kaye, and David Sprunger. Full abstraction for digital circuits, 2022. `doi:10.48550/arXiv.2201.10456`.

14 Peter Hines. Coherence and strictification for self-similarity, 2015. `arXiv:1304.5954`.

15 A. Joyal and R. Street. Braided tensor categories. *Advances in Mathematics*, 102(1):20–78, 1993. `doi:10.1006/aima.1993.1055`.

16 André Joyal and Ross Street. The geometry of tensor calculus, i. *Advances in Mathematics*, 88(1):55–112, 1991. doi:10.1016/0001-8708(91)90003-P.

17 Yves Lafont. Towards an algebraic theory of Boolean circuits. *Journal of Pure and Applied Algebra*, 184(2-3):257–310, November 2003. doi:10.1016/S0022-4049(03)00069-0.

18 Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1997. doi:10.1007/978-1-4757-4721-8.

19 Paul-André Melliès. Functorial boxes in string diagrams. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2006. doi:10.1007/11874683_1.

20 Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010. arXiv:0908.3347.

21 Michael Shulman. A practical type theory for symmetric monoidal categories, 2019. doi:10.48550/arXiv.1911.00818.

22 Stuart Sutherland, Simon Davidmann, and Peter Flake. *SystemVerilog for Design Second Edition: A Guide to Using SystemVerilog for Hardware Design and Modeling*. Springer Science & Business Media, 2006.

23 Paul Wilson, Dan Ghica, and Fabio Zanasi. String diagrams for non-strict monoidal categories, 2022. doi:10.48550/arXiv.2201.11738.

24 Paul Wilson and Fabio Zanasi. The cost of compositionality: A high-performance implementation of string diagram composition, 2021. arXiv:2105.09257.

25 Paul Wilson and Fabio Zanasi. Categories of differentiable polynomial circuits for machine learning, 2022. doi:10.48550/arXiv.2203.06430.

A Sequential Normal Form

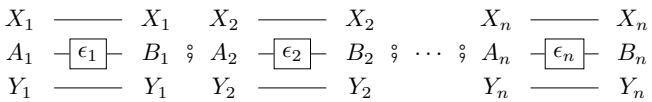
The following proposition is well-known (see for example [17]) and straightforward to prove, but we provide a proof anyway for completeness.

▶ **Proposition 36.** *let $\mathscr{C}$ be a monoidal category presented by generators Σ and some equations. Then any (finite) term t representing a morphism of $\mathscr{C}$ can be factored into "slices":*

$$(\mathsf{id} \otimes \epsilon_1 \otimes \mathsf{id}) \fatsemi (\mathsf{id} \otimes \epsilon_2 \otimes \mathsf{id}) \fatsemi \ldots \fatsemi (\mathsf{id} \otimes \epsilon_n \otimes \mathsf{id})$$

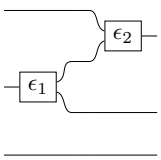
where each ϵ_i is a generator.

This factorization can be diagrammed as follows:



Note that in general $X_i \neq X_{i+1}$ and so on- i.e., the generators need not be "aligned" in this factorization. For example, we can have morphisms like the following:

▶ **Example 37.**



Proof. We proceed by induction on terms. Let S_0 denote the set of terms consisting of identities and generators, Then let

$$S_n = S_0 \cup \{t \fatsemi u | t, u \in S_{n-1}\} \cup \{t \otimes u | t, u \in S_{n-1}\}$$

It is clear that terms in S_0 are already in sequential normal form, so it remains to prove the inductive case, beginning with composition. Let v be a term in S_{n+1}. Now by inductive hypothesis, any term in $w \in S_n$ has an equivalent term in sequential normal form, which we'll denote $\hat{w}$. Now there are three cases:

1. If $v \in S_n$, then we have $\hat{v}$ by inductive hypothesis.
2. If $v = t \fatsemi u$, then $\hat{t}$ and $\hat{u}$ exist by inductive hypothesis, and we can form $\hat{v} = \hat{t} \fatsemi \hat{u}$.
3. If $v = t \otimes u$, then $\hat{v} = (\hat{t} \otimes \mathsf{id}) \fatsemi (\mathsf{id} \otimes \hat{u})$

and the proof is complete. ◀

B $\mathcal{N}$ is well-defined

In this appendix we verify that $\mathcal{N}$ is well-defined. This amounts to two things: first that $\mathcal{N}$ is well-defined with respect to the interchange law, and second that it respects the equations of Definition 1.

In the former case, sequential normal forms are only unique up to interchange, so it must be verified that $\mathcal{N}$ preserves this property. This is a straightforward if tedious exercise, which can be done by verifying each of the cases in Definition 6. Essentially, the only axioms required are naturality and equations [10, 2.12, 2.13].

Finally, we need to verify the equations of 1. Specifically, for each of the monoidal, adapter, and associator/unitor equations $\texttt{lhs} = \texttt{rhs}$, we show that $\mathcal{N}(\texttt{lhs}) = \mathcal{N}(\texttt{rhs})$. We give derivations for these below. Using these derivations one can also tediously check that the equations hold for cases $\mathsf{id} \bullet \texttt{lhs} \bullet \mathsf{id} = \mathsf{id} \bullet \texttt{rhs} \bullet \mathsf{id}$, and so $\mathcal{N}$ is equal under any rewrite involving those equations; the only cases of interest are for associator and unitor equations, which require the use of the pentagon and triangle axioms, respectively.

We begin with the functor equations (2)

$$\mathcal{N}(\overline{\mathsf{id}_A}) = \mathsf{id}_A = \mathcal{N}(\mathsf{id}_{\overline{A}}) \qquad \mathcal{N}(\overline{f} \fatsemi \overline{g}) = \mathcal{N}(\overline{f}) \fatsemi \mathcal{N}(\overline{g}) = f \fatsemi g = \mathcal{N}(\overline{f \fatsemi g})$$

Now the adapter equations (3):

$$\begin{aligned}
\mathcal{N}(\Phi \fatsemi (\overline{f} \bullet \overline{g}) \fatsemi \Phi^*) &= \mathcal{N}(\Phi) \fatsemi \mathcal{N}(\overline{f} \bullet \overline{g}) \fatsemi \mathcal{N}(\Phi^*) \\
&= \mathcal{N}(\overline{f} \bullet \overline{g}) \\
&= \mathcal{N}(\overline{f} \bullet \mathsf{id}) \fatsemi \mathcal{N}(\mathsf{id} \bullet \overline{g}) \\
&= (f \otimes \mathsf{id}) \fatsemi (\mathsf{id} \otimes g) \\
&= f \otimes g \\
&= \mathcal{N}(\overline{f \otimes g})
\end{aligned}$$

$$\begin{aligned}
\mathcal{N}(\phi \fatsemi \phi^*) &= \mathcal{N}(\phi) \fatsemi \mathcal{N}(\phi^*) \\
&= \mathsf{id}_{I_{\mathscr{C}}} \fatsemi \mathsf{id}_{I_{\mathscr{C}}} \\
&= \mathsf{id}_{I_{\mathscr{C}}} \\
&= \mathcal{N}(\mathsf{id}_{I_{\overline{\mathscr{C}}}})
\end{aligned}$$

$$\begin{aligned}
\mathcal{N}(\Phi^* \fatsemi \overline{f \otimes g} \fatsemi \Phi^*) &= \mathcal{N}(\Phi^*) \fatsemi \mathcal{N}(\overline{f \otimes g}) \fatsemi \mathcal{N}(\Phi) \\
&= \mathcal{N}(\overline{f \otimes g}) \\
&= f \otimes g \\
&= (f \otimes \mathsf{id}) \fatsemi (\mathsf{id} \otimes g) \\
&= \mathcal{N}(\overline{f} \bullet \mathsf{id}) \fatsemi \mathcal{N}(\mathsf{id} \bullet \overline{g}) \\
&= \mathcal{N}((\overline{f} \bullet \mathsf{id}) \fatsemi (\mathsf{id} \bullet \overline{g})) \\
&= \mathcal{N}(\overline{f} \bullet \overline{g})
\end{aligned}$$

$$\begin{aligned}
\mathcal{N}(\phi^* \fatsemi \phi) &= \mathcal{N}(\phi^*) \fatsemi \mathcal{N}(\phi) \\
&= \mathsf{id}_{I_{\mathscr{C}}} \fatsemi \mathsf{id}_{I_{\mathscr{C}}} \\
&= \mathsf{id}_{I_{\mathscr{C}}} \\
&= \mathcal{N}(\overline{\mathsf{id}_{I_{\mathscr{C}}}}) \\
&= \mathcal{N}(\mathsf{id}_{\overline{I_{\mathscr{C}}}})
\end{aligned}$$

Finally the associator/unitor equations (4):

$$\begin{aligned}\mathcal{N}(\Phi^* \fatsemi (\mathsf{id} \bullet \Phi^*) \fatsemi (\Phi \bullet \mathsf{id}) \fatsemi \Phi) &= \mathcal{N}(\Phi^*) \fatsemi \mathcal{N}(\mathsf{id} \bullet \Phi^*) \fatsemi \mathcal{N}(\Phi \bullet \mathsf{id}) \fatsemi \mathcal{N}(\Phi) \\ &= \mathsf{id} \fatsemi \mathsf{id} \fatsemi \alpha \fatsemi \mathsf{id} \\ &= \alpha \\ &= \mathcal{N}(\overline{\alpha})\end{aligned}$$

$$\begin{aligned}\mathcal{N}(\Phi^* \fatsemi (\Phi^* \bullet \mathsf{id}) \fatsemi (\mathsf{id} \bullet \Phi) \fatsemi \Phi) &= \mathcal{N}(\Phi^*) \fatsemi \mathcal{N}(\Phi^* \bullet \mathsf{id}) \fatsemi \mathcal{N}(\mathsf{id} \bullet \Phi) \fatsemi \mathcal{N}(\Phi) \\ &= \mathsf{id} \fatsemi \alpha^{-1} \fatsemi \mathsf{id} \fatsemi \mathsf{id} \\ &= \alpha^{-1} \\ &= \mathcal{N}(\overline{\alpha^{-1}})\end{aligned}$$

$$\begin{aligned}\mathcal{N}(\Phi^* \fatsemi (\phi^* \bullet \mathsf{id})) &= \mathcal{N}(\Phi^*) \fatsemi \mathcal{N}(\phi^* \bullet \mathsf{id}) \\ &= \mathsf{id} \fatsemi \lambda \\ &= \lambda \\ &= \mathcal{N}(\overline{\lambda})\end{aligned}$$

$$\begin{aligned}\mathcal{N}((\phi \bullet \mathsf{id}) \fatsemi \Phi) &= \mathcal{N}(\phi \bullet \mathsf{id}) \fatsemi \mathcal{N}(\Phi) \\ &= \lambda^{-1} \fatsemi \mathsf{id} \\ &= \lambda^{-1} \\ &= \mathcal{N}(\overline{\lambda^{-1}})\end{aligned}$$

$$\begin{aligned}\mathcal{N}(\Phi^* \fatsemi (\mathsf{id} \bullet \phi^*)) &= \mathcal{N}(\Phi^*) \fatsemi \mathcal{N}(\mathsf{id} \bullet \phi^*) \\ &= \mathsf{id} \fatsemi \rho \\ &= \rho \\ &= \mathcal{N}(\overline{\rho})\end{aligned}$$

$$\begin{aligned}\mathcal{N}((\mathsf{id} \fatsemi \phi) \fatsemi \Phi) &= \mathcal{N}(\mathsf{id} \fatsemi \phi) \fatsemi \mathcal{N}(\Phi) \\ &= \rho^{-1} \fatsemi \mathsf{id} \\ &= \rho^{-1} \\ &= \mathcal{N}(\overline{\rho^{-1}})\end{aligned}$$

Thus $\mathcal{N}$ is well-defined with respect to the monoidal equations.

Supported Sets – A New Foundation for Nominal Sets and Automata

Thorsten Wißmann
Radboud University, Nijmegen, the Netherlands

Abstract

The present work proposes and discusses the category of supported sets which provides a uniform foundation for nominal sets of various kinds, such as those for equality symmetry, for the order symmetry, and renaming sets. We show that all these differently flavoured categories of nominal sets are monadic over supported sets. Thus, supported sets provide a canonical finite way to represent nominal sets and the automata therein, e.g. register automata and coalgebras in general. Name binding in supported sets is modelled by a functor following the idea of de Bruijn indices. This functor lifts to the well-known abstraction functor in nominal sets. Together with the monadicity result, this gives rise to a transformation process from finite coalgebras in supported sets to orbit-finite coalgebras in nominal sets. One instance of this process transforms the finite representation of a register automaton in supported sets into its configuration automaton in nominal sets.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Nominal Sets, Monads, LFP-Category, Supported Sets, Coalgebra

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.38

Related Version *Full Version*: `https://arxiv.org/abs/2201.09825`

Funding *Thorsten Wißmann*: Supported by the NWO TOP project 612.001.852.

Acknowledgements The author thanks Jurriaan Rot, Joshua Moerman, and Frits Vaandrager for inspiring discussions. The definition of supported sets originated from discussions with Lutz Schröder, Dexter Kozen, and Stefan Milius.

1 Introduction

Nominal sets provide an elegant framework to reason about structures that involve names, the permutation of names, and name binding. Originally used 100 years ago by Fraenkel and Mostowski for the entirely different purpose of axiomatic set theory, Gabbay and Pitts [19] re-discovered them in 1999 to define λ-expressions modulo α-equivalence as an inductive data type. Its associated structural recursion principle allowed it to define capture avoiding substitution as a total function. Since then, a plethora of results using nominal sets were established in many areas, including proof assistants [42, 5], calculi [18, 41], and automata models [7, 39, 29]. Nominal automata are capable of processing words over infinite alphabets (*data alphabets*), while having good computational properties. Their expressiveness is similar (and sometimes identical) to that of register automata [27, 13, 8], which are automata with a *finite* description processing infinite alphabets. This finiteness condition translates into the notion of *orbit-finiteness* in the nominal world, which requires extra work to obtain a finite description of nominal automata, because orbit-finite objects are infinite in general.

In recent years, more general concepts of nominal sets were considered that generalize from the permutation of names to other operations. One branch is called *renaming sets* [41, 21, 34] and allows that distinct names are mapped to the same identifier; in another branch, symmetries on other data alphabets [7, 46] were considered, for example monotone bijections on rational numbers $\mathbb{Q}$, called the *total order symmetry*.

Nominal sets are not the only categorical approach to name binding. Multiple presheaf-based approaches to name binding were developed over the years [16, 17] which are strongly related to nominal sets [23]. Also, the theory of *named sets* were introduced for the study of history dependent automata [15, 36], they were shown to be equivalent (in the categorical sense) to nominal sets [23], and they also feature a name binding construction [9]. In contrast to nominal sets, the name binding in presheaf approaches and named sets follow the method of *de Bruijn indices* [11].

Despite their rich categorical structure, it is known that the category of nominal sets (in all above-mentioned flavours) is not *monadic* over sets, that is, their theory is not an algebraic theory that can be described by a monad on sets – unlike groups, rings, vector spaces, etc, which can be described by algebraic theories.

In order to still make algebraic methods applicable, we introduce the category of *supported sets* and show that nominal sets are monadic over those. Thus, we make nominal sets applicable to results and methods known from algebraic categories, such as the generalized powerset construction [40, 6] or results about the representation of algebras [1] that provide a canonical finite representation of orbit-finite nominal sets. Moreover, supported sets do not require symmetries on the data alphabet, so they can also serve as a categorical foundation for data alphabets that are not described by symmetries. In fact, we discovered supported sets while working on a learning algorithm for register automata for general data alphabets.

Structure of the paper. After recalling the basic definitions for nominal sets (Section 3), we introduce supported sets and discuss their basic categorical properties (Section 4). Having established the monadicity of nominal sets (Section 5), we show that supported sets have a functor for name binding that lifts to nominal sets (Section 6). This yields a coalgebraic construction from (register) automata in supported sets to nominal automata (Section 7). Full proofs and also additional explanations of definitions and examples can be found in the appendix of the full version: `https://arxiv.org/abs/2201.09825`.

2 A Quick Overview of Supported Sets

Let us first take a more higher-level look at the properties of supported sets. The proposed category has such a simple definition that we can state it already here in full detail:

▶ **Definition.** *For a fixed set A, the category* $\mathsf{Supp}(A)$ *contains the following data:*

- *a supported set* (X, s_X) *is a set* X *and a map* $\mathsf{s}_X \colon X \to \mathcal{P}_\mathsf{f}(A)$*, called the* support *(where* $\mathcal{P}_\mathsf{f}$ *denotes finite powerset),*
- *a supported map* $f \colon (X, \mathsf{s}_X) \to (Y, \mathsf{s}_Y)$ *is a map* $f \colon X \to Y$ *with the property that* $\mathsf{s}_Y(f(x)) \subseteq \mathsf{s}_X(x)$ *for all* $x \in X$*.*

This definition reflects the basic principles when working with data alphabets A: the terms or structures $x \in X$ of interest store finitely many data values $\mathsf{s}_X(x) \subseteq A$. The computations f on such a structure x can not invent new data values but may decide to drop data values, e.g. by clearing a register: $\mathsf{s}_Y(f(x)) \subseteq \mathsf{s}_X(x)$.

In contrast to nominal sets, the map s_X is a structural property of a supported set X and not a derived notion. Thus, we can define supported sets by simply specifying X and s_X without any notion of permutation or renaming of data values.

$\mathsf{Supp}(A)$ has nice categorical properties: it is complete, cocomplete, and cartesian closed. $\mathsf{Supp}(A)$ is locally finite, that is, a supported set is *finitely presentable* iff it is finite, and so it is easy to represent supported sets for algorithmic purposes.

Despite the little structure of supported sets, *name binding* exists as a functor on $\mathsf{Supp}(A)$ (for countably infinite A) which is reminiscent of de Bruijn indices [11], and thus different to the permutation based *abstraction functor* of nominal sets. Surprisingly, the binding functor of $\mathsf{Supp}(A)$ *lifts* to the abstraction functor in nominal sets (up to natural isomorphism).

Different kinds of *nominal theories* can be modelled as monads on supported sets. If A is a countable infinite set, the following categories are then *monadic* over $\mathsf{Supp}(A)$:

1. **Nominal Sets** (for the equality symmetry) [19, 38].
2. **Nominal renaming sets** in which atoms can be merged together [41, 21, 34].
3. **Ordered nominal sets** in which the atoms are rational numbers and can only be renamed in a monotone way [7, 46].

These monadic adjunctions make general results about monadic adjunctions applicable:

1. Finite representation [1]: a (possibly infinite) nominal set is orbit-finite iff it can be described by a finite supported set of *generators* and finitely many *equations*.
2. Generalized Powerset Construction [40, 6]: a register automaton lives as a finite coalgebra in supported sets. The construction transforms it into an orbit-finite nominal coalgebra, whose state-space is then the configuration space of the original register automaton.

These applications show that supported sets should not be understood as a competitor to nominal sets, but as a common foundation for different nominal symmetries, and it may serve as a categorical framework for data alphabets that do not admit symmetries at all.

3 Preliminaries on Nominal Sets

Before introducing supported sets in detail, we first recall some notations and basic concepts of nominal sets. We assume that the reader is familiar with basic category theory, but we restrict to set-theoretic definitions wherever possible.

▶ **Notation 3.1.** *Given sets X, Y, the set of all maps $X \to Y$ is written Y^X. Injective maps are denoted by $\rightarrowtail$, surjective maps by $\twoheadrightarrow$, and $\cdot$ denotes map composition. We fix sets $1 = \{0\}$, $2 = \{0, 1\}$. The cycle notation $(a_0\ a_1 \cdots a_{n-1})$ for elements of a set A denotes the bijection $A \to A$ sending $a_0 \mapsto a_1$, $a_1 \mapsto a_2$, …, $a_{n-1} \mapsto a_0$, and fixing all other elements of A.*

Nominal sets and various flavours thereof are built around the notion of monoid actions, which specifies how atoms nested in some structure can be permuted or renamed.

▶ **Definition 3.2.** *Given a monoid $(M, \cdot, e)$, an M-set is a set X together with a map "$\cdot$": $M \times X \to X$, called the* action *and written infix $m \cdot x$ for $x \in X$, $m \in M$, such that $e \cdot x = x$ and $(m \cdot m') \cdot x = m \cdot (m' \cdot x)$ for all $m, m' \in M$, $x \in X$. Thus, we use "$\cdot$" both for the monoid multiplication and the action. A map $f\colon X \to Y$ between M-sets $(X, \cdot)$, $(Y, \star)$ is* equivariant *if $f(m \cdot x) = m \star f(x)$ for all $m \in M$, $x \in X$.*

Throughout the paper, M will be a submonoid of $(A^A, \cdot, \mathsf{id}_A)$ for a set A, written $M \leq A^A$, in which most of the results are parametric. The set A is called the set of *atoms*, which can intuitively be understood as names of registers or data-values that appear in the input of an automaton or as names used for binding operations. The submonoid M determines a subset of maps of interest, closed under composition. Thus, we use $\cdot$ both for map composition and monoid-multiplication, and moreover, the unit of the monoid M is simply id_A.

▶ **Example 3.3.** The main instances of monoids M for nominal techniques are as follows:

1. For a set A, let $\mathfrak{S}_{\mathsf{f}}(A)$ be the bijections on A that modify only finitely many elements, i.e.

$$\mathfrak{S}_{\mathsf{f}}(A) := \big\{\phi\colon A \to A \mid \phi \text{ bijective and } \{a \in A \mid \phi(a) \neq a\} \text{ is finite}\big\}$$

   For nominal sets (over the equality symmetry), one fixes a countably infinite set $\mathbb{A}$ (understood as *names*) and fixes $M := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})$ [19, 38].

2. For the set $\mathbb{Q}$ of rational numbers, let $\mathsf{Aut}(\mathbb{Q}, <)$ be the set of bijective and monotone maps $\mathbb{Q} \to \mathbb{Q}$. For nominal sets over the total order symmetry, one considers $M := \mathsf{Aut}(\mathbb{Q}, <)$ [7].
3. Let $\mathsf{Fin}(A) \subseteq A^A$ be the set of maps $f\colon A \to A$ for which $\{f(a) \neq a \mid a \in A\}$ is finite. For *nominal renaming sets*, one considers $A := \mathbb{A}$ and $M := \mathsf{Fin}(\mathbb{A})$ [21].

An element $x \in X$ of an M-set $(X, \cdot)$ can be understood as a structure with elements from A embedded. We can alter these embedded elements according to $m \in M$, yielding $m \cdot x \in X$.

▶ **Example 3.4.** For every set A and $M \leq A^A$, the following sets are M-sets:
1. The set A itself with the action $m \cdot a := m(a)$, for $m \in M$, $a \in A$.
2. If X is an M-set, then so is $\mathcal{P}_\mathsf{f}(X)$, the set of finite subsets of X, where the action is defined point-wise: $m \cdot S := \{m \cdot x \mid x \in S\}$ for $m \in M, S \in \mathcal{P}_\mathsf{f}X$.
3. Every set D can be equipped with the *discrete* action: $m \cdot d = d$ for all $m \in M, d \in D$.
4. The set M itself is an M-set with monoid multiplication $M \times M \to M$ as the action.

The outcome of $m \cdot x$ only depends on what m does on the atoms $S \subseteq A$ buried in x:

▶ **Definition 3.5.** *We say that $m, m' \in M$ are* identical on $S \subseteq A$*, written $m \approx_S m'$, if $m(a) = m'(a)$ for all $a \in S$.*

With the intuition that the action of an M-set X renames the atoms A in an element $x \in X$, we can derive from the action which atoms an element $x \in X$ carries. Then, an M-set is *nominal*, if each $x \in X$ carries only finitely many atoms.

▶ **Definition 3.6** [21, Def. 7]**.** *A set $S \subseteq A$* supports *$x \in X$ of an M-set X, if for all $m \approx_S m'$ (in M), we have $m \cdot x = m' \cdot x$. An M-set X is* nominal *if every element of X has some finite support, i.e. is supported by a finite set $S \subseteq A$.*

If M happens to be a group, then the definition of support can be simplified slightly [19, 38].

▶ **Example 3.7.** Most of the M-sets from Example 3.4 are nominal:
1. A is nominal: every $a \in A$ is supported by $S := \{a\}$, and also by any superset of S.
2. If X is a nominal M-set, then so is $\mathcal{P}_\mathsf{f}(X)$. A set $E \in \mathcal{P}_\mathsf{f}(X)$ of elements is supported by the union of finite supports of the elements $x \in E$. This union is finite because E is so.
3. Every discrete M-set D is nominal because every $x \in D$ is supported by the empty set.
4. For all monoids M of interest for nominal techniques, M considered as an M-set is not nominal: whenever M is a nominal M-set, then *every* M-set is nominal. For example, $\mathfrak{S}_\mathsf{f}(\mathbb{A})$ is not nominal because no $\sigma \in \mathfrak{S}_\mathsf{f}(\mathbb{A})$ has finite support.

▶ **Definition 3.8.** *Let $\mathsf{Nom}(M)$ be the category of nominal M-sets and equivariant maps.*

For our main instances (Example 3.3), we obtain the following categories:
1. The category of *nominal sets* is denoted by $\mathsf{Nom} = \mathsf{Nom}(\mathfrak{S}_\mathsf{f}(\mathbb{A}))$ (slightly overloading notation), that is for $A := \mathbb{A}$ and $M := \mathfrak{S}_\mathsf{f}(\mathbb{A})$. This is called *the equality symmetry.*
2. *Ordered nominal sets* are $\mathsf{OrdNom} = \mathsf{Nom}(\mathsf{Aut}(\mathbb{Q}, <))$, i.e. for $A := \mathbb{Q}$, $M := \mathsf{Aut}(\mathbb{Q}, <)$.
3. *Nominal renaming sets* are $\mathsf{RnNom} = \mathsf{Nom}(\mathsf{Fin}(\mathbb{A}))$, i.e. for $A := \mathbb{A}$, $M := \mathsf{Fin}(\mathbb{A})$.

▶ **Definition 3.9** [7, Definition 4.11]**.** *A monoid $M \leq A^A$* admits least supports *if each element of a nominal M-set has a least finite support. If so, we write $\mathsf{supp}_X\colon X \to \mathcal{P}_\mathsf{f}(A)$ for the map that sends elements of the M-set X to their least finite support.*

In all running examples of nominal set flavours, the monoid admits least supports, i.e. Nom [20, 38], OrdNom [7] and RnNom [21]. Intuitively, $\mathsf{supp}(x) \subseteq A$ can be understood as precisely the atoms that appear in x. For example, $\mathsf{supp}_A\colon A \to \mathcal{P}_\mathsf{f}(A)$ sends a to $\{a\}$, and $\mathsf{supp}_{\mathcal{P}_\mathsf{f}(X)}\colon \mathcal{P}_\mathsf{f}(X) \to \mathcal{P}_\mathsf{f}(A)$ is $\mathsf{supp}_{\mathcal{P}_\mathsf{f}(X)}(E) = \bigcup\{\mathsf{supp}_X(x) \mid x \in E\}$. The opposite of an atom $a \in A$ in the support is:

▶ **Definition 3.10.** *An atom $a \in A$ is* fresh *for $x \in X$ in a nominal set X (notated $a \# x$), if $a \notin \mathsf{supp}_X(x)$. For multiple elements, we write $a \# x, y$ to denote that a is fresh for x and y.*

Both freshness and support are preserved by equivariant maps. Intuitively, equivariant maps can possibly forget about atoms, but can never introduce new atoms:

▶ **Lemma 3.11.** *For an equivariant map, if S supports x, then S also supports $f(x)$. If X and Y have least finite supports, then $\mathsf{supp}_Y(f(x)) \subseteq \mathsf{supp}_X(x)$.*

The set of elements in an M-set that can be reached from an element x is called the orbit:

▶ **Definition 3.12.** *Given a group M, the* orbit *of an element x in an M-set is the subset $\{m \cdot x \mid m \in M\} \subseteq X$. A nominal M-set is* orbit-finite *if it consists of finitely many orbits.*

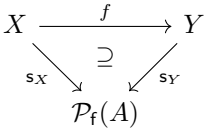
In this definition, we assume M to be a group, because then, every nominal set X is a disjoint union of orbits. For $M := \mathfrak{S}_\mathsf{f}(\mathbb{A})$, the *orbit-finite* nominal sets are precisely the finitely presentable objects in Nom [37, Proposition 2.3.7]. E.g. $\mathbb{A}$ is orbit-finite, $\mathcal{P}_\mathsf{f}\mathbb{A}$ is not. In nominal automata, one requires the state set to be orbit-finite, as opposed to finiteness in classical automata theory.

4 Supported Sets

The central notion of the present paper are supported sets, which are parametric in the set A of atoms or data symbols of interest:

▶ **Definition 4.1.** *For a set A, the category of* supported sets $\mathsf{Supp}(A)$ *contains the following:*

1. *a supported set X is a set X together with a map $\mathsf{s}_X \colon X \to \mathcal{P}_\mathsf{f}(A)$.*
2. *a supported map $f \colon (X, \mathsf{s}_X) \to (Y, \mathsf{s}_Y)$ is a map $f \colon X \to Y$ with $\mathsf{s}_Y(f(x)) \subseteq \mathsf{s}_X(x)$ for all $x \in X$. This means, f makes the following triangle weakly commute.*



Whenever clear from the context, we simply speak of supported sets $X, Y \in \mathsf{Supp}(A)$ and supported maps $f \colon X \to Y$, leaving s_X and s_Y implicit.

The intuition for supported sets comes from the least finite support map supp of nominal sets. Hence, in a supported set X, the map $\mathsf{s}_X(x)$ tells which atoms are carried by $x \in X$, but we can not permute or rename them in general.

The definition of the morphisms in supported sets comes from the property of equivariant maps: they possibly forget about atoms in the support, but they can never introduce new atoms (Lemma 3.11). This observation becomes the defining property of supported maps.

▶ **Example 4.2.** The set A itself is a support set with $\mathsf{s}_A(a) = \{a\}$. However, the singleton subset $\{a\} \subseteq A$ is also a supported set with $\mathsf{s}_{\{a\}}(a) = \{a\}$. In general, every nominal M-set X (for M admitting least supports) yields a supported set by putting $\mathsf{s}_X := \mathsf{supp}_X$.

The difference between nominal and supported sets is that supp in a nominal set X is a derived notion since it is implicit in the M-set action. On the other hand, for supported sets, s_X is part of the syntactical structure. For the sake of clarity, we use different mathematical symbols for the semantical supp and the structural s.

▶ **Lemma 4.3.** *For every $M \leq A^A$, there is a faithful functor $U\colon \mathsf{Nom}(M) \to \mathsf{Supp}(A)$ sending $(X,\cdot)$ to a supported set (X,s_X), where $\mathsf{s}_X(x)$ is the intersection of all finite supports of x in $(X,\cdot)$. If $M \leq A^A$ admits least supports, then $\mathsf{s}_{UX} = \mathsf{supp}_X$. Equivariant maps in $\mathsf{Nom}(M)$ are sent to their underlying map by U.*

Later, we show that this forgetful functor $\mathsf{Nom}(M) \to \mathsf{Supp}(A)$ is right-adjoint and even monadic, so one can consider nominal sets as algebras on supported sets. Before, we establish some categorical properties of $\mathsf{Supp}(A)$ (generic in the choice of A) that will come in handy.

4.1 Universal Constructions and Finiteness

Unsurprisingly, supported sets have a very set-like nature. In the present Section 4.1, we let V denote the forgetful functor $V\colon \mathsf{Supp}(A) \to \mathsf{Set}$ that forgets the support map and sends (X,s_X) to the plain set X. Conversely, every set can be equipped with trivial support:

▶ **Lemma 4.4.** *The inclusion functor $J\colon \mathsf{Set} \hookrightarrow \mathsf{Supp}(A)$ defined by $JX = (X,\mathsf{s}_X)$ with $\mathsf{s}_X(x) = \emptyset$ is right-adjoint to the forgetful $V\colon \mathsf{Supp}(A) \to \mathsf{Set}$ ($V \dashv J$) and $VJ = \mathsf{Id}_{\mathsf{Set}}$.*

In other words, Set is a reflective subcategory of $\mathsf{Supp}(A)$. Similarly to sets, universal constructions such as limits and colimits all exist in supported sets and are (almost) set-like. Given a diagram $D\colon \mathcal{D} \to \mathsf{Set}$, we write $\mathsf{pr}_X\colon \lim D \to DX$ for the limit projection map of $X \in \mathcal{D}$ and $\mathsf{in}_X\colon DX \to \mathsf{colim} D$ for the colimit injections.

▶ **Proposition 4.5.** *All colimits in $\mathsf{Supp}(A)$ exist: Given a diagram $D\colon \mathcal{D} \to \mathsf{Supp}(A)$, the colimit is formed in Set and then equipped with the support $\mathsf{s}\colon \mathsf{colim}\, VD \to \mathcal{P}_{\mathsf{f}}(A)$:*

$$\mathsf{s}(c) = \bigcap \{\mathsf{s}_{DY}(y) \mid Y \in \mathcal{D}, y \in DY, \mathsf{in}_Y(y) = c\}.$$

The intersection handles the case where elements of possibly different support are identified in the colimit. Thus, the intersection vanishes if there are no morphisms in the diagram:

▶ **Example 4.6.** The coproduct of supported sets X, Y is given by their disjoint union, $X + Y$ equipped with support $\mathsf{s}_{X+Y}(\mathsf{in}_X(x)) = \mathsf{s}_X(x)$ and $\mathsf{s}_{X+Y}(\mathsf{in}_Y(y)) = \mathsf{s}_Y(y)$.

Limits on the other hand are formed differently because of the finite support map:

▶ **Proposition 4.7.** $\mathsf{Supp}(A)$ *is complete. The limit of a diagram $D\colon \mathcal{D} \to \mathsf{Supp}(A)$ is the subset of finitely supported elements in the limit $\lim VD$ in Set:*

$$\lim D = \{x \in \lim VD \mid \bigcup_{Y \in \mathcal{D}} \mathsf{s}_{DY}(\mathsf{pr}_Y(x)) \text{ is finite}\} \tag{1}$$

The process of restricting to finitely supported elements also happens for limits in nominal sets. For finite limits, this side-condition disappears:

▶ **Corollary 4.8.** $V\colon \mathsf{Supp}(A) \to \mathsf{Set}$ *preserves all finite limits.*

$\mathsf{Supp}(A)$ is cartesian closed, that is, we have an internal hom object. Similarly to nominal set, this internal hom object X^E in $\mathsf{Supp}(A)$ contains more than just the hom set $\mathsf{hom}(E,X)$:

▶ **Definition 4.9.** *For supported sets X and E, let X^E contain those maps $f\colon E \to X$ for which $\mathsf{s}_{X^E}(f) = \bigcup_{e \in E} \mathsf{s}_X(f(e)) \setminus \mathsf{s}_E(e)$ is finite.*

Note that "*map*" really means ordinary maps between sets. Intuitively, a map $f \in X^E$ may introduce finitely many atoms when mapping elements of E to X. For example, if E is a finite supported set, then X^E contains all maps $E \to X$. In contrast, a *supported* map may not introduce any new atoms at all, hence, a map $f\colon E \to X$ is a supported map if and only if $\mathsf{s}_{X^E}(f) = \emptyset$. In particular, $\mathsf{hom}(E, X) \subseteq X^E$.

▶ **Proposition 4.10.** *$(-) \times E$ is left adjoint to $(-)^E$, for all supported sets E.*

In contrast to nominal sets, categorical finiteness notions in $\mathsf{Supp}(A)$ express actual finiteness. For the present paper, we do not need the precise definition of finitely presentable objects and locally finitely presentable (lfp) categories [3, 22]. For the present purposes, it is enough to mention that $\mathsf{Supp}(A)$ is lfp and that the finitely presentable objects are the finite supported sets, i.e. $\mathsf{Supp}(A)$ is locally finite. Detailed definitions can be found in the proof.

▶ **Proposition 4.11.** *$\mathsf{Supp}(A)$ is lfp and finite presentability is actual finiteness.*

4.2 Injectivity, Surjectivity, Quasitopoi

Notions of surjective and injective maps generalize from Set:

▶ **Lemma 4.12.** *Monomorphisms in $\mathsf{Supp}(A)$ are precisely the injective supported maps and epimorphisms are precisely the surjective supported maps.*

However, not all bijective supported maps are isomorphisms in $\mathsf{Supp}(A)$, because they may drop atoms. That is, the difference between bijections and isomorphisms is the following:

▶ **Definition 4.13.** *A map $f\colon X \to Y$ is called* support-reflecting *if $\mathsf{s}_Y \cdot f = \mathsf{s}_X$.*

▶ **Lemma 4.14.** *The isomorphisms in $\mathsf{Supp}(A)$ are the support-reflecting bijective maps.*

▶ **Example 4.15.** The unit $\eta_A\colon A \to JVA$ of the above adjunction $V \dashv J$ to Set is a bijective supported map but not support-reflecting, because $\mathsf{s}_A(a) = \{a\}$ and $\mathsf{s}_{JVA}(a) = \emptyset$. Thus, it is not an isomorphism in $\mathsf{Supp}(A)$.

This shows that $\mathsf{Supp}(A)$ is not a topos – in contrast to Set and Nom. As we will see, it is a *quasitopos* [26, Def. 2.6.1], which entails that $\mathsf{Supp}(A)$ is locally cartesian closed and that it has a subobject classifier for *regular* monomorphisms. The precise definition of regularity is not relevant here (but cf. [2, Rem 7.76(2)]), since it can be nicely characterized via support-reflection:

▶ **Lemma 4.16.** *A monomorphism is regular iff it is support-reflecting. Moreover, $t\colon 1 \to 2$, $0 \mapsto 1$ is a regular-subobject classifier, i.e. for every supported set X, the support-reflecting monomorphisms $m\colon S \to X$ are in correspondence to characteristic maps $\chi_S\colon X \to 2$.*

In summary, for every set A, $\mathsf{Supp}(A)$ is (co)complete, cartesian closed, and locally finite, and we can express many of the above-mentioned properties in one line:

▶ **Theorem 4.17.** *$\mathsf{Supp}(A)$ is a quasitopos (for both [26, Def. 2.6.1] and [2, Def. 28.7]).*

Thus, $\mathsf{Supp}(A)$ constitutes a rich basis to study algebraic theories on it, such as nominal sets.

5 Monadicity of Nominal M-Sets

If a category is *monadic*, then intuitively, it is a well-behaved class of algebras. If so, it becomes applicable for many results and constructions, e.g. regarding representation and the generalized powerset construction. Algebraic theories have been studied extensively throughout the decades and are in correspondence to monads: given a monad T on a category $\mathcal{C}$, e.g. Set, its *Eilenberg-Moore category* $\mathsf{EM}(T)$ contains the models of the algebraic theory defined by T (see e.g. [4, 10.3]). The forgetful functor $U\colon \mathsf{EM}(T) \to \mathcal{C}$ is a right-adjoint functor and its left-adjoint $F\colon \mathcal{C} \to \mathsf{EM}(T)$ sends an object $X \in \mathcal{C}$ (e.g. a set of generators) to the *free algebra* on X:

$$F\colon \mathsf{Supp}(A) \to \mathsf{EM}(T) \quad \dashv \quad U\colon \mathsf{EM}(T) \to \mathsf{Supp}(A) \qquad (\text{for } \mathcal{C} = \mathsf{Supp}(A)) \tag{2}$$

The category of M-sets is monadic over Set, but $\mathsf{Nom}(M)$ fails to be monadic over Set in the instances of interest (Example 3.3). The reason for this is that infinite products in $\mathsf{Nom}(M)$ are different than in Set [38], so the forgetful functor $\mathsf{Nom}(M) \to \mathsf{Set}$ is not right-adjoint and therefore not monadic. As we will show, $\mathsf{Nom}(M)$ is still monadic, but *over supported sets*.

▶ **Definition 5.1.** *A functor $U\colon \mathcal{D} \to \mathcal{C}$ is called* monadic (over $\mathcal{C}$) *if there is a monad T on $\mathcal{C}$ such that $\mathcal{D}$ is $\mathsf{EM}(T)$ and U is the forgetful functor.*

We first show that $U\colon \mathsf{Nom}(M) \to \mathsf{Supp}(A)$ is right-adjoint and then that it is monadic. Then, we can view nominal sets as algebras over $\mathsf{Supp}(A)$. There, the algebraic operations come from the following (supported) set:

▶ **Definition 5.2.** *For $M \leq A^A$ and a set $S \subseteq A$, put $[m]_S$ for the $\approx_S$-equivalence class of m (cf. Definition 3.5) and $M/S = \{[m]_S \mid m \in M\}$ for the supported set of equivalence classes with $\mathsf{s}_{M/S}([m]_S) := m[S] = \{m(a) \mid a \in S\}$.*

We can consider M/S either as equivalence classes of M-elements that are identical on S or alternatively as special maps $S \to A$ that are obtained by restricting some $m \in M \subseteq A^A$ to $S \subseteq A$. Intuitively, M/S is the free nominal set on one generator with support S. Hence, the free nominal M-set over a supported set X is the union of multiple M/S:

▶ **Definition 5.3.** *Fix the functor $M\bullet\colon \mathsf{Supp}(A) \to \mathsf{Supp}(A)$ by $M\bullet X = \coprod_{x \in X} M/\mathsf{s}(x)$.*

$M\bullet$ is the monad that will define nominal M-sets as an algebraic theory. Written with elements, we have $M\bullet X = \{([m]_{\mathsf{s}_X(x)}, x) \mid x \in X\}$ with $\mathsf{s}_{M\bullet X}([m]_{\mathsf{s}(x)}, x) = m[\mathsf{s}(x)]$. For a supported set X, every such element $([m]_{\mathsf{s}(x)}, x)$ of the nominal set $M\bullet X$ is equivalently an element $x \in X$ together with an $|\mathsf{s}(x)|$-tuple of atoms. Of course, such an equivalence class $[m]_{\mathsf{s}(x)}$ is not an arbitrary tuple of elements of A but only one that is obtained from restricting some $m \in M \subseteq (A \to A)$ to $\mathsf{s}_X(x) \subseteq A$. For example, for the equality symmetry $M := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})$, such a $t \in M/\mathsf{s}_X(x)$ is a tuple of *distinct* elements of A.

▶ **Proposition 5.4.** *$M\bullet X$ with the M-set action $\ell \cdot ([m]_{\mathsf{s}(x)}, x) = ([\ell \cdot m]_{\mathsf{s}(x)}, x)$ is nominal and gives rise to a functor $F\colon \mathsf{Supp}(A) \to \mathsf{Nom}(M)$, $FX = M\bullet X$.*

Note that there is no statement about the least finite support of $([m]_{\mathsf{s}(x)}, x) \in M\bullet X$. In the proof, we show that it is supported by $m[\mathsf{s}(x)]$, but the least support might be smaller. Nevertheless, the existence of finite supports in $M\bullet X$ suffices to show the adjunction $F \dashv U$:

▶ **Proposition 5.5.** *If $M \leq A^A$ admits least supports, then $F\colon \mathsf{Supp}(A) \to \mathsf{Nom}(M)$ is left-adjoint to $U\colon \mathsf{Nom}(M) \to \mathsf{Supp}(A)$ with unit $\eta_X\colon X \to UFX$, $\eta_X(x) = ([\mathsf{id}_A]_{\mathsf{s}(x)}, x)$.*

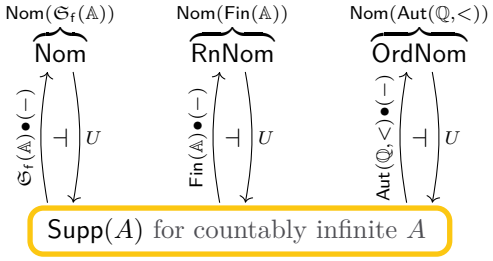


Figure 1 Monadic adjunctions (Ex. 5.10).

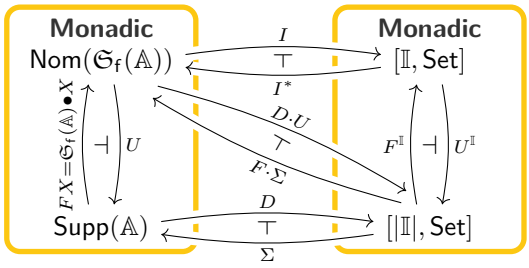


Figure 2 Relation to presheaves for $M := \mathfrak{G}_{\mathsf{f}}(\mathbb{A})$.

This adjunction shows that every nominal M-set is an Eilenberg-Moore algebra and that every supported set generates a free nominal M-set satisfying a universal mapping property. The remaining direction to prove is that every Eilenberg-Moore algebra for $M\bullet$ is indeed a nominal set, or concretely, that the adjunction is monadic. For doing so, we impose a property on the monoid M of interest, which will not only be used in the monadicity proof but also help us to characterize the least finite supports of the free nominal sets:

▶ **Definition 5.6** [7, Def. 9.6]**.** *A monoid $M \leq A^A$ is called* fungible *if for all finite $R \subseteq A$ and $a \in A \setminus R$, there is some $\ell \in M$ with $\ell \approx_R \mathsf{id}_A$ and $\ell(a) \neq a$.*

Intuitively, the condition expresses that the atoms in the support can be renamed independently of each other: if an element is supported by $R \cup \{a\}$, we can always rename a to something fresh while keeping the rest of the support fixed.

▶ **Example 5.7.** All the leading examples (Ex. 3.3) have fungible monoids.

- For $M := \mathfrak{G}_{\mathsf{f}}(\mathbb{A})$ and $M := \mathsf{Fin}(\mathbb{A})$, consider a finite $R \subseteq \mathbb{A}$ and $a \# R$. Then for $b \# a, R$, the permutation $\ell = (a\, b)$ fulfils the desired property $(a\, b) \approx_R \mathsf{id}_{\mathbb{A}}$.
- For $M := \mathsf{Aut}(\mathbb{Q}, <)$, the verification uses a notion called homogeneity [47, Lemma 5.2].

▶ **Lemma 5.8.** *If M is fungible, then $M\bullet X$ is a nominal M-set with $\mathsf{supp}([m]_{\mathsf{s}(x)}, x) = m[\mathsf{s}(x)]$ for every $m \in M$ and $x \in X$.*

We prove the adjunction to be monadic via Beck's theorem [32, Sec. VI.7], which will provide us with the monadicity of the leading examples of nominal sets by instantiating M.

▶ **Theorem 5.9.** *$U\colon \mathsf{Nom}(M) \to \mathsf{Supp}(A)$ is monadic and $\mathsf{Nom}(M) = \mathsf{EM}(M\bullet(-))$, for every fungible M admitting least supports.*

▶ **Example 5.10.** The following categories are monadic over $\mathsf{Supp}(A)$, for A being countably infinite. The monadic adjunctions are visualized in Figure 1, and the monads listed below.

1. The category Nom of nominal sets (with equality symmetry) is monadic over $\mathsf{Supp}(\mathbb{A})$. The operations on a generator x in the corresponding theory are injective maps $\mathsf{s}(x) \rightarrowtail \mathbb{A}$:

$$\mathfrak{G}_{\mathsf{f}}(\mathbb{A})\bullet X = \left\{(\pi, x) \mid x \in X, \pi\colon \mathsf{s}_X(x) \rightarrowtail \mathbb{A}\right\}$$

2. The category RnNom of nominal renaming sets is monadic over $\mathsf{Supp}(\mathbb{A})$. The operations on a generator x are arbitrary maps $\mathsf{s}(x) \to \mathbb{A}$ (not necessarily injective):

$$\mathsf{Fin}(\mathbb{A})\bullet X = \left\{(\pi, x) \mid x \in X, \pi\colon \mathsf{s}_X(x) \to \mathbb{A}\right\}$$

3. The category OrdNom of nominal sets for the total order symmetry is monadic over $\mathsf{Supp}(\mathbb{Q})$ (using $\mathbb{Q} \cong A$). The operations on x are monotone injective maps $\mathsf{s}(x) \rightarrowtail \mathbb{Q}$:

$$\mathsf{Aut}(\mathbb{Q},<)\bullet X = \big\{(\pi, x) \mid x \in X, \pi\colon \mathsf{s}_X(x) \rightarrowtail \mathbb{Q}\ \forall q,p \in \mathsf{s}_X(x) : q < p \Rightarrow \pi(q) < \pi(p)\big\}$$

As a direct application of the monadicity, we can characterize orbit-finite nominal sets and finitely presentable objects in $\mathsf{Nom}(M)$ using a general result about algebraic categories [1, Thm. 3.7]:

▶ **Example 5.11.** A nominal M-set X is finitely presentable iff it can be described by a finite supported set G of *generators* and a finite subset $E \subseteq (M\bullet G) \times (M\bullet G)$ of *equations*. This characterization means that given such finite G and E, we obtain projection maps

$$E \underset{r}{\overset{\ell}{\rightrightarrows}} M\bullet G \qquad \text{in } \mathsf{Supp}(A) \qquad \Longleftrightarrow \qquad M\bullet E \underset{\bar{r}}{\overset{\bar{\ell}}{\rightrightarrows}} M\bullet G \qquad \text{in } \mathsf{Nom}(M)$$

and their coequalizer in $\mathsf{Nom}(M)$ is X.

For $M := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})$, 1. every orbit-finite nominal set can be described by such finite supported sets G and E, and 2. any such finite system of equations E on G presents an orbit-finite nominal set. For example, the nominal set of unordered pairs of atoms can be described by one generator g encoded as a supported set $G = \{g\}, \mathsf{s}_G(g) := \{a, b\}$ (for fixed $a, b \in \mathbb{A}$) and one equation $E := \big\{(a\,b)\cdot g = \mathsf{id}\cdot g\big\}$. Here, we use intuitive notation to denote

$$E := \big\{\big(([(a\,b)]_{\{a,b\}}, g),\ ([\mathsf{id}]_{\{a,b\}}, g)\big)\big\} \subseteq (M\bullet G) \times (M\bullet G).$$

With the projections $\ell, r\colon E \to M\bullet G$ (in $\mathsf{Supp}(A)$) and their extensions $\bar{\ell}, \bar{r}$ to Nom, we have the unordered pairs as a coequalizer diagram in Nom:

$$M\bullet E \underset{\bar{r}}{\overset{\bar{\ell}}{\rightrightarrows}} M\bullet G \longrightarrow\!\!\!\!\!\rightarrow \big\{\{c, d\} \mid c, d \in \mathbb{A}, c \neq d\big\}.$$

▶ Remark 5.12 (Relation to Presheaves). The category of supported sets $\mathsf{Supp}(\mathbb{A})$ nicely fits into an existing diagram of Kurz, Petrisan, and Velebil [31] relating nominal sets Nom (for the equality symmetry $\mathfrak{S}_{\mathsf{f}}(\mathbb{A})$) with two presheaf categories:

1. $[\mathbb{I}, \mathsf{Set}]$ is the category of functors $P\colon \mathbb{I} \to \mathsf{Set}$ (*sets in context*), where the objects of $\mathbb{I}$ are finite subsets of $\mathbb{A}$ (i.e. $|\mathbb{I}| = \mathcal{P}_{\mathsf{f}}\mathbb{A}$) and the morphisms are injective maps (i.e. $\mathbb{I} \neq (\mathcal{P}_{\mathsf{f}}\mathbb{A}, \subseteq)$).
2. $[|\mathbb{I}|, \mathsf{Set}]$ is the category of functors $P\colon |\mathbb{I}| \to \mathsf{Set}$, from the set $\mathcal{P}_{\mathsf{f}}(\mathbb{A})$ to Set, i.e. P is a $\mathcal{P}_{\mathsf{f}}(\mathbb{A})$-indexed family of sets.

Figure 2 shows the result when extending the diagram of Kurz et al. [31] by $\mathsf{Supp}(\mathbb{A})$. Let us go through the functors and adjunctions relating the categories:

1. The left adjoint $\Sigma\colon [|\mathbb{I}|, \mathsf{Set}] \to \mathsf{Supp}(\mathbb{A})$ sends a family $X\colon \mathcal{P}_{\mathsf{f}}(\mathbb{A}) \to \mathsf{Set}$ to the coproduct $\Sigma(X) = \coprod_{S \in \mathcal{P}_{\mathsf{f}}(\mathbb{A})} X(S)$, where the component for $S \in \mathcal{P}_{\mathsf{f}}(\mathbb{A})$ has support S.
2. The right-adjoint $D\colon \mathsf{Supp}(\mathbb{A}) \to [|\mathbb{I}|, \mathsf{Set}]$ forms down-sets: for a supported set X, the family $DX\colon \mathcal{P}_{\mathsf{f}}\mathbb{A} \to \mathsf{Set}$ is given by $DX(S) = \{x \in X \mid \mathsf{s}_X(x) \subseteq S\}$.
3. Since adjunctions compose, we have the adjunction $F\cdot\Sigma \dashv D\cdot U$, which is *not monadic* [31], so the monad $DUFR$ does not have Nom as its Eilenberg-Moore category.
4. Instead, $[\mathbb{I}, \mathsf{Set}] = \mathsf{EM}(DUFR)$ [31], and $F^{\mathbb{I}} \vdash U^{\mathbb{I}}$ is the corresponding adjunction.
5. The induced comparison functor from Nom is itself adjoint: Nom is (equivalent to) the full reflective subcategory of pullback preserving functors (cf. [23, 19]).

Note that $\mathsf{Supp}(\mathbb{A})$ is the only category in Figure 2 that is not a topos, and therefore not a presheaf category.

▶ Remark 5.13 (Relation to named sets). Another notion to reason about names are *named sets* [15], which have been defined in slightly different ways over the years. For a fixed countably infinite set of names $\mathbb{A} = \{v_1, v_2, \ldots\}$ the definitions are as follows:

1. In the definition by Ferrari, Montanari, Pistore [15, Def. 2], every element x of a named set X does not have an explicit support, but is only associated with a natural number $n \in \mathbb{N}$, and so its support is implicitly considered to be $\{v_1, \ldots, v_n\} \subseteq \mathbb{A}$. Moreover, every element is equipped with a subgroup of $\mathfrak{S}_\mathsf{f}(n)$ that describes how the atoms $v_1, \ldots, v_n$ in the support may be permuted. In later works, this natural number $n \in \mathbb{N}$ denoting only the size was replaced with a set of atoms:
2. Montanari and Pistore [36] define a named set to be a set X together with a map $n_X \colon X \to \mathcal{P}(\mathbb{A})$ (without any further information about permutations). A *named function* $m \colon (X, n_X) \to (Y, n_Y)$ is a map on the sets $m \colon X \to Y$ and for each $x \in X$ an injective map $m[x] \colon n_Y(f(x)) \rightarrowtail n_X(x)$. A named set (X, n_X) is called *finitely named* if $n_X(x)$ is finite for every $x \in X$.
   The category of all named sets and named functions is quite different from $\mathsf{Supp}(\mathbb{A})$. For instance, the finitely named set $(\mathbb{A}, n_\mathbb{A})$ with $n_\mathbb{A}(a) = \{a\}$ has infinitely many automorphisms $(\mathbb{A}, n_\mathbb{A}) \to (\mathbb{A}, n_\mathbb{A})$, whereas both in $\mathsf{Nom}(\mathfrak{S}_\mathsf{f}(\mathbb{A}))$ and in $\mathsf{Supp}(\mathbb{A})$, there is only one morphism $\mathbb{A} \to \mathbb{A}$, the identity. Without the finiteness restriction, infinite limits in the category of named sets are Set-like and not Nom-like. More precisely, the infinite product $\prod_{n \in \mathbb{N}} (\mathbb{A}, n_\mathbb{A})$ contains streams making use of infinitely many names.
3. Gaducci, Miculan, and Montanari [23] restrict to *finitely* named sets and additionally equip every element $x \in X$ of a named set (X, n_X) with a subgroup $G_X(x)$ of the group $\mathfrak{S}_\mathsf{f}(n_X(x))$ of all permutations on the names $n_X(x)$. Here, a named function $f \colon (X, n_X, G_X) \to (Y, n_Y, G_Y)$, is a map $f \colon X \to Y$ and additionally for each $x \in X$ a non-empty set of injections $f_x \subseteq (n_Y(f(x)) \rightarrowtail n_X(x))$ satisfying an additional coherence condition involving G_X. Details are not relevant here, because the resulting category NSet was shown to be equivalent to $\mathsf{Nom}(\mathfrak{S}_\mathsf{f}(\mathbb{A}))$ in the same work [23, Prop. 29]. This has a couple of immediate implications regarding the relationship to supported sets:
   - NSet is monadic over $\mathsf{Supp}(\mathbb{A})$.
   - The right-adjoint of this monadic adjunction is a faithful functor $\mathsf{Supp}(\mathbb{A}) \to \mathsf{NSet}$, which is not full.

Having discussed categorical properties of supported sets and its relation to other name-aware notions, we now continue to see how name binding can be accomplished.

6 Name Abstraction and de Bruijn indices

In λ-calculus, the computational steps (β-reduction) essentially consist of a substitution rule $(\lambda x.\, T)\, P \longrightarrow_\beta T[x := P]$ which requires some bound variable names in T to be sufficiently fresh. Thus, it might be necessary to rename those bound variables ($\to_\alpha$) in T before a β step can be performed. But even in such a renaming step $\to_\alpha$, a similar side-condition needs taken care of, because otherwise, the reduction would lead to wrong results.

Thus, there are several approaches to define λ-expressions directly modulo α-equivalence, making substitution total and β-reduction always applicable to λ-expressions containing a reducible expression (redex). In 1972, de Bruijn [11] invented a technique of replacing the variable names with an index (the *de Bruijn index*) that counts the number binders between the variable and its corresponding binder. In 1999, Gabbay and Pitts [19] presented another way to define λ-expressions directly as α-equivalence classes, in which λ-abstraction is a functor on *nominal sets* (called *FM-sets* back then). From now on, we stick to their setting of equality symmetry:

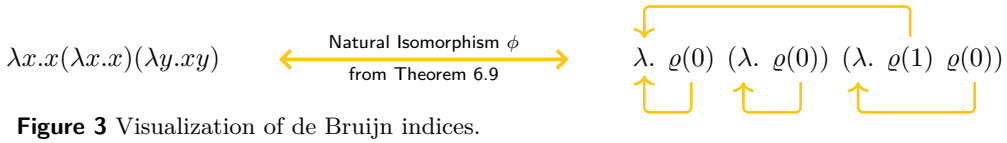


Figure 3 Visualization of de Bruijn indices.

▶ **Assumption 6.1.** *For the rest of the paper, fix* $A := \mathbb{A}$ *and* $M := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})$*. Hence, we may assume a bijection* $\varrho\colon \mathbb{N} \to \mathbb{A}$ *and we simply write* Nom *for the* M*-nominal sets.*

For the name abstraction functor, the notion of α-equivalence is first defined for arbitrary nominal sets, which is then used in the definition of the abstraction functor:

▶ **Definition 6.2** [19]. *For* $X \in \mathsf{Nom}$*, the equivalence relation* $\sim_\alpha$ *on* $\mathbb{A} \times X$ *is defined by*

$$(a, x) \sim_\alpha (b, y) \quad :\Leftrightarrow \quad \exists c \# (a, b, x, y)\colon (c\,a) \cdot x = (c\,b) \cdot y$$

The abstraction functor $[\mathbb{A}]X\colon \mathsf{Nom} \to \mathsf{Nom}$ *is given by* $[\mathbb{A}]X = (A \times X)/{\sim_\alpha}$*, where* $\langle a\rangle x$ *denotes the equivalence class of* $(a, x) \in \mathbb{A} \times X$*.*

In the equivalence class $\langle a\rangle x$, a disappears from the support: $\mathsf{supp}_{[\mathbb{A}]X}(\langle a\rangle x) = \mathsf{supp}_X(x)\backslash\{a\}$.

▶ **Example 6.3** [19]. The initial algebra of the functor $\Lambda X = \mathbb{A} + [\mathbb{A}]X + X \times X$ is carried by the nominal set of λ-expressions modulo α-equivalence. The first summand $\mathbb{A}$ describes variables $a \in \mathbb{A}$, the second summand $[\mathbb{A}]X$ describes λ-abstractions $\lambda x.T$, where $x \in \mathbb{A}$ and T is a λ-expression, and the third summand $X \times X$ describes the application $T\,S$ of one λ-expression to one other.

In the present paper, we define an abstraction functor on supported sets for which we require the set of atoms to be a countably infinite set. This functor in fact has a lifting to nominal sets and the lifting is (naturally isomorphic to) the above abstraction functor $[\mathbb{A}]\colon \mathsf{Nom} \to \mathsf{Nom}$ on nominal sets.

The definition of $[\mathbb{A}]$ makes use of the $\mathfrak{S}_{\mathsf{f}}(\mathbb{A})$-action to capture α-equivalence. When introducing abstraction as a functor directly on supported sets, we do not have renaming available, and so we use de Bruijn indices via the bijection $\varrho\colon \mathbb{N} \to \mathbb{A}$ (Assumption 6.1).

▶ **Definition 6.4.** *The* de Bruijn functor $\mathcal{B}\colon \mathsf{Supp}(\mathbb{A}) \to \mathsf{Supp}(\mathbb{A})$ *sends a supported set* X *to the same set* $\mathcal{B}X = X$ *but with a different support function. To distinguish elements of* X *and* $\mathcal{B}X$*, we write* $\lambda.x \in \mathcal{B}X$ *for* $x \in X$ *(i.e.* λ *is a nameless binder). The support on* $\mathcal{B}X$ *is*

$$\mathsf{s}_{\mathcal{B}X}\colon \mathcal{B}X \to \mathcal{P}_{\mathsf{f}}(\mathbb{A}) \qquad \mathsf{s}_{\mathcal{B}X}(\lambda.x) := \{\varrho(k) \mid \varrho(k+1) \in \mathsf{s}_X(x), k \in \mathbb{N}\}. \tag{3}$$

A supported map $f\colon X \to Y$ *is sent to the same map* $\mathcal{B}f\colon \mathcal{B}X \to \mathcal{B}Y$*,* $\mathcal{B}f(\lambda.x) = \lambda.f(y)$*.*

The definition of $\mathsf{s}_{\mathcal{B}X}$ captures the idea of de Bruijn indices: we can think of the notation $\lambda.x$ as a lambda abstraction of nameless binder $\lambda.$ of a lambda term x (visualized in Figure 3):

- The variables referring to $\lambda.$ have the de Bruijn index of 0 (at the level of x), because there is no other binder between x and "$\lambda.$". Since $\varrho(0)$ is bound, $\mathsf{s}_{\mathcal{B}X}(x)$ does not depend on whether $\varrho(0) \in \mathsf{s}_X(x)$.
- All other variables $\varrho(k+1) \in \mathsf{s}_X(x)$ refer to variables that are free in $\lambda.x$ and so refer to binders "more above" than $\lambda.x$. A variable $\varrho(k) \in \mathsf{s}_{\mathcal{B}X}(\lambda.x)$ refers to the same binder as the variable $\varrho(k+1) \in \mathsf{s}_X(x)$ under "$\lambda.$", because the latter is one level further down.

▶ **Lemma 6.5.** $\mathcal{B}\colon \mathsf{Supp}(\mathbb{A}) \to \mathsf{Supp}(\mathbb{A})$ *is a functor.*

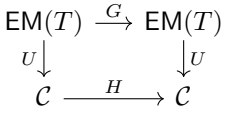


Figure 4 Diagram for Definition 6.6.

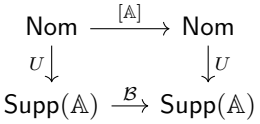


Figure 5 Diagram for Theorem 6.9.

We use a slightly generalized notion of a *lifting* of the functor $\mathcal{B}$ to nominal sets:

▶ **Definition 6.6.** *For a monad T on a category $\mathcal{C}$ and a functor $H\colon \mathcal{C} \to \mathcal{C}$, a functor $G\colon \mathsf{EM}(T) \to \mathsf{EM}(T)$ is called a* lifting of H *if HU and UG are naturally isomorphic functors (see Figure 4 for the diagram). We say that a lifting is* strict *if $HU = UG$.*

▶ Remark 6.7. Usually, not just a natural isomorphism but identity is required. Strict liftings $G\colon \mathsf{EM}(T) \to \mathsf{EM}(T)$ are in one-to-one correspondence to distributive laws $TH \to HT$ [25].

This generalization to natural isomorphisms is sound, because they induce strict liftings:

▶ **Lemma 6.8.** *For every natural isomorphism $\phi\colon HU \to UG$, there is a unique strict lifting $\bar{H}\colon \mathsf{EM}(T) \to \mathsf{EM}(T)$ such that $\phi\colon H \to G$ is a natural isomorphism in $\mathsf{EM}(T)$.*

This means that for $(C, \gamma) \in \mathsf{EM}(T)$, $\phi_{(C,\gamma)}$ is a T-algebra isomorphism $\bar{H}(C,\gamma) \to G(C,\gamma)$.

▶ **Theorem 6.9.** *The abstraction functor $[\mathbb{A}]\colon \mathsf{Nom} \to \mathsf{Nom}$ is a lifting of the de Bruijn functor $\mathcal{B}\colon \mathsf{Supp}(\mathbb{A}) \to \mathsf{Supp}(\mathbb{A})$ (see Figure 5 for the diagram). That is, there is a natural isomorphism $\phi\colon \mathcal{B}U \longrightarrow U[\mathbb{A}]$.*

Intuitively, ϕ translates between the de Bruijn indices and the nominal abstraction functor: in the term $\lambda.x \in \mathcal{B}X$, we have $\varrho(0)$ implicitly bound, like it is in $\langle \varrho(0) \rangle (x) \in [\mathbb{A}]X$. However, every $\varrho(k+1)$ in x appears as $\varrho(k)$ in the support of $\lambda.x$, so ϕ essentially renames $\varrho(\ell+1) \mapsto \varrho(\ell)$ in x for all $\ell \geq 1$.

▶ Remark 6.10 (Abstraction in named sets). A de Bruijn-style abstraction functor can also defined directly on Nom [9, Def. 3.3]. The defining property (3) of abstraction on $\mathsf{Supp}(\mathbb{A})$ then reappears as a theorem of the support function supp in nominal sets [9, Thm. 4.1]. This adheres to the principle that in $\mathsf{Supp}(\mathbb{A})$, the support is part of the data, whereas in nominal sets, supp is a derived notion. Since named sets are equivalent to nominal sets, we have name abstraction there, too, and an explicit definition is provided by Ciancia and Montanari [9, Sect. 7.1].

▶ Remark 6.11 (Abstraction in presheaves). Fiore et al. [16] also use de Bruijn indices in their abstraction functor, but treat support in a different way. They consider presheaves $X \in \mathsf{Set}^{\mathbb{F}}$ where $\mathbb{F}$ is the full subcategory of Set containing only natural numbers as objects (considering natural numbers as finite cardinals). Thus for every natural number $n \in \mathbb{N}$, the set $X(n)$ intuitively denotes the elements that make only use of the atoms $\varrho(0), \ldots, \varrho(n-1) \in \mathbb{A}$. For each map $\pi\colon n \to m$, the map $X(\pi)\colon X(n) \to X(m)$ renames embedded variables. Their name abstraction functor $\delta\colon \mathsf{Set}^{\mathbb{F}} \to \mathsf{Set}^{\mathbb{F}}$ sends a presheaf X to the presheaf $\delta(X)(n) = X(n+1)$, implicitly binding $\varrho(0)$, and "shifting" the role of the other atoms. Despite this similarity, it is unclear whether there is a formal categorical connection between $\mathsf{Supp}(A)$ and $\mathsf{Set}^{\mathbb{F}}$ or between $\mathcal{B}$ and δ.

With name binding in supported sets, we can now study automata in supported sets and relate them to nominal automata.

7 Register Automata to Nominal Automata

The well-known powerset construction for automata is an instance of a more general principle called *generalized determinization* [40, 6] that *internalizes side-effects* modelled by a monad. The input of the standard powerset construction is a *non-deterministic* finite automaton (NFA), which can be understood as a deterministic automaton extended with non-deterministic branching as a side-effect. With Q as the set of states of the NFA, the construction returns a deterministic automaton with states $\mathcal{P}_\mathsf{f}Q$, which happens to be precisely the set of *configurations* of the original NFA. This construction generalizes from the (finite) powerset monad $\mathcal{P}_\mathsf{f}$ to arbitrary monads $T\colon \mathcal{C} \to \mathcal{C}$ and from automata to state-based systems modelled via coalgebras for a functor $H\colon \mathcal{C} \to \mathcal{C}$. We apply this principle to the monad from the monadicity result in Section 5.

A *coalgebra* (for $H\colon \mathcal{C} \to \mathcal{C}$) is an object $Q \in \mathcal{C}$ together with a morphism $Q \to HQ$. E.g. an H-coalgebra for $H\colon \mathsf{Set} \to \mathsf{Set}$, $HX = 2 \times X^\Sigma$, is a deterministic automaton, i.e. a set Q together with a map $d\colon Q \to 2 \times Q^\Sigma$. For a state $q \in Q$, the first component $\mathsf{pr}_1(d(q)) \in 2$ specifies the finality of q, and the second component $\mathsf{pr}_2(d(q)) \in Q^\Sigma$ sends input symbols $a \in \Sigma$ to successor states in Q (the initial state $q_0 \in Q$ is not important here).

On the other hand, a non-deterministic automaton is simply a coalgebra for the composed functor $H\mathcal{P}_\mathsf{f}\colon \mathsf{Set} \to \mathsf{Set}$, i.e. a map $c\colon Q \to 2 \times (\mathcal{P}_\mathsf{f}Q)^\Sigma$. The generalized determinization assumes that $T\colon \mathcal{C} \to \mathcal{C}$ is a monad (with unit η and multiplication μ) and that $H\colon \mathcal{C} \to \mathcal{C}$ is a functor that lifts to the Eilenberg-Moore category of T, i.e. that we have a functor $G\colon \mathsf{EM}(T) \to \mathsf{EM}(T)$ with $\phi\colon HU \xrightarrow{\cong} UG$ (Definition 6.6) – for example $HX = 2 \times X^\Sigma$ and $TX = \mathcal{P}_\mathsf{f}X$. Then the generalized determinization turns an HT-coalgebra on Q into a G-coalgebra on TQ, using the adjunction $F \dashv U$ between $\mathcal{C}$ and $\mathsf{EM}(T)$ (cf (2)):

$$c\colon Q \to HTQ \quad \text{in } \mathcal{C} \qquad \overset{\text{Internalization}}{\longmapsto} \qquad d\colon FQ \to GFQ \quad \text{in } \mathsf{EM}(T)$$

In the instance $TX = \mathcal{P}_\mathsf{f}X$ of the powerset construction, $\mathsf{EM}(T)$ is the category of join-semilattices, and every non-deterministic automaton on Q is turned into a deterministic automaton on $\mathcal{P}_\mathsf{f}Q$. Since we internalized the side-effect of non-deterministic branching in the states, the resulting state space $\mathcal{P}_\mathsf{f}Q$ is the configuration space of the non-deterministic automaton and the induced transition structure d preserves joins. Instantiating the generalized determinization to supported sets and nominal sets yields a construction that internalizes the side-effect of rearranging atoms or registers. Here, we stick to the equality symmetry $M := \mathfrak{S}_\mathsf{f}(\mathbb{A})$, $TX := \mathfrak{S}_\mathsf{f}(\mathbb{A})\bullet X$, i.e. we stick to Assumption 6.1.

▶ **Construction 7.1.** *Fix $M := \mathfrak{S}_\mathsf{f}(\mathbb{A})$. For functors $H\colon \mathsf{Supp}(\mathbb{A}) \to \mathsf{Supp}(\mathbb{A})$ and $G\colon \mathsf{Nom} \to \mathsf{Nom}$ with $HU \cong UG$, we have the internalization process*

$$c\colon Q \to H(M\bullet Q) \ \textit{ in } \mathsf{Supp}(\mathbb{A}) \qquad \overset{\text{Internalization}}{\longmapsto} \qquad d\colon M\bullet Q \to G(M\bullet Q) \ \textit{ in } \mathsf{Nom}.$$

Here, the nominal set $M\bullet Q$ is the configuration space (states + register assignments) of the original automaton (Q, c). The configuration $\mathsf{id}_A \cdot q$ in $M\bullet Q$ behaves like $q \in Q$ in c, and the resulting transition structure d is equivariant. We can apply this construction to many different system types H and G that arise from the following functors. Here, $\mathcal{P}_{\mathsf{ufs}}X$ (for a nominal set X) contains those subsets $S \subseteq X$ for which $\bigcup\{\mathsf{supp}(x) \mid x \in S\}$ is finite [39].

▶ **Proposition 7.2.** *The functors G on Nom that are the lifting of a functor H on $\mathsf{Supp}(\mathbb{A})$ contain $\mathcal{P}_\mathsf{f}$, $\mathcal{P}_{\mathsf{ufs}}$, $[\mathbb{A}]$, and all constant functors and are closed under all (possibly infinite) products and coproducts.*

▶ **Example 7.3.** All Nom-functors arising from binding signatures [30, 16] are such liftings. In particular, $\Lambda X = \mathbb{A} + [\mathbb{A}]X + X \times X$ (Example 6.3) is the lifting of $HX = \mathbb{A} + \mathcal{B}X + X \times X$. The coalgebras of Λ are possibly infinite λ-trees modulo α-equivalence [30]. Such a λ-tree can then be represented by a supported set X and a supported map

$$f \colon X \to \mathbb{A} \;+\; \mathcal{B}(\mathfrak{S}_{\mathsf{f}}(\mathbb{A}) \bullet X) \;+\; (\mathfrak{S}_{\mathsf{f}}(\mathbb{A}) \bullet X) \times (\mathfrak{S}_{\mathsf{f}}(\mathbb{A}) \bullet X)$$

▶ **Example 7.4.** Nominal automata with name binding considered by Schröder et al. [39] are coalgebras for the Nom-functor $KX = 2 \times \mathcal{P}_{\mathsf{ufs}}([\mathbb{A}]X + \mathbb{A} \times X)$ which is a lifting of $HX = 2 \times \mathcal{P}_{\mathsf{ufs}}(\mathcal{B}X + \mathbb{A} \times X)$ on supported sets. Thus, these nominal automata can be represented by finite coalgebras in $\mathsf{Supp}(\mathbb{A})$.

Interpreting $\mathbb{A}$ as the names of registers, register automata in the style of Cassel et al. [8] straightforwardly adapt to HT-coalgebras in supported sets:

▶ **Example 7.5.** Let $\mathcal{G}$ be a nominal set of *guards*, where we understand $g \in \mathcal{G}$ as a predicate involving register names $\mathsf{supp}_G(g) \in \mathcal{P}_{\mathsf{f}}(\mathbb{A})$, e.g. $\mathsf{iszero}(a)$, $\mathsf{divides}(a, b)$, $\mathsf{plus}(a, b, c)$. It is important that the atoms $a \in \mathbb{A}$ are the register names, and not the data itself. So e.g. $\mathsf{iszero}(a_3)$ is satisfied if the data value stored in register a_3 is zero – this is the *symbolic* semantics of register automata [44, 45]. Using the forgetful $U \colon \mathsf{Nom} \to \mathsf{Supp}(\mathbb{A})$, we can also use $\mathcal{G}$ as a supported set where the support of $g \in \mathcal{G}$ is the set of register names $a \in \mathbb{A}$ appearing in g. Now, a register automaton is a HT-coalgebra in $\mathsf{Supp}(A)$

$$c \colon Q \to 2 \times \mathcal{B}\mathcal{P}_{\mathsf{f}}(U\mathcal{G} \times \mathfrak{S}_{\mathsf{f}}(\mathbb{A}) \bullet Q) \qquad \text{for } \; HX = 2 \times \mathcal{B}\mathcal{P}_{\mathsf{f}}(\mathcal{G} \times X) \;\text{ and }\; TX = \mathfrak{S}_{\mathsf{f}}(\mathbb{A}) \bullet X.$$

As before, the first component of $c(q)$ defines the finality of a state $q \in Q$. While in state q, the registers $\mathsf{s}(q) \subseteq \mathbb{A}$ are filled with data and the second component of $c(q)$ binds the next input data symbol into a fresh register ($\mathcal{B}$) and then provides a finite number of transitions of the form $q \xrightarrow{g,\pi} q'$. In such a transition, the guard $g \in \mathcal{G}$, specifies when this transition can be taken, depending on the register contents of q and the freshly read input symbol. When a transition is taken, the registers are rearranged via the permutation π before state q' is entered. The map $\pi \colon \mathsf{s}(q') \rightarrowtail \mathbb{A}$ specifies for each register $r \in \mathsf{s}(q')$ of the target set, where the data symbol is drawn from. The map c being a supported map ensures that whenever we transfer to state q', then all registers $\mathsf{s}(q')$ can be filled with data, coming for the support of the previous state q or from the "input register" in $\mathcal{B}$ (a concrete register automaton is discussed in Example 7.6). The internalization process turns c into an equivariant map $d \colon \bar{Q} \to 2 \times [\mathbb{A}]\mathcal{P}_{\mathsf{f}}(\mathcal{G} \times \bar{Q})$, which is the nominal automaton of configurations $\bar{Q} = \mathfrak{S}_{\mathsf{f}}(\mathbb{A}) \bullet Q$.

The construction also nicely interacts with *initial* states. An initial state of a register automaton is simply a supported map $i \colon 1 \to Q$, that is, an element of Q with empty support. Applying the functor $\mathfrak{S}_{\mathsf{f}}(\mathbb{A}) \bullet$ to i yields an equivariant map, $\mathfrak{S}_{\mathsf{f}}(\mathbb{A}) \bullet i \colon \mathfrak{S}_{\mathsf{f}}(\mathbb{A}) \bullet 1 \to \bar{Q}$. Since the only element of $1 = \{0\}$ has empty support, we have $\mathfrak{S}_{\mathsf{f}}(\mathbb{A}) \bullet 1 \cong 1$, so $\mathfrak{S}_{\mathsf{f}}(\mathbb{A}) \bullet i$ is equivalent to an equivariant map $i' \colon 1 \to \bar{Q}$.

▶ **Example 7.6.** An example register automaton is visualized in Figure 6: the upper half of the nodes provide the state names $Q = \{q_0, q_1, q_2\}$, the lower half specifies their support $\mathsf{s}_Q(q_0) = \emptyset$, $\mathsf{s}_Q(q_1) = \{\ell\}$, $\mathsf{s}_Q(q_2) = \{s\}$, where $\ell, s \in \mathbb{A}, \ell \neq s$ are arbitrary (standing for ℓast and $s$econd last). The initial state is q_0, i.e. the supported map $i \colon 1 \to Q$ is $i(0) = q_0$, and all states are final.We mimic existing register automata notation [8, 44] by defining $p := \varrho(0) \in \mathbb{A}$ (note that we do *not* require ℓ, s to be distinct from p). Also, we use $\mathsf{old} \colon \mathbb{A} \to \mathbb{A}$ defined by $\mathsf{old}(\varrho(k)) = \varrho(k+1)$, which satisfies $a \in \mathsf{s}_{\mathcal{B}X}(\lambda.x)$ iff $\mathsf{old}(a) \in \mathsf{s}_X(x)$ for all supported sets X and $x \in X$. The nominal set of guards is defined by

$$\mathcal{G} := \{\top\} + \{\mathsf{plus}\} \times \mathbb{A}^3,$$

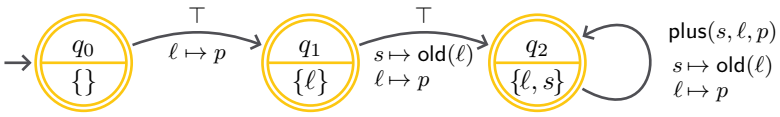


Figure 6 Example of a (symbolic) register automaton.

so the constant $\top$ represents "true" and the ternary relation symbol $\mathsf{plus}(a, b, c)$ represents that the sum of the register contents of a and b equals the register content of c. Since we use symbolic semantics, the concrete data domain does not need to be specified here; but of course one can interpret plus over rational numbers for example. The supported map $c\colon Q \to 2 \times \mathcal{BP}_{\mathsf{f}}(U\mathcal{G} \times \mathfrak{S}_{\mathsf{f}}[\mathbb{A}]Q)$ representing the automaton is sincerely visualized in Figure 6:

- The transition $q_0 \xrightarrow{g,\pi} q_1$ has the guard $g = \top$ and the register reassignment $\pi\colon \mathsf{s}(q_1) \rightarrowtail \mathbb{A}$ is defined by $\pi(\ell) = p$, meaning that when entering state q_1, the register ℓ will be filled with what was in the input p before: $c(q_0) = (1, \lambda.\{(\top, (\pi, q_1))\})$. This satisfies support preservation because $\mathsf{s}(\top, (\pi, q_1)) = \pi[\mathsf{s}(q_1)] = \{p\}$ and so $\mathsf{s}(\lambda.(\top, (\pi, q_1))) = \emptyset \subseteq \mathsf{s}(q_0)$.
- The transition $q_1 \xrightarrow{g,\sigma} q_2$ has the same guard $g = \top$ and $\sigma\colon \mathsf{s}(q_2) \rightarrowtail \mathbb{A}$ is defined by $\sigma(s) = \mathsf{old}(\ell)$ and $\sigma(\ell) = p$. Again, $c(q_1) = (1, \lambda.\{(\top, (\sigma, q_2))\})$ preserves support because

$$\mathsf{s}(\top, (\sigma, q_2)) = \sigma[\mathsf{s}(q_2)] = \sigma[\{\ell, s\}] = \{p, \mathsf{old}(\ell)\} \quad \text{and} \quad \mathsf{s}(\lambda.\{(\top, (\sigma, q_2))\}) = \{\ell\} \subseteq \mathsf{s}(q_1).$$

- The loop $q_2 \xrightarrow{g',\sigma} q_2$ has the guard $g' = \mathsf{plus}(s, \ell, p)$ and literally the same $\sigma\colon \mathsf{s}(q_2) \rightarrowtail \mathbb{A}$ as in the previous transition. Support preservation holds for the mapping $c(q_1) = (1, \lambda.\{(\top, (\sigma, q_2))\})$, because

$$\mathsf{s}(\mathsf{plus}(s, \ell, p), (\sigma, q_2)) = \{\mathsf{old}(s), \mathsf{old}(\ell), p\} \text{ and } \mathsf{s}(\lambda.\{(\mathsf{plus}(s, \ell, p), (\sigma, q_2))\}) = \{s, \ell\} \subseteq \mathsf{s}(q_2).$$

The coherence axioms of register automata naturally translate into c being a supported map. Construction 7.1 transforms this finite coalgebra in $\mathsf{Supp}(\mathbb{A})$ into an nominal automaton, in the style of symbolic semantics [44] of register automata.

Not only in this example, but also in general, Construction 7.1 preserves finite presentability. So every finite coalgebra in $\mathsf{Supp}(\mathbb{A})$ is turned into an orbit-finite coalgebra in Nom. The semantics of orbit-finite coalgebras can be characterized for many functors [33] that arise from the examples listed here (Proposition 7.2).

8 Conclusions and Future Work

We have seen that by going from the base category of sets to supported sets, nominal sets for various symmetries surprisingly turn out to be monadic. Supported sets have a functor for name binding, which even lifts to the abstraction functor in nominal sets. It remains for future work whether a similar name binding functor can be found for other data alphabets, most notably for the total order symmetry on $\mathbb{Q}$, and whether multiple atoms can be bound simultaneously, as it is possible in nominal sets [10]. It can be conjectured that such generalizations are not possible on the level of supported sets.

On the positive side, due to the little structure of supported sets, it provides a common foundation for the nominal sets for different symmetries, in the sense of being described by monads on supported sets. The monadicity can be used to relate nominal automata with register automata, which have a natural definition in supported sets. It remains for future investigation how the *data semantics* of register automata can be phrased in supported sets. We are optimistic that it helps to develop a categorical semantics for register automata for data alphabets and signatures beyond symmetries (e.g. those for priority queues [8]). When

developing algorithms, in particular learning algorithms and minimization algorithms, for register or nominal automata [8, 43, 35], supported sets directly yield a finite representation that can help in the implementation and complexity analysis. Also for other algorithmic tasks such as efficient coalgebraic minimization [12, 14, 24, 28, 48, 49], supported sets can help to canonically represent the nominal coalgebras subject to minimization.

References

1 Jiří Adámek, Stefan Milius, Lurdes Sousa, and Thorsten Wißmann. Finitely presentable algebras for finitary monads. *Theory and Applications of Categories*, 34(37):1179–1195, November 2019. URL: http://www.tac.mta.ca/tac/volumes/34/37/34-37abs.html.

2 Jiří Adámek, Horst Herrlich, and George E. Strecker. Abstract and concrete categories. the joy of cats, 2004.

3 Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories*. Cambridge University Press, 1994.

4 Steve Awodey. *Category Theory*. Oxford Logic Guides. OUP Oxford, 2010.

5 Brian E. Aydemir, Aaron Bohannon, and Stephanie Weirich. Nominal reasoning techniques in coq: (extended abstract). *Electron. Notes Theor. Comput. Sci.*, 174(5):69–77, 2007. doi:10.1016/j.entcs.2007.01.028.

6 Falk Bartels. *On generalized coinduction and probabilistic specification formats: Distributive laws in coalgebraic modelling*. PhD thesis, Vrije Universiteit Amsterdam, 2004.

7 Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10, 2014. doi:10.2168/LMCS-10(3:4)2014.

8 Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. Extending automata learning to extended finite state machines. In Amel Bennaceur, Reiner Hähnle, and Karl Meinke, editors, *Machine Learning for Dynamic Software Analysis: Potentials and Limits – International Dagstuhl Seminar 16172*, volume 11026 of *LNCS*, pages 149–177. Springer, 2018. doi:10.1007/978-3-319-96562-8_6.

9 Vincenzo Ciancia and Ugo Montanari. A name abstraction functor for named sets. In Jirí Adámek and Clemens Kupke, editors, *Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science, CMCS 2008*, volume 203, 5 of *Electronic Notes in Theoretical Computer Science*, pages 49–70. Elsevier, 2008. doi:10.1016/j.entcs.2008.05.019.

10 Ranald Clouston. Generalised name abstraction for nominal sets. In Frank Pfenning, editor, *Foundations of Software Science and Computation Structures – 16th International Conference, FOSSACS 2013*, volume 7794 of *Lecture Notes in Computer Science*, pages 434–449. Springer, 2013. doi:10.1007/978-3-642-37075-5_28.

11 N.G de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972. doi:10.1016/1385-7258(72)90034-0.

12 Hans-Peter Deifel, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Generic partition refinement and weighted tree automata. In *Formal Methods – The Next 30 Years, Proc. 3rd World Congress on Formal Methods (FM 2019)*, volume 11800 of *LNCS*, pages 280–297. Springer, October 2019.

13 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. doi:10.1145/1507244.1507246.

14 Ulrich Dorsch, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Efficient coalgebraic partition refinement. In *Proc. 28th International Conference on Concurrency Theory (CONCUR 2017)*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017.

15 Gian Luigi Ferrari, Ugo Montanari, and Marco Pistore. Minimizing transition systems for name passing calculi: A co-algebraic formulation. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 129–158. Springer, 2002. doi:10.1007/3-540-45931-6_10.

16 Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding (extended abstract). In *Proc. 14th LICS Conf.*, pages 193–202. IEEE, Computer Society Press, 1999.

17 Marcelo P. Fiore and Sam Staton. Comparing operational models of name-passing process calculi. *Inf. Comput.*, 204(4):524–560, 2006. doi:10.1016/j.ic.2005.08.004.

18 Murdoch Gabbay and James Cheney. A sequent calculus for nominal logic. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 139–148. IEEE Computer Society, 2004. doi:10.1109/LICS.2004.1319608.

19 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 214–224. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782617.

20 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects Comput.*, 13(3-5):341–363, 2002. doi:10.1007/s001650200016.

21 Murdoch James Gabbay and Martin Hofmann. Nominal renaming sets. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008*, volume 5330 of *LNCS*, pages 158–173. Springer, 2008. doi:10.1007/978-3-540-89439-1_11.

22 Peter Gabriel and Friedrich Ulmer. *Lokal präsentierbare Kategorien*, volume 221 of *Lecture Notes Math.* Springer-Verlag, 1971.

23 Fabio Gadducci, Marino Miculan, and Ugo Montanari. About permutation algebras, (pre)sheaves and named sets. *Higher Order Symbol. Comput.*, 19(2-3):283–304, September 2006. doi:10.1007/s10990-006-8749-3.

24 Jules Jacobs and Thorsten Wißmann. Fast coalgebraic bisimilarity minimization. In *Principles of Programming Languages, POPL '23*. ACM, 2023. to appear. URL: https://arxiv.org/abs/2204.12368.

25 Peter T. Johnstone. Adjoint Lifting Theorems for Categories of Algebras. *Bull. London Math. Soc.*, 7(3):294–297, November 1975.

26 Peter T Johnstone. *Sketches of an elephant: a Topos theory compendium.* Oxford logic guides. Oxford Univ. Press, New York, NY, 2002.

27 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

28 Barbara König and Sebastian Küpper. Generic partition refinement algorithms for coalgebras and an instantiation to weighted automata. In *Theoretical Computer Science, IFIP TCS 2014*, volume 8705 of *LNCS*, pages 311–325. Springer, 2014. doi:10.1007/978-3-662-44602-7.

29 Dexter Kozen, Konstantinos Mamouras, Daniela Petrisan, and Alexandra Silva. Nominal kleene coalgebra. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *ICALP 2015, Proceedings*, volume 9135 of *LNCS*, pages 286–298. Springer, 2015. doi:10.1007/978-3-662-47666-6_23.

30 Alexander Kurz, Daniela Petrisan, Paula Severi, and Fer-Jan de Vries. Nominal coalgebraic data types with applications to lambda calculus. *Logical Methods in Computer Science*, 9(4), 2013. doi:10.2168/LMCS-9(4:20)2013,http://arxiv.org/abs/1311.1395.

31 Alexander Kurz, Daniela Petrisan, and Jiri Velebil. Algebraic theories over nominal sets. *CoRR*, abs/1006.3027, 2010. URL: http://arxiv.org/abs/1006.3027, arXiv:1006.3027.

32 Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer New York, 1998. URL: http://books.google.de/books?id=eBvhyc4z8HQC.

33 Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Regular behaviours with names. *Applied Categorical Structures*, 24(5):663–701, 2016. doi:10.1007/s10485-016-9457-8.

34 Joshua Moerman and Jurriaan Rot. Separation and Renaming in Nominal Sets. In Maribel Fernández and Anca Muscholl, editors, *CSL 2020*, volume 152 of *LIPIcs*, pages 31:1–31:17, Dagstuhl, Germany, 2020. LIPIcs. doi:10.4230/LIPIcs.CSL.2020.31.

35 Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michal Szynwelski. Learning nominal automata. In Giuseppe Castagna and Andrew D. Gordon, editors, *POPL 2017*, pages 613–625. ACM, 2017. doi:10.1145/3009837.3009879.

36 Ugo Montanari and Marco Pistore. History-dependent automata: An introduction. In Marco Bernardo and Alessandro Bogliolo, editors, *SFM-Moby 2005: Formal Methods for Mobile Computing*, volume 3465 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2005. doi:10.1007/11419822_1.

37 Daniela Petrişan. *Investigations into Algebra and Topology over Nominal Sets*. dissertation, University of Leicester, 2011.

38 Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013.

39 Lutz Schröder, Dexter Kozen, Stefan Milius, and Thorsten Wißmann. Nominal automata with name binding. In Javier Esparza and Andrzej Murawski, editors, *FoSSaCS 2017*, volume 10203 of *LNCS*, pages 124–142. Springer, 2017. doi:10.1007/978-3-662-54458-7_8.

40 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing determinization from automata to coalgebras. *Log. Methods Comput. Sci.*, 9(1), 2013. doi:10.2168/LMCS-9(1:9)2013.

41 Sam Staton. Name-passing process calculi: operational models and structural operational semantics. Technical Report UCAM-CL-TR-688, University of Cambridge, Computer Laboratory, June 2007. doi:10.48456/tr-688.

42 Christian Urban and Christine Tasson. Nominal techniques in isabelle/hol. In Robert Nieuwenhuis, editor, *CADE-20*, volume 3632 of *LNCS*, pages 38–53. Springer, 2005. doi:10.1007/11532231_4.

43 Henning Urbat and Lutz Schröder. Automata learning: An algebraic approach. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 900–914. ACM, 2020. doi:10.1145/3373718.3394775.

44 Frits W. Vaandrager and Abhisek Midya. A myhill-nerode theorem for register automata and symbolic trace languages. In Violet Ka I Pun, Volker Stolz, and Adenilso Simão, editors, *ICTAC 2020*, volume 12545 of *LNCS*, pages 43–63. Springer, 2020. doi:10.1007/978-3-030-64276-1_3.

45 Frits W. Vaandrager and Abhisek Midya. A myhill-nerode theorem for register automata and symbolic trace languages. *Theor. Comput. Sci.*, 912:37–55, 2022. doi:10.1016/j.tcs.2022.01.015.

46 David Venhoek, Joshua Moerman, and Jurriaan Rot. Fast computations on ordered nominal sets. In Bernd Fischer and Tarmo Uustalu, editors, *ICTAC 2018*, volume 11187 of *LNCS*, pages 493–512. Springer, 2018. doi:10.1007/978-3-030-02508-3_26.

47 David Venhoek, Joshua Moerman, and Jurriaan Rot. Fast computations on ordered nominal sets. *Theor. Comput. Sci.*, 935:82–104, 2022. doi:10.1016/j.tcs.2022.09.002.

48 Thorsten Wißmann, Hans-Peter Deifel, Stefan Milius, and Lutz Schröder. From generic partition refinement to weighted tree automata minimization. *Formal Aspects of Computing*, pages 1–33, March 2021. doi:10.1007/s00165-020-00526-z.

49 Thorsten Wißmann, Ulrich Dorsch, Stefan Milius, and Lutz Schröder. Efficient and modular coalgebraic partition refinement. *Logical Methods in Computer Science*, 16:1:8:1–8:63, January 2020. doi:10.23638/LMCS-16(1:8)2020.